Хмельницький національний університет

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Гуменюка Іллі Андрійовича

Прізвище, ім'я, по батькові студента(ки)

на здобуття ступення вищої освіти Бакалавра

Вебсистема для автоматизації логістичних операцій

Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101073.01.05.ПЗ

Виконав студент IV курсу група ІПЗ-21-1 _____ Ілля ГУМЕНЮК

Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА

Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер ст. викладач _____ Ганна БЕДРАТЮК

Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:

Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК

Підпис Ім'я, ПРІЗВИЩЕ

3 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
2.01. 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гуменюку Іллі Андрійовичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Вебсистема для автоматизації логістичних операцій

Керівник кваліфікаційної роботи Праворська Наталія Іванівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23
2. Строк подання студентом роботи на кафедру 06.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 15 шт.), діаграма варіантів використання, ER-діаграма, діаграма діяльності, діаграма станів, DFD діаграми, діаграма розгортання

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бедратюк Г.І, ст. викладач		
Антиплагіат	Форкун Ю.В., к.т.н., доцент	12.05.2025	28.05.2025

7. Дата видачі завдання « » 2025р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою кваліфікаційних робіт (КвР), визначення та узгодження індивідуальної теми	01.12 – 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проектування програмного забезпечення	21.02 – 20.03.2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог стандартів	01.05 – 25.05.2025	
6 Попередній захист КвР	Травень 2025	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Студент _____ Гуменюк І.А.
Підпис Ініціали, прізвище

Керівник роботи _____ Граборська Н.Э.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: Програмний застосунок для керування проєктами.

Автор проекту: Гуменюк Ілля Андрійович.

Керівник проекту: Праворська Наталія Іванівна.

Пояснювальна записка: 76 с., 34 рис., 6 табл., 3 дод., 40 джерел.

Графічна частина: 15 слайдів.

СИСТЕМА УПРАВЛІННЯ ТРАНСПОРТОМ ДЛЯ АВТОМАТИЗАЦІЇ ЛОГІСТИЧНИХ ОПЕРАЦІЙ НА LARAVEL ТА VUE.JS

Метою проекту є розробка MVP вебсистеми для підвищення ефективності роботи логістичних компаній, завдяки автоматизації логістичних операцій, а саме відслідковування активних перевезень.

У кваліфікаційній роботі було проведено детальне дослідження предметної області, розроблено технічне завдання, детально спроєктовано програмне забезпечення та база даних, розроблено і протестовано.

Для розробки була використана фреймворк Laravel на основі PHP, JavaScript бібліотека Vue.js та база даних PostgreSQL.

В результаті було розроблено MVP система управління транспорту у вигляді SaaS для логістичних компаній, який має зручний та інтуїтивний інтерфейс, підтримує функціонал, який автоматизує та покращує ефективність роботу, такі як: створення контрактів, аналітики, користувачів, відстеження процесу перевезення.

02.06.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101073.01.05.ПЗ	Пояснювальна записка	76		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ.2101074.01.05.Е8	DFD-діаграма	1		
5	А4	КвРІПЗ.2101074.01.05.Е8	ER-діаграма	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2101073.01.05.ПЗ		
Виконав		Гуменюк І.А..		2105.	Вебсистема для автоматизації логістичних операцій	Літ.	Арк.	Аркушів
Керівник		Праворська Н.І.		2106				
Рецензент		Кашубін І.В.		2106				
Н. Контр.		Бедратюк Г.І		2106		ХНУ, ІПЗ-21-1		
Зав. Каф.		Бедратюк Л.П.		2106	Пояснювальна записка			

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ШІ — Штучний інтелект

SaaS — Software as a Service (Програмне забезпечення як послуга)

ПЗ — Програмне забезпечення

MVP — Minimum Valuable Product (Мінімально життєздатний продукт)

TMS — Transport Management System (Система управління транспортом)

ER — Entity-relationship (Модель «сутність-зв’язок»)

UML — Unified Modeling Language (Уніфікована мова моделювання)

DFD — Data Flow Diagram (Діаграма потоків даних)

CRM — Customer relationship management (Управління відносинами з клієнтами)

FK — Foreign Key

БД — База даних

СКБД — Система керування базою даних

ORM — Object-relational mapper

CRUD — Create Read Update Del

ВСТУП

Одна з основних переваг технологічного розвитку, в межах розумного — автоматизація рутинних задач та прискорення роботи. Людям все менше потрібно піклуватися про по декуди нудну і рутинну роботу. Для прикладу, бухгалтерам не потрібно створювати квартальні звіти самостійно від руки. Достатньо зайти у кабінет платника податків, згенерувати потрібний звіт і ввести дані, якщо у цьому є потреба, оскільки дані підтягуються з бази даних. Все швидко і доступно. І немає стосів непотрібних паперів, які займають дуже багато місця. І ця процедура займає значно менше часу. Згідно даних, 31% компаній повністю автоматизували принаймні одну функцію. 76% використовують автоматизацію для стандартизації щоденних робочих процесів, 58% - для звітності, 36% - для регулювання та дотримання вимог законодавства [1].

З вище наведених даних, можна дійти висновку, що багато бізнес процесів не автоматизовано. Але з розвитком штучного інтелекту та машинного навчання автоматизація процесів ще більше пришвидшилося. Понад 82% компаній використовують або вивчають можливості застосування ШІ. У світі налічується 333,34 мільйона компаній, а це означає, що понад 266 мільйонів компаній використовують або досліджують ШІ у своїх бізнес-операціях [2].

Однією з сфер, яка має можливості автоматизації, є сфера логістики. Станом на 2024 рік приблизно 25% складів у всьому світі впровадили ту чи іншу форму автоматизації, і лише 10% використовують передові технології автоматизації [3]. Автоматизувавши облік документів, відслідковування завантажень та перевезень і інших бізнес процесів — значно підвищить ефективність роботи компанії.

Дана проблема є актуальною, оскільки на ринку не багато рішень, і то вони не ефективні, або коштують дорого у встановлені та налаштуванні. Не має рішення, яке дозволяло би компанія самостійно налаштовувати систему для роботи, у вигляді SaaS. Дана проблема є досить актуальною, оскільки не всі

компанії можуть дозволити собі встановлення ПЗ на початкових етапах існування.

Отже метою даної кваліфікаційної роботи є розробка MVP системи керування транспортом (або скорочено TMS) для логістичної компанія, яка дозволятиме логістам, керівникам та іншим співробітникам компанії відстежувати перебіг контрактів та перевезень, аналітику, єдину базу водії та клієнтів.

Щоб досягнути поставленої мети, потрібно:

– дослідити детально предметну області кваліфікаційної роботи, визначити особливості даної області;

– проаналізувати існуючі рішення, визначити основні переваги та недоліки цих рішень;

– визначити функціональні та нефункціональні вимоги, які мають бути виконанні для успішного досягнення поставленої мети;

– детально спроектувати базу даних, за допомогою ER-діаграми, та систему, використовуючи UML діаграми;

– обрати потрібний стек технологій, враховуючи особливості області;

– успішно програмно реалізувати систему за використанням вибраного стеку технологій та урахувавши поставлені вимоги;

– виконати тестування системи на правильність виконання та відповідність функціональних вимог.

Даний проект має потенціал перетворитися на працюючий SaaS продукт, який зможе задовільнити потреби логістичних компаній та полегшити життя у керуванні компаній. Це може бути можливо завдяки автоматизації рутинних час, які зазвичай забирають багато часу.

У майбутньому, продукт може включати багато різних корисних функцій, як генерування документів, створення додатку для водії та трекінг транспорту, або впровадження штучного інтелекту, який буде аналізувати та рекомендувати оптимальні рішення для складних задач.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Логістика є однією з основних сфер, що забезпечує безперебійне функціонування глобального бізнесу. Вона включає широкий спектр процесів, пов’язаних із транспортуванням товарів і наданням послуг від виробника до кінцевого споживача. Останні десятиліття її роль значно зросла завдяки розвитку електронної комерції, змінам у ланцюжках постачання та прагненню компаній до підвищення ефективності.

Логістика – це процес планування, організації та управління переміщенням товарів, послуг або інформації від місця їх виробництва до кінцевого споживача. Вона включає в себе все, що потрібно для того, щоб товар або ресурс опинився в потрібний час, у потрібному місці та в потрібній кількості з мінімальними витратами [4].

Основні функції, які виконує логістика:

– організація руху товарів між різними пунктами;

– забезпечення належних умов для продукції до її відправлення споживачам;

– регулювання кількості товарів для покриття попиту без зайвих витрат;

– управління процесом виконання замовлень, включно з поверненнями;

– використання даних для аналізу та підвищення ефективності логістики.

Ефективні логістичні операції є основою будь-якої організації і гарантують, що бізнес може отримувати, виробляти та доставляти свою продукцію або послуги кінцевому споживачеві вчасно та економічно ефективно.

Логістична компанія, також відома як постачальник логістичних послуг, надає широкий спектр послуг, пов'язаних з управлінням постачанням, складським господарством, упаковкою та доставкою товарів. Вона є ключовою

ланкою в ланцюжку постачання, забезпечуючи ефективність і плавність всіх етапів. Логістична компанія займається прогнозуванням та плануванням потреб у товарах, управлінням запасами, організацією транспортування, а також контролем та відстеженням поставок [5].

Логістичні компанії значно полегшують життя бізнесу, а саме:

– логістична компанія бере на себе всі організаційні та операційні завдання пов'язані з поставками, що дозволяє бізнесу зосередитися на своїй основній діяльності;

– завдяки досвіду та професіоналізму логістичної компанії можна досягти оптимальної вартості доставки та зберігання товарів, а також покращити управління запасами;

– логістична компанія надає високий рівень сервісу, відстежуючи поставки, обробляючи повернення та вирішуючи будь-які можливі проблеми;

– за допомогою логістичної компанії бізнес може доставляти свої товари по всьому світу, розширюючи свою клієнтську базу та збільшуючи обсяги продажу.

Дані компанії надають низку послуг. Основі послуги кожної компанії [5]:

– транспортні послуги — організація та координація доставки товарів з використанням різних видів транспорту;

– послуги складу для зберігання та управління запасу;

– пакування та маркування товарів перед відправкою, підготовка товарів до відправки;

– контроль перевезень товарів на всіх етапах поставки, включаючи відстеження вантажу, моніторинг термінів та вирішення різноманітних проблем, які можуть виникати під час перевезення.

Як можна зрозуміти з вище наведених функцій та послуг, які надають логістичні компанії, логістика займає важливу роль у функціонуванні багатьох бізнесів. Якщо автоматизувати просто декілька рутинних бізнес-процесів, то можна досягнути значне підвищення ефективності роботи логістів. Серед функцій, які можуть підпадати під автоматизацію, можна виділити:

– відслідковування перебігу перевезень;

– обробку заявок на перевезень, і у майбутньому можна інтегрувати ШІ, які буде кваліфікувати потенційних клієнтів;

– автоматична генерація потрібних документів;

– управління водіями за допомогою єдиної бази водіїв;

– створення аналітики та звітів;

– у майбутньому, є потенціал створення додатку для водіїв, щоб забезпечити безперервне оновлення даних про перевезення.

Створивши вебсистему у вигляді SaaS платформи, можна допомогти логістичним компаніям з легкістю впроваджувати такі систему у роботу, без потреби звертатися до розробників. Дане рішення буде доступне навіть для компанії, які тільки на ранніх етапах розвитку.

1.2. Аналіз наявного програмно-технічного забезпечення предметної області

Щоб правильно зрозуміти предметну область, які функціоналі вимоги є обов'язковими, потрібно зробити аналіз наявного програмно-технічного забезпечення предметної області.

В рамках даної кваліфікаційної роботи буде проаналізовані наступні рішення: LBS Cloud, Qguar TMS, Tocan. Нижче розглянуто загальні функції, недоліки та переваги кожної системи окремо.

LBS Cloud пропонує готові CRM-рішення для розвитку різних типів та розміру бізнесу. LBS Cloud має різний спектр програм для підвищення ефективності бізнесу, прискорення процесів і зниження витрат. Додатки вже інтегровані один з одним, тому додавання нових модулів до CRM, таких як бухгалтерія, завдання або контакт-центр, відбувається швидко і без залучення програмістів [6].

Для логістичних компаній LBS Cloud пропонує наступні функцій:

– зберігання всю інформацію про клієнтів, їх замовлення та проблеми в одній базі даних;

– планування маршрутів, вибору транспортних засобів та розкладу доставки, що дозволяє зменшити витрати на транспортування та підвищити ефективність доставки;

– обговорення робочих моментів в єдиному централізованому середовищі;

– контроль фінансових надходження, здійснення оплати рахунків, введення, бухгалтерський облік та аналіз фінансових даних;

– створення проектів та розподіл завдань між працівниками;

– розширення функціоналу за допомогою модулів.

На рисунку 1.1 зображено приклад інтерфейс системи.

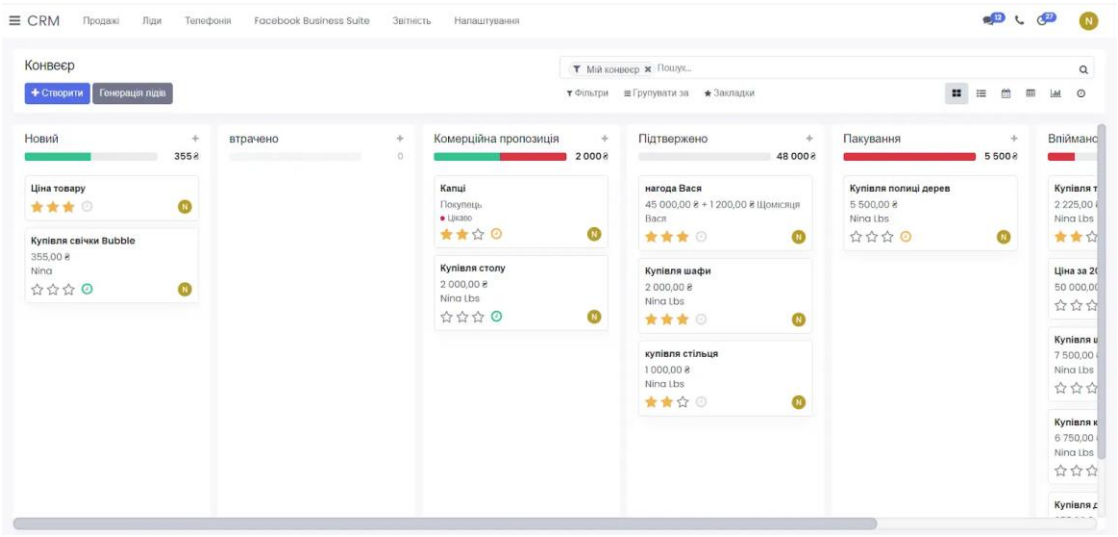


Рисунок 1.1 — Інтерфейс системи LBS Cloud

Перевагою даного рішення є те, що вона розроблена у вигляді SaaS. Тобто користувачі можуть самостійно, без втручання розробників, налаштувати систему до роботи. Іншою важливою перевагою даної системи є інтеграція з багатьма сторонніми сервісами.

Основною проблемою даної системи є те, що вона не націлена на логістику. Дана система може не закривати всі потреби логістичної компанії, такі як відслідковування перевезення, складу і інших функцій.

Qguar TMS — це автоматизований програмний комплекс, який використовується в центральному офісі компаній, а також у віддалених підрозділах підприємств [7]. Qguar TMS є інструментом, призначеним для широкого кола підприємств, які здійснюють перевезення різними видами транспортних засобів. Використання системи ефективно як для компаній, що мають власний автопарк, так і при використанні орендованого транспорту.

Серед базового функціоналу Qguar TMS, можна виділити:

– управління заявками на перевезення;

– створення та планування транспортних маршрутів;

– обслуговування додаткових експедиторських послуг;

– можливість роботи з мобільними пристроями різного типу;

– розрахунок транспортних послуг;

– розрахунки тимчасових вікон доставки;

– обслуговування ADR;

– обслуговування договорів, що стосуються транспортних послуг;

– обумовлені й настроюються користувачами алгоритми розрахунку вартості перевезень.

На рисунку 1.2 зображено інтерфейс системи Qguar TMS

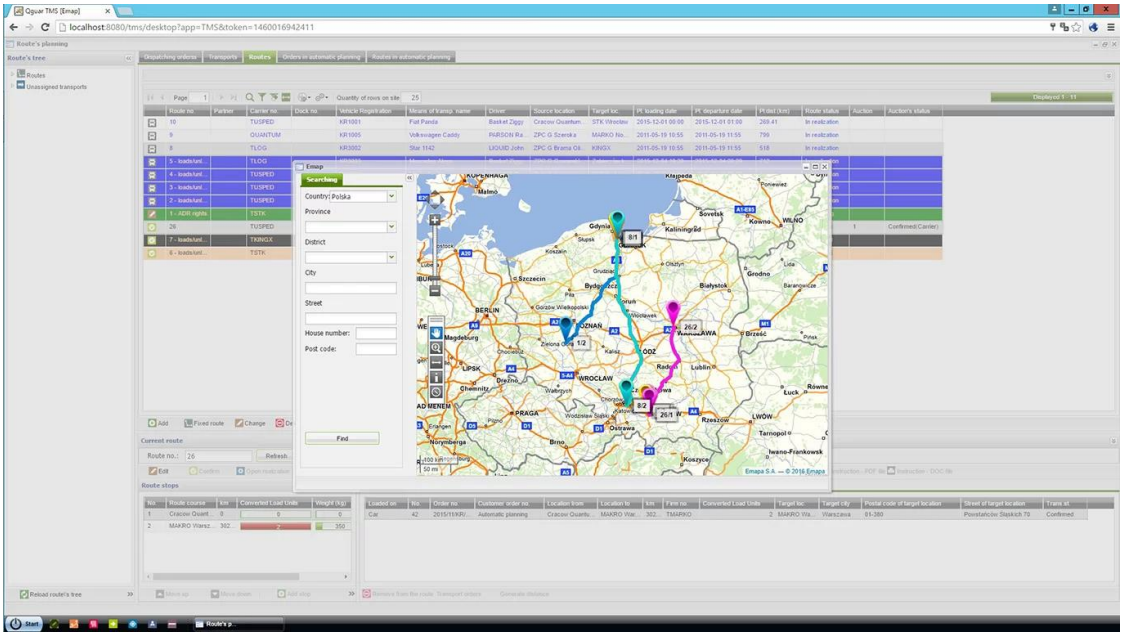


Рисунок 1.2 — Інтерфейс Qguar TMS

Основною перевагою даної системи є високий рівень довіри та велика кількість позитивних відгуків. Дана система має високий рівень довіри через роботу з великими брендами, як Яготинське, Royal Cannin, Епіцентр.

Недоліком є складність впровадження. Дана система не зроблена у вигляді SaaS. Тобто компаніям потрібно додатково розробників, щоб вони встановили та налаштували систему на сервері. Це може коштувати дорого і не всі компанії можуть дозволити встановлення такого ПЗ.

Останнім готовим рішенням є Tocan. Tocan TMS - це сучасне програмно-апаратне рішення для автоматизації та керування процесами у транспортній логістиці. Підходить для компаній, що надають транспортно-експедиційні послуги та для компаній, що мають власний парк машин. Також Tocan підходить як для магістральних перевезень, так і для внутрішньо-міської адресної доставки [8].

Серед базового функціоналу можна виділити наступне:

– планування маршрутів виходячи з більш ніж 100 параметрів, що використовуються;

– відстеження пересування машин та співробітників на карті в режимі онлайн;

– мобільний додаток для водія, експедитора, кур'єра, торгового представника, мерчандайзера, сервісного інженера, служби охорони та ін.;

– модуль картографії з будь-якими картографічними сервісами;

– прогнозування завантаженості транспорту;

– ефективне управління працівниками, вантажем, подвижной состав;

– можливість справлятися із завданням меншою кількістю машин;

– зменшення впливу людського фактору;

– швидке прокладання оптимальних маршрутів;

– контроль за паливом та рухом;

– синхронізація з будь-якою обліковою системою;

– можливість встановлення будь-яких датчиків;

– легка адаптація під індивідуальні запити.

На рисунку 1.3 зображено приклад інтерфейсу системи Tocan TMS.

Рисунок 1.3 — Інтерфейс Tocan TMS

Як зазначає сам розробник системи, основною перевагою є ціна. Розробник пропонує два варіанта використання: у вигляді SaaS або повне придбання готового рішення. Тому дане рішення може підійти, як і маленьким компаніям, так і великим компаніям. Також система має інтеграції з іншими системами. Інша велика перевага — це ціла система модулів, які зв’язані між собою. Це дає повний контроль над всіма процесами.

Недоліком є застарілий та доволі не зручний дизайн. При перегляді інтерфейс виникає «страх», що все дуже важко.

У даному розділі, було проведено аналіз наявного програмно-технічного забезпечення предметної області на ринку, а саме: LBS Cloud, Qguar TMS та Tocan TMS. Визначено основні функції, які надають дані рішення. Дана інформація є надзвичайно цінною, оскільки дала загальне розуміння, які функцій повинні бути у розроблені MVP у рамках даної роботи. На основі аналізу наявних рішень можна виділити наступний функціонал необхідний для MVP системи, яка розробляється у межах даної кваліфікаційної роботи:

– прийом та обробка заявок, планування маршрутів, вибір транспортних засобів;

– онлайн-відстеження машин і вантажів;

– розрахунок вартості перевезень, управління договорами, фінансовий контроль;

– розподіл завдань між співробітниками, централізоване обговорення робочих процесів;

– модульна структура для розширення, інтеграція з ERP та картографічними сервісами.

Окремо недоліком даних систем є не привабливий та не інтуїтивний дизайн. Тому окремою нефункціональною вимогою є зручний та привабливий дизайн системи. Щоб користувачі могли інтуїтивно та без перешкод працювати з системою.

Завдяки інформації отриманої під час аналіз наявних рішень, у наступному розділі визначено вимоги до розробленої системи у рамках даної кваліфікаційної роботи та поставити задачі для виконання.

1.3. Визначення вимог до вебсистеми та постановка задачі

Завдяки даними отриманих у результаті аналізу предметної області та існуючих рішень, потрібно визначити чіткі функціональні та нефункціональні вимоги до даного проекту. Для кращого розуміння та наочності використано UML діаграму варіантів використання.

Першочергово визначимо роль, які будуть присутнім у даному MVP кваліфікаційної роботи:

– директор компанії;

– керівники відділів компанії;

– логісти.

У рамках даного проекту, не буде розглянуто роль водія і не буде розроблено окрему систему для відслідковування машин під час перевезення. Даний функціонал можна розробити та провадити у майбутньому.

Функції, до яких має доступ мати логісти компанії, відносяться до операційного керування перевезеннями. Серед функцій має бути впроваджено:

– управління заявками на перевезення, а саме, додавання, прийом, та підтвердження перевезення заявок, призначення транспорту та водіїв;

– управління базою водіїв;

– відстеження перебігу перевезення, взаємодія з водіями;

– контролювання контрактів;

– перегляд історії маршрутів та їх ефективності.

На рисунку 1.4 зображено діаграму варіантів використання для ролі логіста.

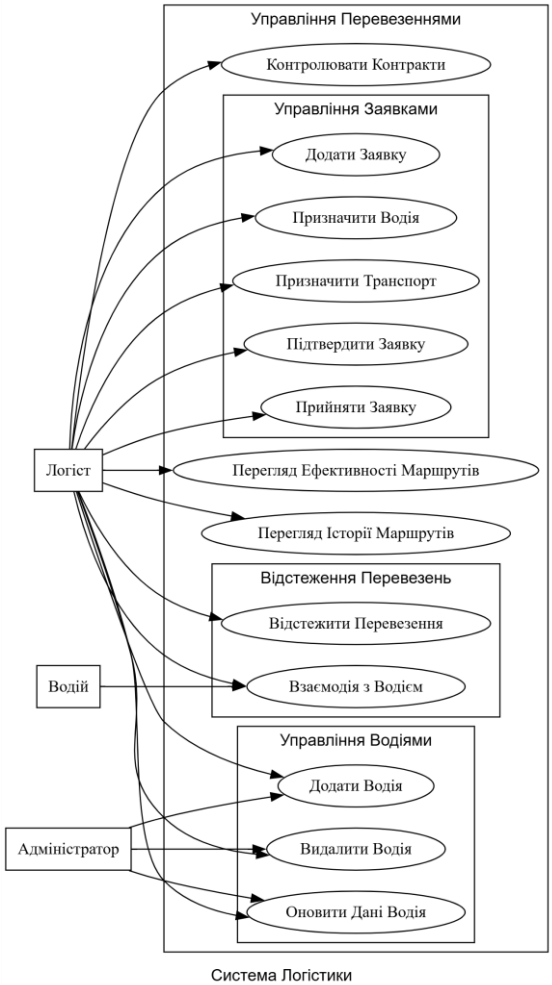


Рисунок 1.4 — Діаграма варіантів використання для ролі логіста

Керівник відділу — це член команди, який контролює виконання обов’язків логістами, тобто він керує персоналом. Серед функцій, які має мати керівник відділу у системі, має бути впроваджено:

– перегляд аналітики, яка відображає дані ефективності та огляд загальної статистики перевезень;

– призначення завдань для логістів, перевірка їх виконання;

– керування базою товарів;

– керування контрактами, можливість призначення логістів до виконання контракту.

На рисунку 1.5 зображено діаграму варіантів використання для ролі керівника відділу.

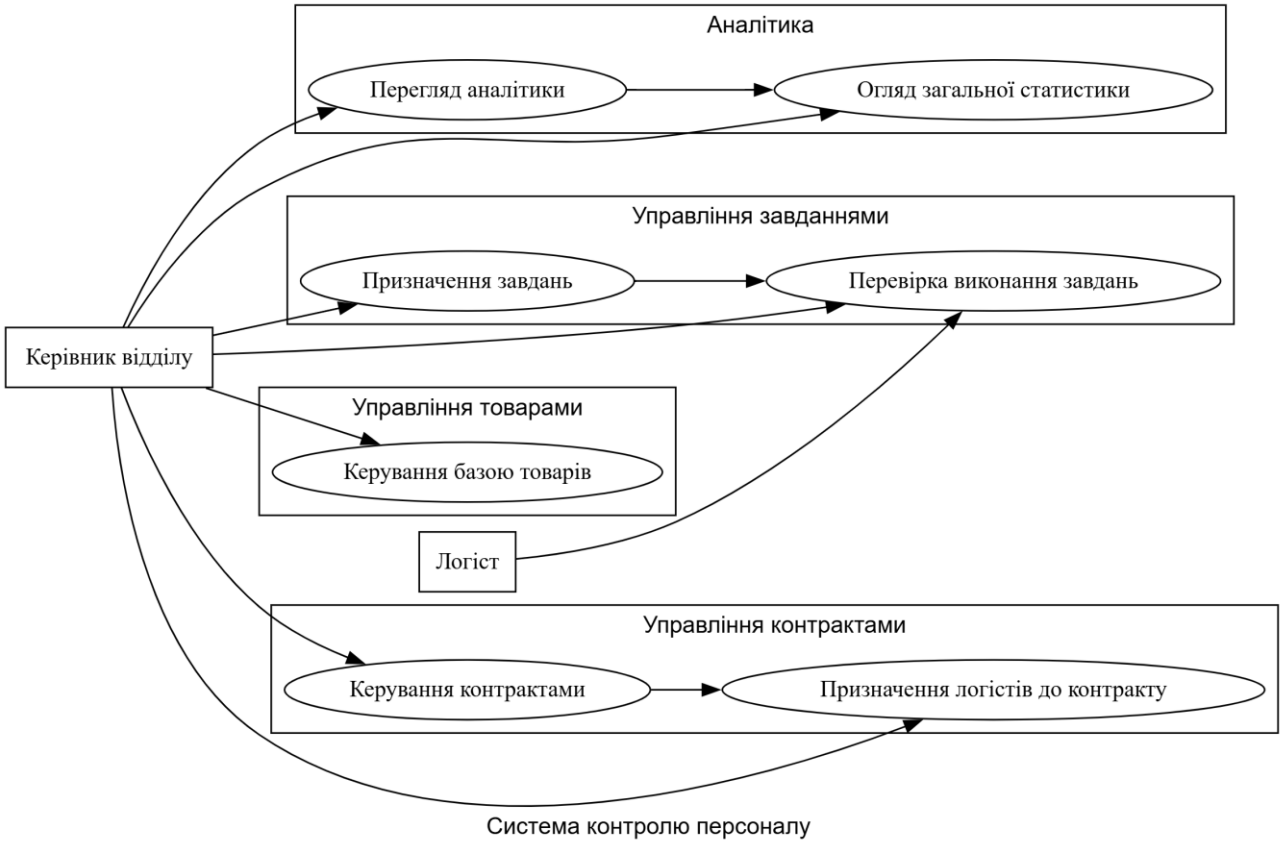


Рисунок 1.5 — Діаграма варіантів використання для ролі керівника відділу

Основна функція директора компаній — робити все, що впливає на успіх компанії на ринку. Він не лише займається поточним управлінням, але й формує

стратегічний напрямок, приймає важливі фінансові та кадрові рішення. Серед функцій, які має мати директор компанії, має бути впроваджено:

– управління користувачами, а саме додавання, редагування та видалення співробітників (логістів, керівників), та керування рівнями доступу до системи;

– перегляд аналітики, яка відображає фінансові дані, дані ефективності компанії та команди, контроль заборгованостей та платежів, огляд загальної статистики перевезень;

– налаштування параметрів роботи системи (валюта, типи перевезень тощо);

– керування роботою команди.

На рисунку 1.6 зображено діаграму варіантів використання для ролі директора компанії.

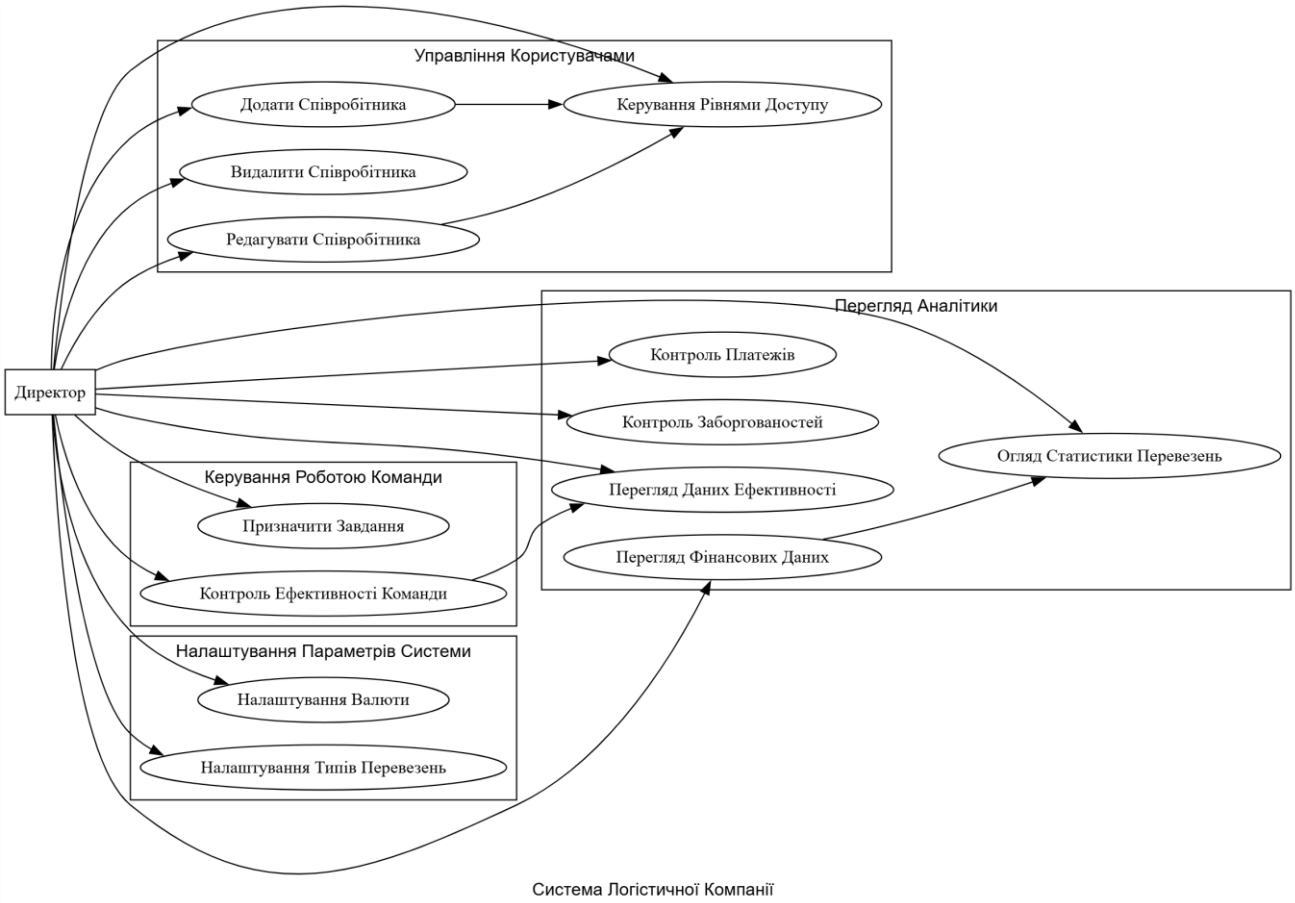


Рисунок 1.6 — Діаграма варіантів використання для директора компанії

Серед нефункціональних вимог виділено наступні вимоги:

– система має бути швидко загружатися, а саме не більше 3 секунд;

– обробка одночасних запитів без втрати продуктивності;

– система має підтримувати збільшення кількості користувачів та перевезень без впливу на продуктивність;

– однакова робота на всіх девайсах та на всіх операційних системах;

– зручний, привабливий та інтуїтивно зрозумілий інтерфейс.

1.4. Висновки дослідження предметної області

У даному розділі, було чітко визначено основні функціональні та не функціональні вимоги, які мають бути виконані у рамках даної роботи. Було розглянуто три ролі, а саме: директор, керівник відділу та логіст. У даній роботі не береться до уваги водії, через короткі терміни виконання даної роботи.

Першочергово було проаналізовано сферу логістики. Розглянуто основні функції логістики, що дала розуміння які функції потрібно реалізувати в вебсистемі. На основі цієї інформації було вибрано 3 системи, які були мають потрібний функціонал. Дані системи було проаналізовано та виділено переваги та недоліки.

Для кожної ролі користувачів, було прописано окремі функціональні вимоги. Для кращого візуального сприйняття, були розроблені UML діаграми варіантів використання, які демонструють потрібний функціонал кожному виду користувачів. Вся інформація є критично важливою для детального проектування системи, а саме для вибору архітектури, проектування модулів, класів та БД, вибору стеку реалізації.

Завдяки детальному дослідженню предметної області, було поставлено чітко завдання та вимоги до вебсистеми для автоматизації логістичних операцій. Завдання та вимоги до програмного забезпечення було представлено у вигляді детального технічного завдання, яке можна переглянути в додатку А.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз та проєктування архітектури системи

На основі розділу 1, наступним етапом розробки системи є детальне проектування системи, а саме: визначення архітектури програмного забезпечення, детальне проєктування системи на основі UML, проєктування база даних та вибір методів реалізації програмного забезпечення.

Архітектура програмного забезпечення [9] — це розроблена структура програмного забезпечення, яка включає визначення взаємодії компонентів інтерфейсу з внутрішніми процесами програми. Простіше кажучи, це своєрідний підхід, який визначає які функції за що відповідають та як вони взаємодіють між собою.

Існує декілька видів архітектури програмного забезпечення:

– монолітна архітектура — програма складається з єдиного блоку коду, де всі частини взаємодіють прямо одна з одною [10]. Якщо це велика програма, у ній є всі можливості та функції в одному місці. Основними плюсами даної архітектури це простота на початку, чітке розуміння взаємозв’язків між функціями та модулями програми. Очевидним недоліком є слабка масштабованість;

– мікросервісна архітектура — поділяє систему на багато дрібних автономних частин (мікросервісів), де кожен виконує окрему функцію. Це фактично міні-програми [11]. Ці частини взаємодіють між собою через спеціальні протоколи, для прикладу API. Перевагами, які одразу падають у око, є простота в підтримці та реалізація нових функцій, без втручання у інші модулів. У результаті, зменшення кількості можливих багів. На проти вагу, при мікросервісній архітектурі, важче налаштувати взаємодію всіх модулів;

– клієнт-серверна архітектура — тип архітектури, при якому є окремо клієнт, для прикладу додаток на телефоні, і окремо сервер, який опрацьовує запити клієнтів [12]. Чітке розділення на клієнт та сервер, і можливість змінювати

сервер без посереднього впливу на клієнтів. З цього випливає відповідний недолік, якщо сервер перестає працювати, то перестають працювати клієнти, і необхідність постійного з'єднання з сервером.

Більшість веб-застосунків використовують клієнт-серверну архітектуру, через простоту реалізації і доступність більшості клієнтів, що забезпечує доступність для всіх, якщо користувач використовує не застарілі браузери. Саме через ці причини у рамках даної кваліфікаційної роботи буде використовуватися клієнт-серверна архітектура. На рисунку 2.1 зображено принцип роботи клієнт-серверної архітектури.

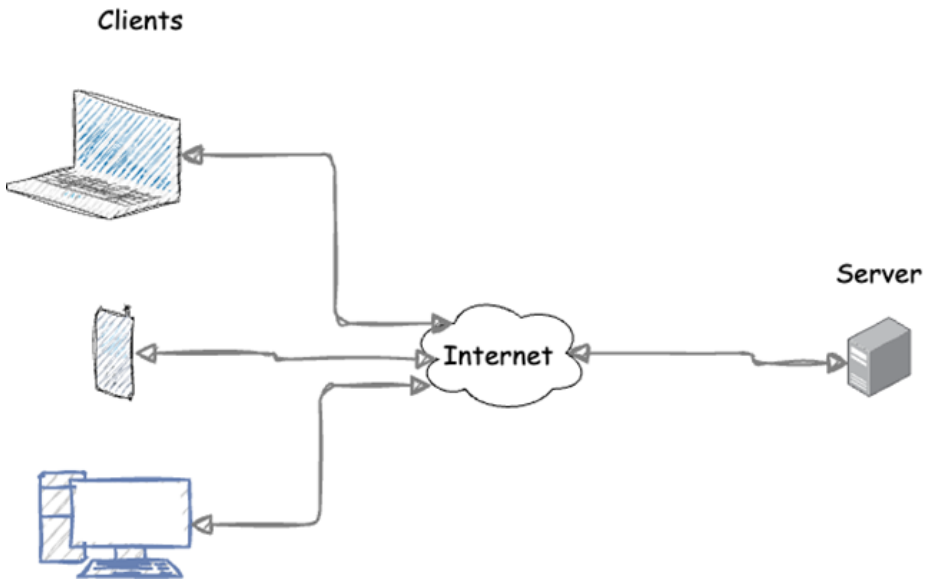


Рисунок 2.1 — Принцип роботи клієнт-серверної архітектури

Окремим пунктом, який обов'язково має бути врахованим є вибір шаблону архітектури програмного забезпечення. Архітектурні шаблони програмного забезпечення (або патерн) — це набір готових «практик», які визначають як буде побудовано програмного забезпечення. У рамках даної кваліфікаційної роботи, буде використовуватися шаблон Model View Controller.

MVC (Model-View-Controller) – це архітектурний шаблон програмного забезпечення, який зазвичай використовується для розробки користувацьких інтерфейсів [13]. Згідно шаблону, архітектура програмного забезпечення має 3 взаємопов'язані сутності:

– модель є представленням даних, бізнес-логіки та правила додатку. У рамках даної вебсистеми моделі є представленнями даних, таких як: контрактів, водіїв, перевезення та інші дані. Всі дані моделей передаються у представлення, де вони відображаються. Додатково, моделі відповідають за оновлення даних отримані з представлень;

– вигляд є безпосередньо інтерфейсом користувача програми. У контексті вебсистеми, всі дані моделей відображаються у представленнях. Варто зазначити, що представлення не містить жодної бізнес-логіки, лише відображення даних;

– контролер є посередником між моделлю та представленням. Саме він обробляє дані, створюються моделі та зберігаються дані, визначають які представлення відображати. Можна сказати, що саме контролери вміщують логіку бізнес-процесів логістичних компаній.

Серед основних переваг MVC є чітко розподілення ролей, що робить сам код легко змінюваним та інтуїтивно зрозумілим. Інші переваги, які варто виділити, це можливість багаторазового використання, легкий супровід та тестування модулів, гарна масштабованість програмного забезпечення.

На противагу перевагам, складність у поділі ролей зазвичай є недоцільним для простих проєктів. Додатково, на початкових етапах можливе використання багато шаблонного коду. Принцип роботи зображено на рисунку 2.2

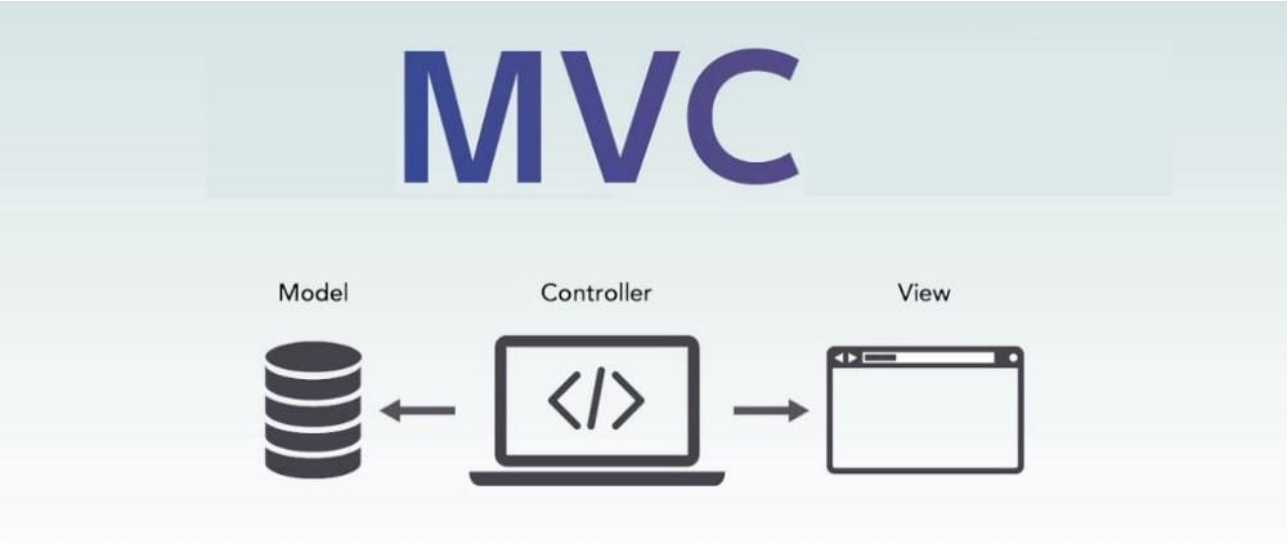


Рисунок 2.2 — Принцип робот паттерну MVC

2.2 Детальне проєктування модулів системи

Визначивши архітектуру системи і архітектурний патерн, наступним важливим етапом проєктування системи є проєктування її модулів. Цей етап включає в себе розробку діаграму, які відображають різні аспекти роботи модулів. Для створення діаграм використовується UML. UML вже використовувалася у першому розділі. Для постановки задачі та розуміння функціональних вимог було використано діаграми використання. Під час проєктування даного MVP у рамках кваліфікаційної роботи, було використано інші UML діаграми, які описано нижче.

Першою розробленою діаграмою була діаграмою діяльності. Діаграма діяльності це в основному блок-схема для представлення переходу від однієї діяльності до іншої [14]. Діяльність можна описати як роботу системи. Основне призначення діаграм активності — відобразити динамічну поведінку системи. Її також називають об’єктно-орієнтованою блок-схемою. Найчастіше такі діаграми використовують, щоб краще зрозуміти бізнес-процеси та моделювання їх.

Для розробки даної діаграми варто описати в загальному процес роботи системи. Основний процес роботи логістичної компанії, яка надає послуги логістики для інших бізнесів, виглядає наступним чином. Першочергово компанія отримує заявки на перевезення. Залежно від товару, компанія може отримувати заявки від фермерів, бізнесів, які мають склади і так далі. Отримавши заявку, компанія опрацьовує її, а саме: перевіряє чи є доступний транспорт та водій, перевіряє ціну перевезення ринку. Отримавши всю необхідну інформацію та оцінивши всі можливі ризик, компанія вирішує чи працювати над цією заявкою, чи відхилити. У разі успіху, компанія створює відповідний контракт з клієнтом та відправляє транспорт. Останнім етапом виконання контракту є відстеження контракту і його перебіг. У разі успішного перебігу контракту, робота вважається завершеною та контракт вважається закритим. На далі дані

використовуються для аналітики. На рисунку 2.3 зображено діаграму діяльності для системи управління перевезеннями, яка візуально описує весь процес.

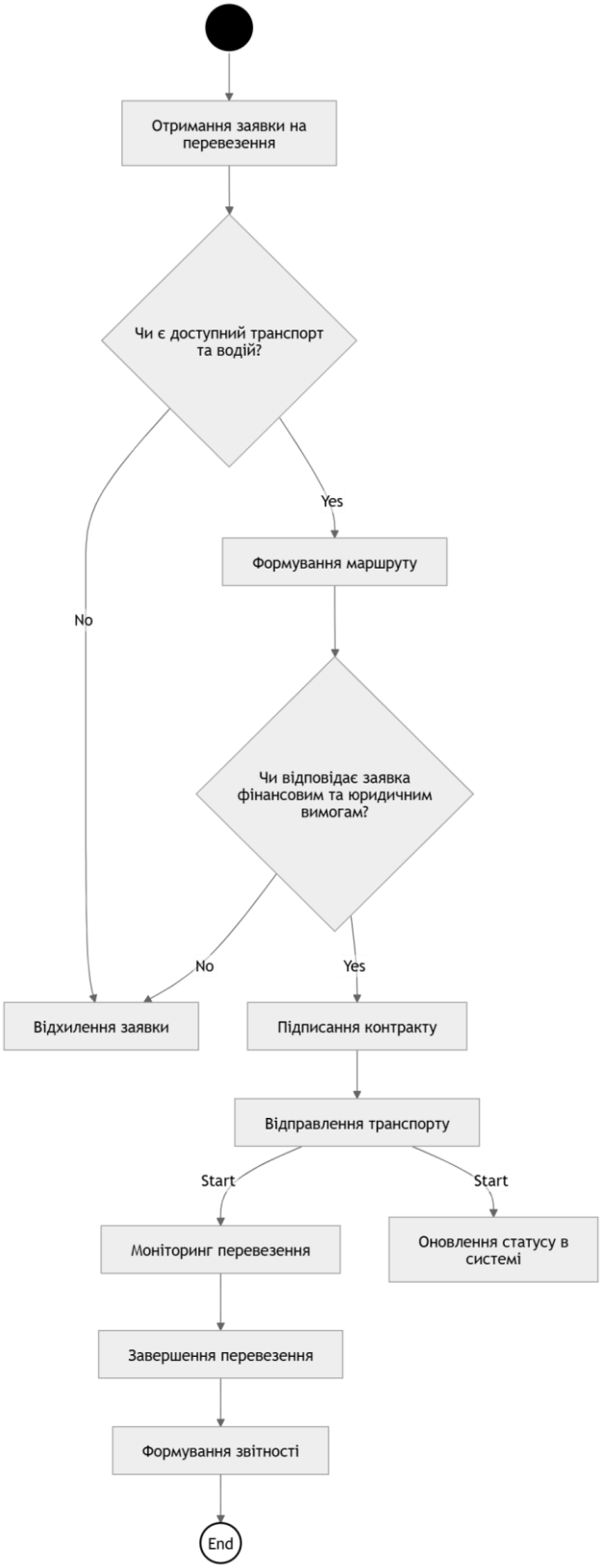


Рисунок 2.3 — Діаграма діяльності для системи

Для кращого розуміння, як відслідковувати перебіг відстеження, потрібно зрозуміти які «стани» проходить будь-яке перевезення. Для моделювання етапів перевезення, було використано UML діаграму станів.

Будь-який об’єкт в систему має скінченну кількість станів протягом свого життєвого циклу, хоча вона може перебувати в більш ніж одному стані одночасно. Кожен стан описується назвою та діями, які можна виконувати, перебуваючи у цьому стані.

Діаграма станів як раз показує ось цей життєвий цикл одного об'єкта, як він змінює свої стани. Діаграма показує перебіг станів, починаючи з моменту, коли об'єкт вперше з'являється на світ, і рухаючись через усі різні стани, в яких може перебувати об'єкт, доки він не буде викинутий і більше не буде використовуватися [15]. Стан на діаграмі станів зображується у вигляді прямокутника із закругленими кутами. Станів може бути будь-яка кількість. Стан може бути розбитий на підстани.

Перехід з одного стану в інший показується односпрямованою стрілкою, що вказує з початкового стану в кінцевий, необов'язково позначеною назвою події, яка спричиняє зміну стану сутності з одного стану в інший, і необов'язково умовами та діями. Початок і кінець життєвого циклу сутності показані спеціальними символами як для початкового стану, який вказує на те, що сутність з'явилася на світ, так і для кінцевого стану, який вказує на те, що сутність ліквідована і життєвий цикл завершено.

Якщо розглядати систему управління перевезеннями, то варто змоделювати стани, через які проходять перевезення. Це відбувається від моменту створення заявки на перевезення. На рисунку 2.4, на якому видно діаграму станів для системи, чітко зображено, всі стани через які проходить заявка до кінця об’єкту. Всі стани зображено у вигляді прямокутників з яких виходять стрілки, які введуть до інших станів. Варто зазначити, що всі стани, у разі виникнення проблем під час перевезення, які не можливо виправити, введуть

до стану Canceled і потім до завершення процесу. Дану діаграму можна використати при розробці статусів перевезень.

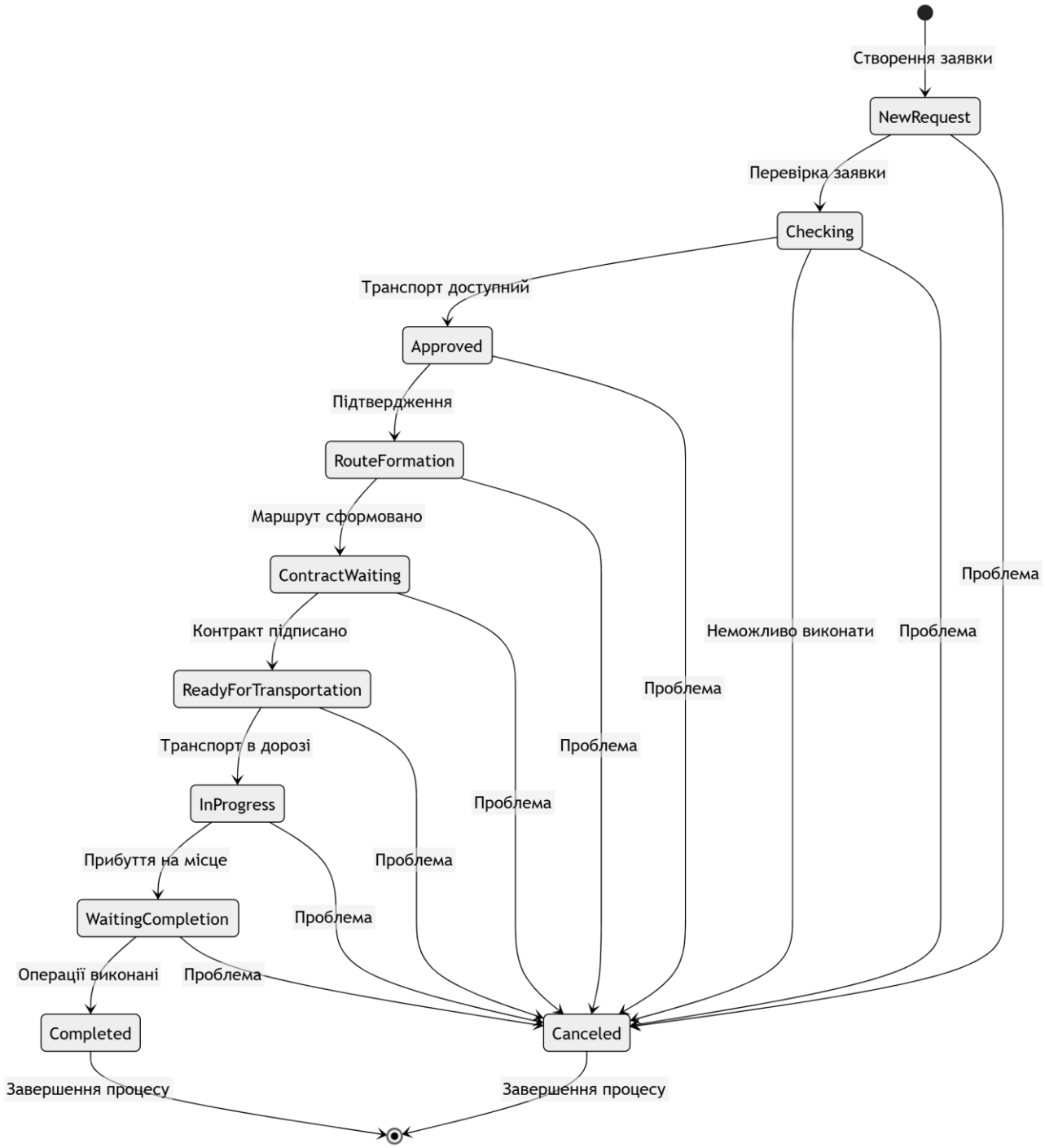


Рисунок 2.4 — Діаграму станів для об'єкту системи управління перевезеннями

На етапі детального проєктування, досить корисно зрозуміти як відбувається опрацювання даних та звідки вони надходять у систему.

Змоделювати потоки даних можна за допомогою діаграми потоків даних (Data Flow Diagram). Діаграма потоків даних або DFD (Data Flow Diagram) – це методологія, яка використовується для графічного структурного аналізу, що описує з якими зовнішніми джерелами та адресатами даних, логічні функції, потоки даних та сховища даних система здійснюється доступ та надає його їм [16]. Варто зазначити, що методологію DFD по праву вважають однією з основних інструментів структурного аналізу та проектування будь-яких систем, що існувала до поширення та застосування UML.

Метою даних діаграм є візуалізація процесу передачі об'єктів, тобто даних між учасниками цього процесу. DFD закриває наступні питання:

– яка структура проєктованої системи?

– як інформація, дані взаємодіють із системою та циркулюють усередині системи?

– що необхідно, щоб потоки даних в проєктованій системі були оброблені?

Згідно методології, діаграми потоків даних поділяються на два типи: логічна та фізична. Логічна DFD зосереджується на бізнесі процесах, тобто зображує потоки даних. У тої же час, фізична DFD показує потоки фізичних сутностей, тобто як переміщуються потоки даних. Якщо називати кожну діаграму одним словом, то логічна це «що», а фізична це «як».

У рамках даної розробки було розроблено ці два види DFD діаграм. На рисунку 2.5 зображено логічну DFD діаграму. З діаграми можна побачити, що у система має 5 глобальних процесів, які працюють з даними, а саме: обробка заявок, створення контрактів та робота з ними, призначення транспорту, підготовка необхідних документів для перевезення та звітів. Дані беруться з 3-ьох сховищ: бази транспортів та водіїв, контрактів та заявок. І з даними взаємодіють 4 сутності: клієнт, водій, державні служби та керівництво компанії. Всі процеси, сутності та сховище з'єднані стрілками, які відображають потік даних між ними.

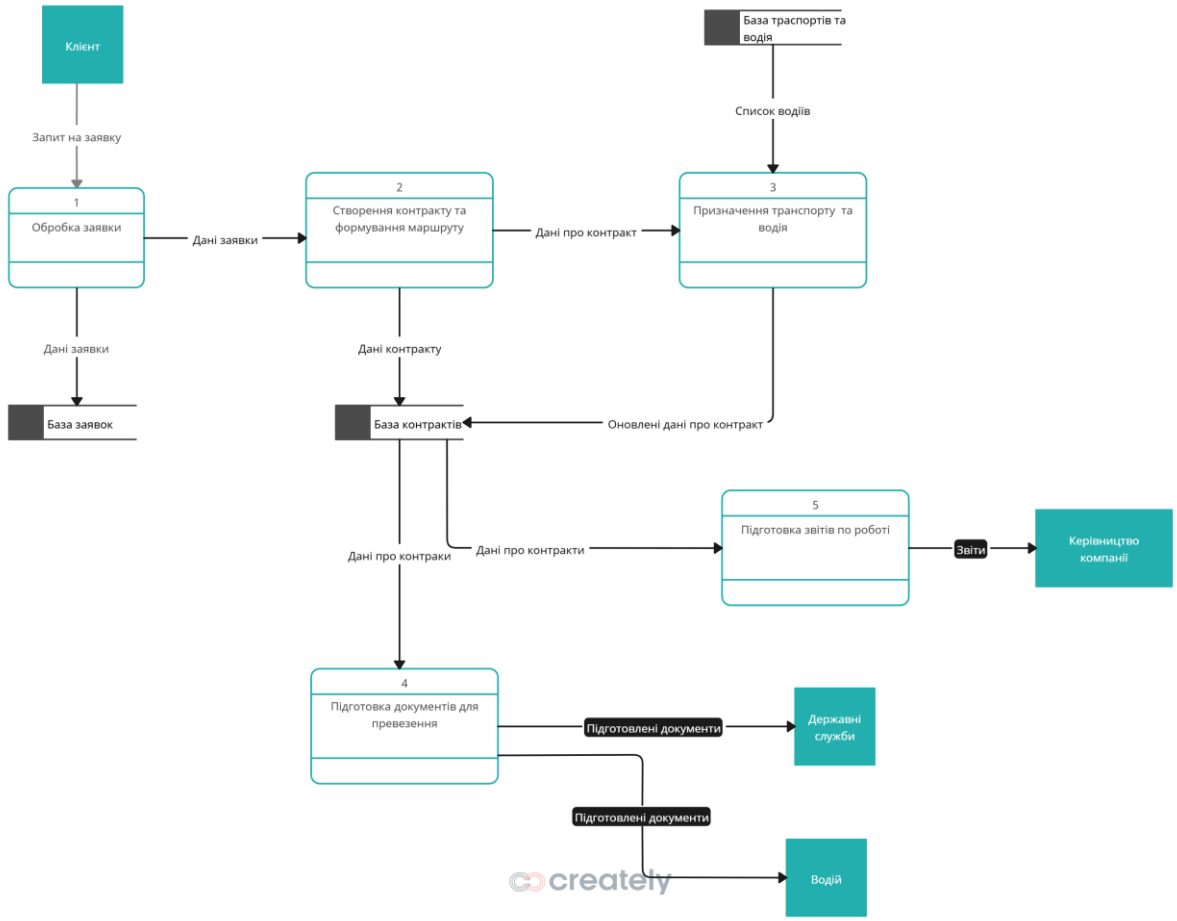


Рисунок 2.6 — Логічна DFD системи управління перевезеннями

Наступною діаграмою було розроблено фізичну DFD. Як було зазначено вище, вона вже стосується безпосередньо самої системи і описує як процеси в ній співпрацюють. На рисунку 2.7 зображено фізичну DFD діаграму для системи управління транспортом у рамках даної кваліфікаційної роботи. У загальному, у рамках MVP даної системи існує 5 процесів, які опрацьовують дані, а саме: авторизація та автентифікація користувачів, відображення та створення контрактів користувачами, оновлення статусів перевезення та генерація звітів. Також існує 4 сховище: бази користувачів, водіїв, контрактів та товарів. І з системою взаємодіють 3 сутності: користувачі (логісти), водії, керівництво компанії. Варто відмітити, що деяких процесів та сутностей не має в фізичній DFD, але були присутніми в логічній. Причина цього, що вони стосувалися саме бізнес процесів, а не системи.

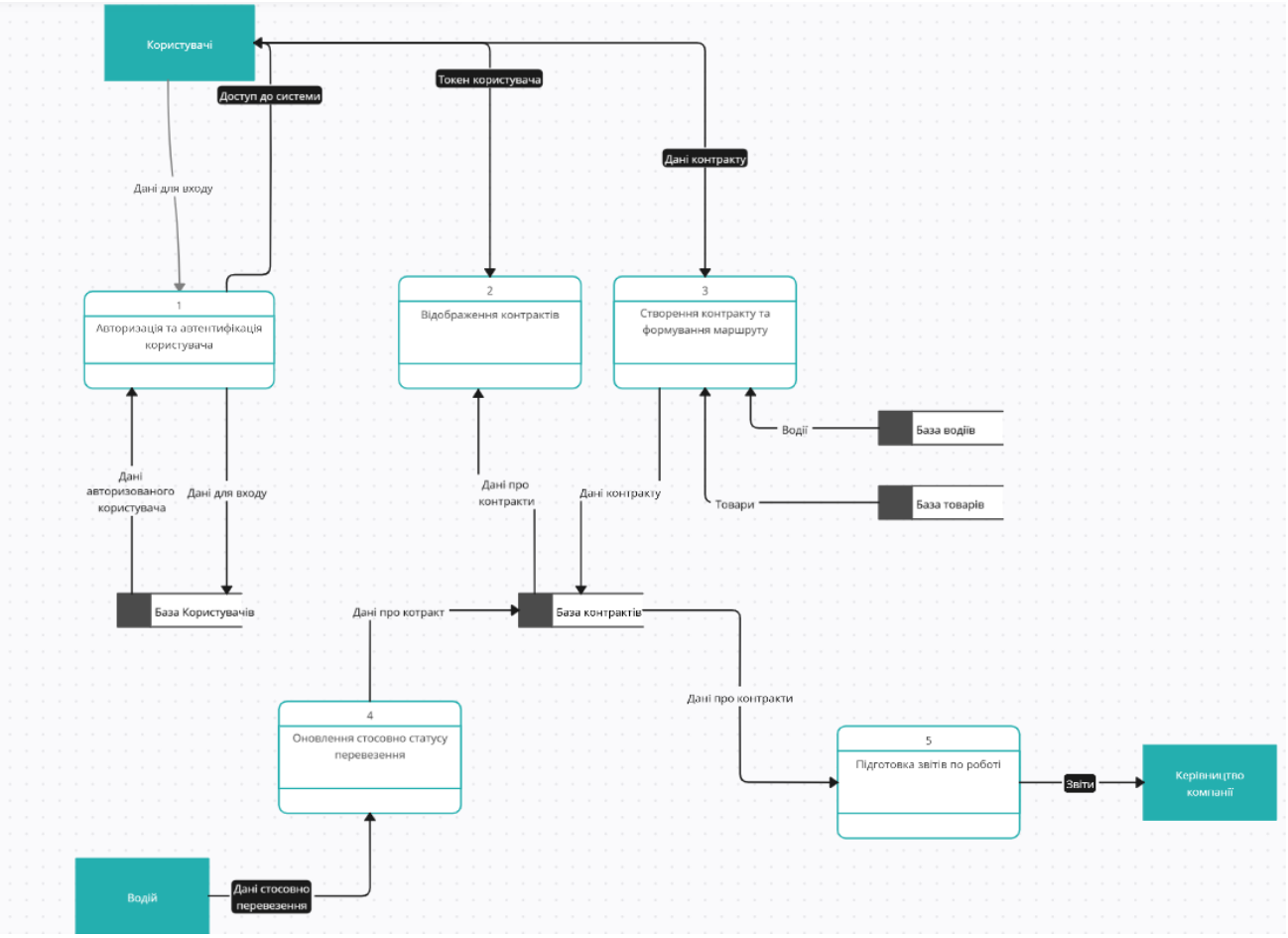


Рисунок 2.7 — Фізична DFD системи управління перевезеннями

Остання діаграма, яка була спроектована на даному етапі є UML діаграма розгортання. Діаграма розгортання — це вид UML діаграм, який визначає фізичне обладнання, на якому працюватиме програмна система. Він також визначає, як система розгортається на базовому обладнанні. Діаграма відображає як прив’язуються частини програмного забезпечення системи до пристрою, який збирається її виконувати [17].

Діаграма розгортання відображає архітектуру програмного забезпечення, створену при проектуванні, на архітектуру фізичної системи, яка її виконує.

Програмні системи проявляються за допомогою різноманітних артефактів (специфікація конкретної сутності реального світу, яка пов’язана з розробкою ПЗ) а потім вони відображаються в вузлах, тобто середовище, яке збирається виконувати програмне забезпечення. У діаграмі розгортання задіяно багато вузлів і зв'язок між ними представлений за допомогою шляхів зв'язку.

На рисунку 2.8 зображено діаграму розгортання для системи управління транспортом. Варто зазначити, що по причині, того, що не було обрано стек, на якому буде розроблятися система, у діаграму було включено можливі вузли, які у фінальні версії, можуть бути іншими.

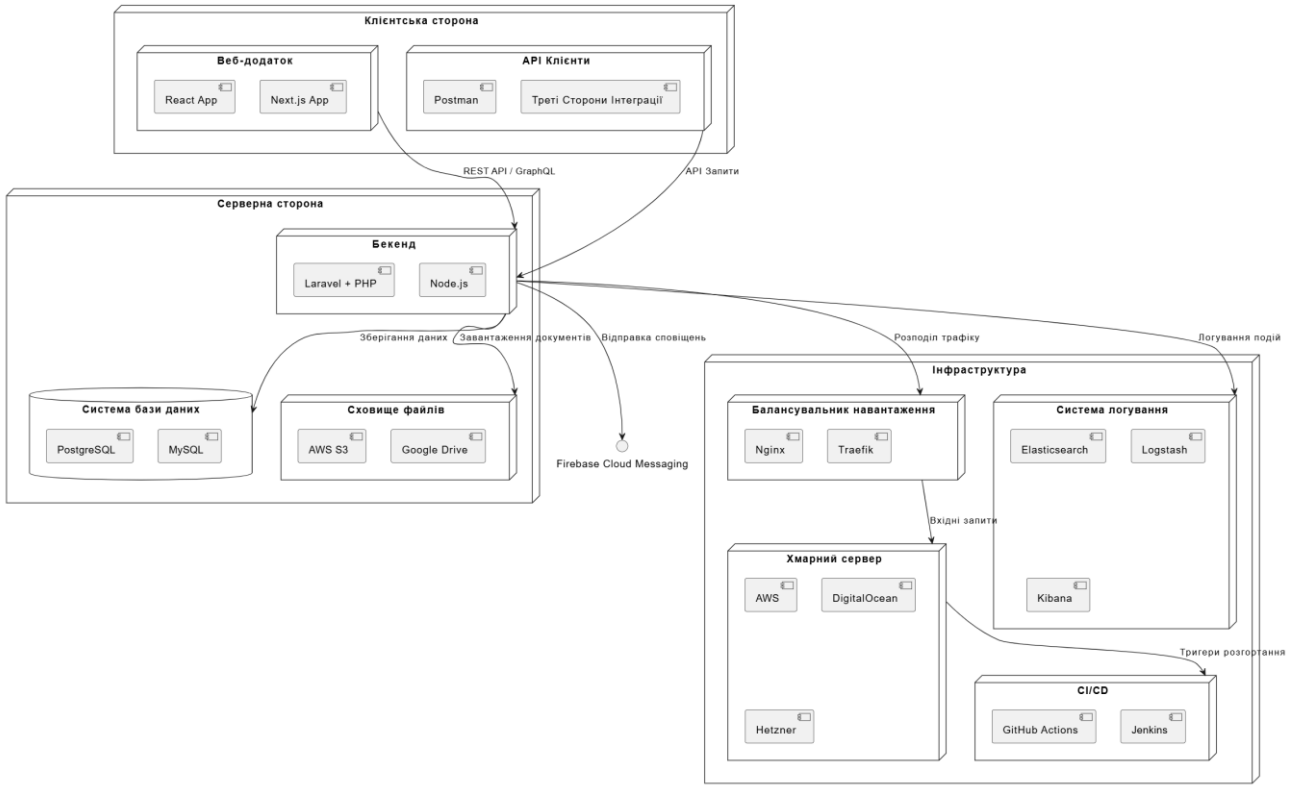


Рисунок 2.8 — Діаграма розгортання системи управління транспортом

Отже у цьому розділі було детально описано етап проєктування MVP системи управління транспортом. Було розглянуто UML діаграми, які були розроблені у рамках даної кваліфікаційної роботи, зображено їх на рисунках та детально описано, що вони відображають. Додатково було розглянуто проєктування DFD діаграм, а саме: логічної та фізичної. Таким чином, було спроєктовано бізнес-процеси логістичної компанії, процеси системи у рамках даної кваліфікаційної роботи та як дані циркулюють у середині компанії та системи. Додатково в кінці було діаграму розгортання, для розуміння того, як система має працювати на працюючих серверах у результаті. Це допоможе у майбутньому правильно розгорнути ПЗ на сервері.

2.3 Аналіз та вибір типу бази даних, проєктування структури

Важливою частиною будь-якої системи є база даних. Саме у базі даних зберігаються всі дані, з якими на далі введеться вся подальша робота. Тому дуже важливо правильно спроєктувати її і вибрати підходящий тип база даних.

Спершу варто проаналізувати типи баз даних, їхні переваги та недоліки. В загальному, бази даних можна поділити на прості, реляційні та нереляційні [18]. Кожна вид свої підвиди, які розглянуто нижче.

Прості бази даних — це бази даних із простою структурою, як зрозуміло з назви. Наприклад, список дозволених IP-адрес для доступу до мережі, налаштування оточення проєкту, список підписників на розсилку компанії тощо. Це все приклади простих баз даних. Також до простих баз даних можна віднести текстові, ієрархічні та мережеві бази даних. Такий тип баз даних не використовується у веб-розробці, оскільки не можливо реалізувати складної структури, відсутність будь-яких зв'язків між взаємопов'язаних даних і найважливіше, це дуже низька масштабованість. При великій кількості даних дуже тяжко працювати з таким типом баз даних. Тому прості бази даних не підходять для даного проекту.

Реляційні бази даних – це тип баз даних є найстарішим. Основна концепція даної база даних полягає в тому, що дані формуються у таблиці з рядків та стовпців. У рядках наводяться відомості про об’єкти (значення властивостей), а стовпці є властивостями об’єктів (поля), тобто в них зберігається дані. Складні відносини об’єктів у реляційних БД моделюються за допомогою зовнішніх ключів — посилань на інші таблиці. Це дає змогу підходити до питання проєктування бази даних із позицій нормалізації — мінімізації надмірності при описі властивостей об’єктів. Запити в реляційних базах даних формують за допомогою структурованої мови SQL. Вона дозволяє робити вибірки, проводити агрегації та угруповання, змінювати та видаляти дані, модифікувати структуру БД, керувати доступом користувачів до тих чи інших операцій тощо.

Такий підхід має переваги:

– мінімізація обсяг бази даних, оскільки не потрібно кожній страві прописувати назву категорії;

– підвищення цілісності системи (для прикладу, додати до позицію до меню ресторану, тобто бази даних, без попереднього додавання категорії страви буде не можливо);

– спрощення масштабування, але є виняток;

– підвищення стійкість до відмов завдяки через оптимальну організацію схеми таблиць запити на вибірку і агрегацію працюватимуть з меншим обсягом даних, а значить швидше.

Основним недоліком є жорстка структура відомостей про об’єкти. Всі об’єкти мають мати чітку структуру. Іншим недоліком, проблема зі швидкістю та все ж проблема з масштабуванням. При великому обсягу даних, виникають проблеми з швидким доступом до даних.

Всі переваги та недоліки реляційних БД засновані на жорсткій структуризації та типізації відомостей про об’єкти. З одного боку, можна оптимізувати зберігання та індексування даних коштом нормалізації або денормалізації. З іншого боку, складно організувати зберігання та обробку погано структурованих (наприклад, об’єкти кешу) або зовсім не структурованих даних (наприклад, дані з кількох джерел). У такому випадку, на допомогу приходять нереляційні бази даних. Вони не мають такої чіткої структури. Їхня структура гнучка, що й пришвидшує роботу з нею. Завдяки цій гнучкості, дані легко масштабуються до більших об’ємів. Саме тому, великі компанії, як Google, Meta та інші використовують саме нереляційні БД.

Але при цих привабливих перевагах, нереляційні БД мають свої недоліки. Не можливість підтримувати складні запити та транзакції так, як реляційні бази та відсутність стандарту запитів може ускладнити використання та інтеграцію. Саме по цих причинах, не всі бізнеси впроваджують їх в роботу.

На основі вище згаданих переваг, найкращим рішенням для системи управління транспортом є реляційна база даних. Серед основних причин:

– потреба в складних зв'язків між даними, чого не може забезпечити не реляційна база даних;

– механізм транзакцій, який є важливим для роботи з важливими бізнес-даними, такими як фінансові звіти, документи, замовлення — це вимагає високий рівень консистентності даних;

– навіть попри те, що кількість даних буде рости і реляційна БД не може забезпечити таку масштабованість, як не реляційна — все рівно реляційна БД може спокійно витримувати зріс навантаження.

Після вибору типу бази даних, потрібно спроєктувати базу даних. Як зазначалося на початку даної кваліфікаційної роботи, передбачається розробка системи управління транспортом у вигляді SaaS. Тобто не можливо зберігати дані всіх користувачів у одній таблиці. Потрібно забезпечити безпеку даних, або ізолювати, та оптимізувати розміри таблиць. У таких випадках використовується архітектура баз даних, яка називається multi-tenancy.

Multi-tenancy — це стратегія побудови бази даних, коли один екземпляр або кілька екземплярів однієї або кількох бази даних працюють в одному спільному середовищі інфраструктури [19]. Для прикладу, декілька користувачів, які ніяк не пов'язані, користуються програмою. Відповідно виникає проблема в тому, що всі користувачі мають доступ до даних інших. Це викликає безпекові проблеми.

При multi-tenancy, кожен користувач має свій окремий екземпляр бази даних, які є ізольовані між собою. Тобто у рамках вебсистеми, кожна логістична компанія буде мати свій екземпляр бази даних. Тоді ніхто не може получити доступ до чужих даних. Як працює multi-tenancy можна побачити на рисунку 2.9 у порівняні звичайним підходом.

Концепція multi-tenancy набула особливого значення при стрімкому розвитку SaaS. Через вище згадані проблеми, у системі управління транспортом, яка розробляється у даній роботі, буде використовуватися цей підхід.

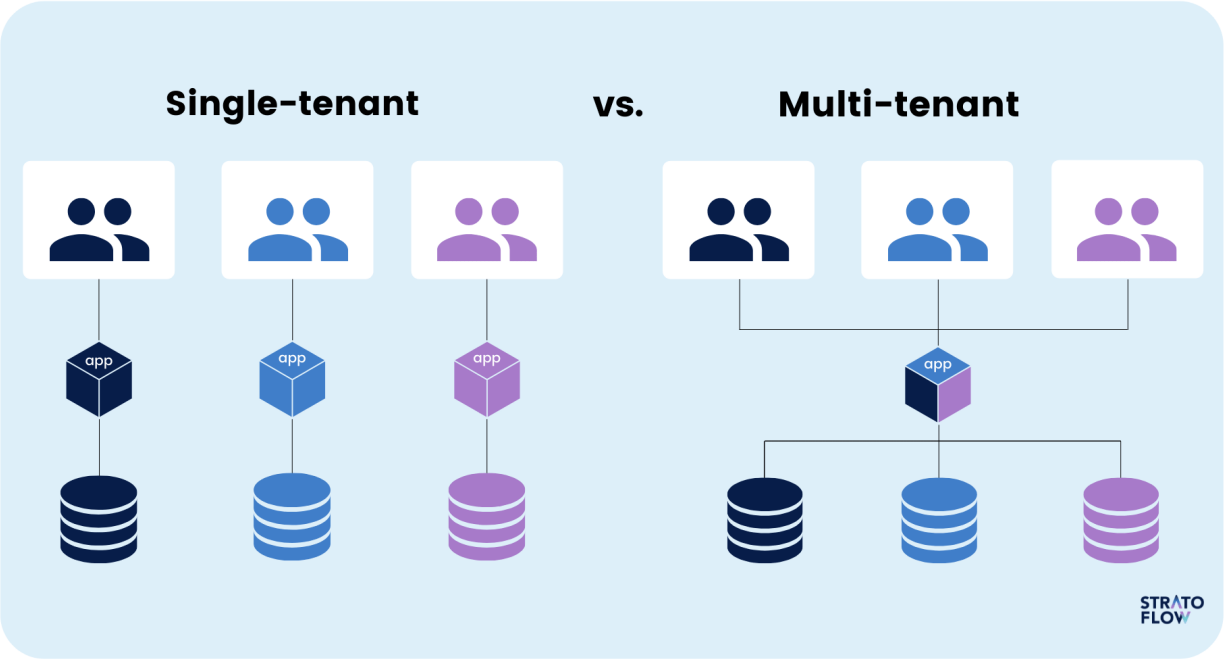


Рисунок 2.9 — Принцип роботи multi-tenancy

Після того, як було визначено архітектуру бази даних, наступним етапом є саме проєктування база даних. Для проєктування бази даних використовується ER-модель. ER-модель (Entity-relationship model або Entity-relationship diagram) – це семантична модель даних, яка призначена для спрощення процесу проектування бази даних. В основі ER-моделі лежать поняття “сутність”, “зв’язок” та “атрибут” [20]. Сутність в базі даних – це будь-який об’єкт в базі даних, який можна виділити виходячи з предметної області для якої розробляється ця база даних.

Перед тим як перейти до моделювання ER-моделі, потрібно визначити сутності, атрибути сутностей та відносини між ними. Нижче описано сутності, які мають бути передбачені при реалізації, їхні атрибути та відносини між ними.

Першою сутністю є Користувачі (Users):

– ID (первинний ключ);

– ім'я;

– email;

– пароль (хешований);

– роль (Директор, Керівник відділу, Логіст);

– дата створення.

Кожен директор можете мати пару Компаній (Companies):

– ID (первинний ключ);

– назва;

– адреса;

– контактні дані;

– ID користувача (директор, FK → Users).

Кожна Компанія може мати безліч Контрактів (Contracts):

– ID (первинний ключ);

– компанія (FK → Companies);

– контрагент;

– дата укладення;

– статус;

– призначений водій (FK → Drivers);

– призначений логіст (FK → Users).

Заявки на перевезення (Transport Requests), які включені в Контракти:

– ID (первинний ключ);

– номер заявки;

– компанія (FK → Companies);

– логіст (FK → Users);

– статус (очікує, підтверджено, в дорозі, завершено);

– дата створення.

Окремою сутністю є Водії (Drivers), які відображають водіїв транспорту:

– ID (первинний ключ);

– ПІБ;

– телефон;

– категорія водійського посвідчення;

– дата найму.

Також окремою сутністю є Транспортні засоби (Vehicles), які належать водіям:

– ID (первинний ключ);

– номерний знак;

– тип транспорту;

– водій (FK → Drivers);

– статус (вільний, у рейсі, на ремонті).

Оскільки клієнти можуть повертатися та маршрути можуть повторюватися, їх буде винесено в окрему сутність Маршрути (Routes):

– ID (первинний ключ);

– заявка на перевезення (FK → Transport Requests);

– пункт відправлення;

– пункт призначення;

– дата виїзду;

– дата прибуття;

– пройдена відстань.

Окремою сутністю є Документи (Documents), які потрібні для перевезень:

– ID (первинний ключ);

– назва;

– шлях до файлу;

– тип документа (контракт, накладна, митний документ);

– пов'язана заявка (FK → Transport Requests);

– дата створення.

Останньою сутністю Товари (Products):

– ID (PK);

– назва;

– опис;

– вага;

– об'єм;

– відправник (FK → Companies);

– отримувач (FK → Companies);

– заявка на перевезення (FK → Transport Requests).

На рисунку 2.10 зображено ER-діаграму, відповідно до сутностей описано вище.

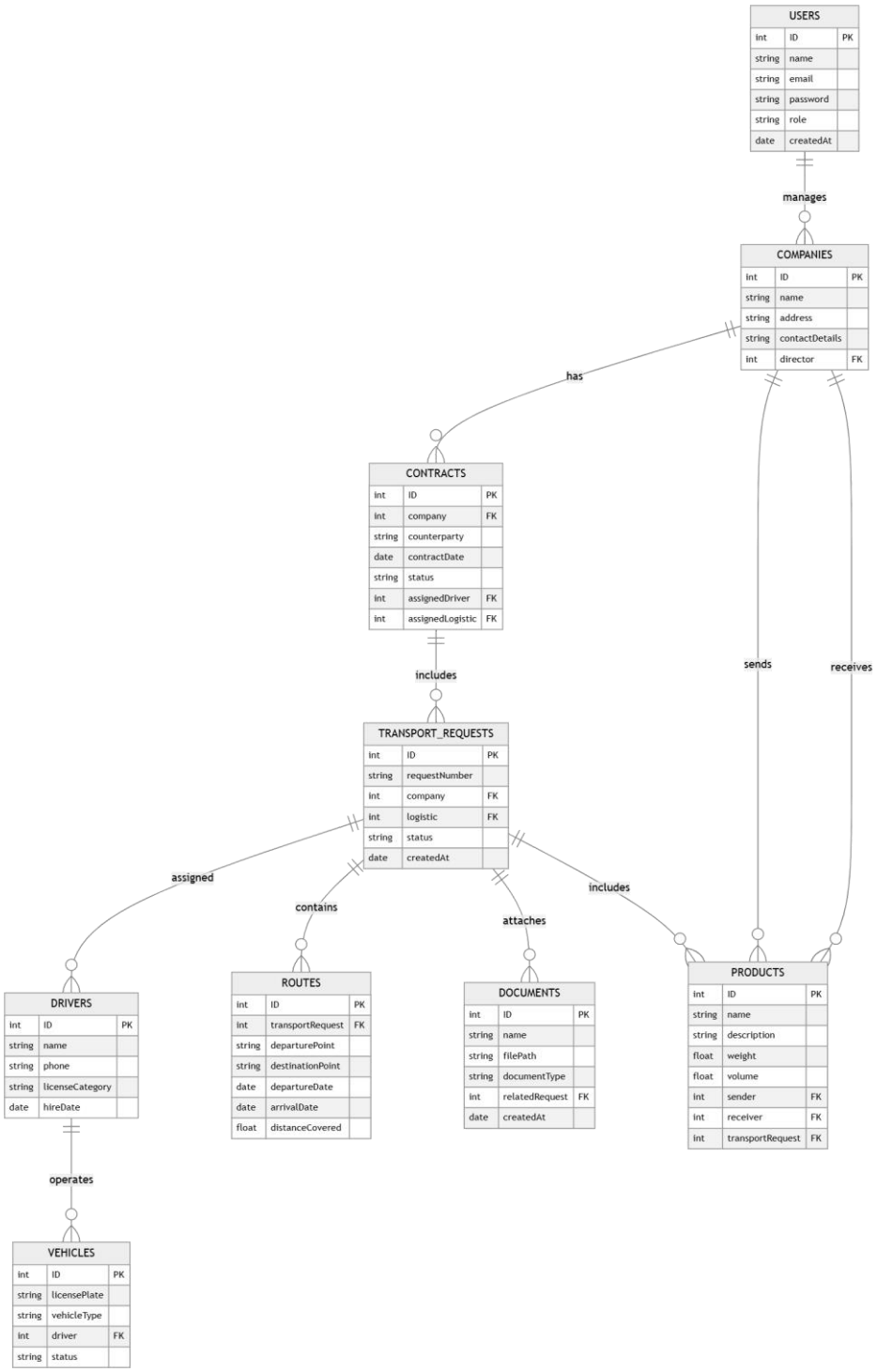


Рисунок 2.10 — ER - діаграма для системи управління транспорт

2.4 Аналіз та вибір технологій і методів реалізації системи

Останнім етапом детального проєктування системи управління транспорту є вибір стеку технологій і методів реалізації системи. Кожна технологія для реалізації має свої переваги, недоліки та сфери роботи.

Спершу варто вибрати технології для реалізації бази даних. Для створення база даних, потрібно використовувати систему управління базами даних. Система керування базами даних (СКБД) — це програмне забезпечення, яке використовується, щоб організовувати, зберігати, маніпулювати та вилучати дані [21]. Серед найбільш відомих реляційних систем керування базами даних можна виділити наступні:

MySQL – це одна з найпопулярніших реляційних баз даних. Дану СКБД часто включають до складу серверів [22]. MySQL працює на всіх основних платформах, а саме: Ubuntu, Debian, Solaris, Microsoft Windows та Apple MacOS. Реалізована підтримка кількох процесорів та багатопоточність. Завдяки API з базами даних можна працювати за допомогою популярних мов програмування. Мінуси в тому, що MySQL не така масштабована, як деякі інші СКБД, та не підходить для роботи з дуже великими даними, має обмежені можливості аналітики.

Досить поширена СКБД PostgreSQL, яка є безкоштовною та об’єктно-реляційною [23]. Незважаючи на те, що СКБД є повністю безкоштовним ПЗ, за її впровадження, модифікацію та підтримку доведеться все же платити додатково. PostgreSQL вирізняється своєю масштабованістю, вона може працювати з базами даних будь-якого розміру та з будь-якою кількістю записів та індексів у таблиці. Особливістю даної СКБД є те, що можна додати власні перетворення типів, домени, індекси, оператори, процедурні мови.

Oracle Database - популярна і повнофункціональна реляційна система управління базами даних, розроблена корпорацією Oracle [24]. Вона відома своєю надійністю, масштабованістю та широкими можливостями. Це робить її

найкращим вибором для підприємств, яким потрібно ефективно керувати та отримувати великі обсяги даних. Oracle Database добре працює незалежно від того, чи використовується вона локально, чи в хмарі, і може впоратися з будь-якими завданнями. Основним недоліком є те, що дана СКБД є платною. Безкоштовна версія Express Edition (XE) обмежена оперативною пам'яттю (1 ГБ), а також лише одним процесором, максимальний об'єм БД для безкоштовної редакції – 11 ГБ. Також варто зазначити, що Oracle Database може виявитися надмірно важкою.

На основі вище описаних перевагах та недоліках для кожного СКБД, для розробки MVP системи буде використовуватися PostgreSQL, через її масштабованість та можливість створювати кастомні поля.

Наступним важливим етапом є вибір мови програмування для реалізації системи. Будь-який веб-застосунок складається з двох частин: front-end (клієнтська частина) та back-end (серверна частина). Спочатку варто вибрати back-end, оскільки система передбачає багато роботи на серверні частині. Нижче розглянуто різні рішення, які можна використати для даного проєкту.

Node.js – це платформа з відкритим вихідним кодом, заснована на мові програмування JavaScript. Вона дає змогу розробникам створювати веб-додатки, що працюють на стороні сервера [25]. Історія Node.js пов'язана з проблемою блокуючого вводу-виводу в традиційних серверних мовах, яка призводила до низької ефективності обробки безлічі одночасних запитів. Node.js появився, щоб розв'язати дану проблему, пропонуючи асинхронну модель роботи, яка дає змогу ефективно обробляти кілька запитів без блокування виконання інших завдань. Також великою перевагою є масштабованість та універсальність. Але попри дані переваги, серед недоліків є досить складна кодова структура та часте оновлення Node.js, що ускладнює підтримку.

Laravel — це безкоштовний PHP-фреймворк з відкритим кодом, який використовується для створення веб-додатків [26]. Він заснований на шаблоні Model-View-Controller (MVC) і пропонує широкий спектр функцій, які

роблять процес розробки більш швидким, простим і зручним. Laravel залишається одним з найпопулярніших PHP-фреймворків завдяки своїй гнучкості, великій спільноті та регулярним оновленням. Laravel має багато переваг, а саме зручна структура проекту, шаблонізатор Blade, масштабованість коду, легка маршрутизація та безпека. Серед недоліків варто виділити швидкість розробки. Якщо команда розробників не має досвіду з Laravel, спеціалістам може знадобитися певний час на вивчення фреймворку.

ASP NET це фреймворк для розробки веб-додатків, створений корпорацією Microsoft [27]. У світі веб-розробки ASP.NET займає важливе місце, забезпечуючи розробникам інструменти для створення масштабованих і сучасних веб-додатків. Розробники можуть зосереджуватися на бізнес-логіці, а не на деталях інфраструктури, завдяки чому процес розробки стає більш ефективним. Одним із недоліків MVC Framework є те, що для цього може знадобитися багато підготовчої роботи, перш ніж програма почне створювати вміст.

Під час розробки даної системи управління транспорту використовувалося PHP фреймовк Laravel. Такий вибір пов’язаний з безпекою, масштабованістю та чітким розподілення коду.

Після того як було вибрано технологію для розробки back-end, залишилося обрати технологію для front-end. Кожен front-end веб-застосунка початково на основі HTML, CSS, JavaScript. Тому знання з цих технологій мають бути по замовчуванню. Але лише на основі цих технологій сучасні веб-застосунки не можливо побудувати. Нижче розглянуто популярні фреймворки для роботи в front-end.

Відомою бібліотекою для роботи з front-end є React.js. Фреймворк React.js — це бібліотека JavaScript з відкритим вихідним кодом, розроблені та підтримувані Facebook і Instagram [28]. Він використовується для швидкого та ефективного створення інтерактивних користувацьких інтерфейсів і вебдодатків із застосуванням значно меншої кількості коду, ніж під час використання

звичайного JavaScript. Основний принцип роботи React базуються на компонентах, окремі блоки коду, які можна перевикористовувати протягом програми. Серед переваг варто виділити високу швидкодію завдяки віртуальному DOM (Document Object Model), велику екосистему з великою кількістю бібліотек і добре підходить для створення інтерактивних і швидких інтерфейсів. Окремо варто відзначити React Native — бібліотека, яка побудована на основі React і використовується для створення мобільних додатків. Але проблема React в тому, що початково він не включає функціонал, такий як управління станами, маршрутизація. Саме це його не робить фреймворком. Також синтаксис React може бути за складним, хоча команда Facebook працює над цим.

Наступним популярним front-end фреймворком є Vue.js. Vue.js – це сучасний JavaScript-фреймворк для розробки користувацьких інтерфейсів. Він надає зручні інструменти та можливості для створення інтерактивних веб-додатків [29]. Vue.js дуже простий у використанні, оскільки всі його компоненти мають схожість зі звичайними HTML та JS. Тому навіть початківці, можуть його легко почати використовувати. Попри таку простоту, він є дуже продуктивним, має достатній функціонал для реалізації будь-якої логіки і відмінно підійде для швидко розгортання на сервері. Варто відмітити, що Vue має багато функцій «з коробки» у порівняні з React. Для прикладу, функції роутера, форми в React впроваджуються окремо. На проти вагу, Vue менше вимагає бібліотек, що полегшує використання та підтримує логічну структуру проєкту.

Останнім, поширеним фреймворком є Angular.js. Angular – це веб-фреймворк, розроблений компанією Google, який використовується для створення динамічних односторінкових додатків, тобто додатки які розміщено лише на одні сторінці [30]. Angular.js є потужним інструментом для розробки складних рішень. Він має безліч переваг, а саме компонентна архітектура, двостороннє зв’язування даних, використання залежностей та ін’єкція, маршрутизація та масштабованість. Angular.js побудований на TypeScript, що

гарантує чітку структуру коду. Але попри це Angular.js є менш гнучким, через строгу типізацію даних. Структура проекту є досить складною і виникає проблема, що розміри файлів є дуже великими, щоб вплине на швидкість завантаження. Також Laravel має погану підтримку Angular, тому він не є найкращим рішенням.

На основі вище наведеного аналізу, для побудови front-end системи було використано Vue.js через його швидкодію, гарну підтримку з Laravel. Vue.js має дуже зрозумілу структуру та документацію, що допомагає швидше реалізувати проєкт у часових рамках виконання кваліфікаційної роботи.

2.5 Висновки проєктування програмного забезпечення

У даному розділі було детально спроєктовано майбутні систему управління транспортом. Першим етапом проєктування було аналіз існуючих видів архітектур ПЗ та вибір найбільш відповідного до вимог. У ході аналізу, було прийнято рішення використовувати клієнт-серверну архітектуру для системи через доступність, яку надає дана архітектура. Окремо було вибрано архітектурний шаблон MVC через його чітко розподілення ролей у коді.

Наступним важливим етап було детальне проєктування модулів системи. На даному етапі, створено багато моделей, які допомогли краще зрозуміти, як варто реалізовувати систему. Створено UML діаграми діяльності, станів та розгортання. Додатково використано DFD моделі.

Окремо було спроєктовано базу даних, щоб забезпечити оптимізоване сховище даних. Завдяки аналізу було вибрано реляційні база даних та вибрано архітектурну стратегію бази даних multi-tenancy.

Останнім етапом даного розділу був вибір стеку технології. Було розглянуто окремо технології для баз даних, front-end та back-end. Було прийнято рішення використовувати СКБД PostgreSQL, PHP фреймворк Laravel та Javascript фреймворк Vue.js.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Розробка бази даних

Основними елементи при роботі з базами даними в Laravel є моделі та міграції. Як зазначалося в попередніх розділах, модель є представленням даних. Для зручної взаємодії Laravel з базою даних використовується ORM, яка називається Eloquent [31]. При використанні Eloquent кожна таблиця бази даних має відповідну «модель», яка використовується для взаємодії з цією таблицею. Окрім вилучення записів із таблиці бази даних, моделі Eloquent також дозволяють вставляти, оновлювати та видаляти записи з таблиці.

Міграції Laravel працюють як контроль версій для бази даних. Вони надають команді розробників можливість визначати та ділитися визначенням схеми бази даних додатка [32]. Таким чином, можна додавати різні елементи, такі як стовпці та таблиці, в базу даних, не звертаючись до її менеджерів, таких як SQLite або phpMyAdmin. Тобто, основним завданням міграцій є визначення таблиць та структури бази даних.

Обидві концепції, в Laravel реалізовані у вигляді класів, які успадковуються від класів. Моделі успадковуються від Laravel класу Model, та міграції від класу Migration.

Іншою важливою концепцією для роботи з Relationship або відношення між моделями [33]. Відношення в Eloquent є відображенням відношень між таблицями в базі даних. Відношення створюються за допомогою методів в класах моделей. Відповідно до типу відношення, використовуються методи, які успадковуються від батьківського класу Model. Існують наступні методи:

– hasOne() встановлює зв'язок "один до одного", де поточна модель має один запис у пов'язаній моделі;

– hasMany() встановлює зв'язок "один до багатьох", де поточна модель має багато записів у пов'язаній моделі;

– belongsTo() встановлює зв'язок "багато до одного", де поточна модель належить до одного запису іншої моделі.

За допомогою цих методів створюються всі типи зв’язку. Також в Laravel є поняття обернені та поліморфічні відношення. Обернений зв'язок створюється, коли використовується hasOne() або hasMany() в одній моделі, і в іншій моделі встановлюється belongsTo(). Поліморфічні відношення це тип відношення, коли одній моделі відносяться декілька інших типів моделей (у звичайних відносинах, одна модель може відноситися лише до однієї моделі).

Як зазначалося у розділі про детальне проєктування модулів системи, оскільки вебсистема планувалася у вигляді SaaS, для бази даних буде використовуватися шаблон multi-tenancy. Для реалізації multi-tenancy потрібно використовувати окрему бібліотеку Tenancy for Laravel. Це потужний інструмент, який дає змогу реалізувати два вида tenancy: single-database tenancy та multi-database tenancy. Це відбувається наступним чином:

– пакет створює міграції, які створює таблицю tenants, де зберігаються всі користувачі, та папку tenants, де зберігаються міграції для баз даних кожного tenant;

– користувачів ідентифікують за допомогою домену, через який увійшов відповідний користувач, або через API токени;

– встановивши tenant, система налаштовується на відповідну базу даних, яка відповідає tenant, обмежує доступ до баз даних інших tenants.

Таким чином, гарантується безпека користувачів та даних. Оскільки дані ізолюються одне від одного і користувачі не можуть побачити дані інших. Щоб вказати які моделі мають бути ізолюваними, пакет має спеціальні класи, які повинні використовуватися в моделях.

Для реалізації вебсистеми для автоматизації логістичних операцій, першочергово варто створити моделі та міграції. Моделі та міграції створюються швидко та легко за допомогою Laravel Artisan. Artisan це інтерфейс командного рядка, що входить до складу Laravel. Artisan знаходиться в кореневому каталозі

додатка як скрипт artisan і надає низку корисних команд, які можуть допомогти вам під час створення додатка.

Першочергово потрібно створити потрібні модель та міграції відповідно до ER діаграми, яка була зображена в попередніх розділах. Створити модель та міграцію можна за допомогою artisan команди:

```
php  artisan make:model <назва моделі> -m
```

Дана команда створює не тільки модель, але й також міграцію одночасно. Всі міграції зберігаються в папці database/migrations і моделі зберігаються в папці app/Models. Таким чином, було створено моделі та міграції, а саме:

– User;

– Company;

– Contract;

– TransportRequest;

– Driver;

– Vechile;

– Route;

– Document;

– Product.

Створивши всі необхідні моделі та міграції, потрібно вказати структуру таблиць, що робиться за допомогою коду. У першу чергу потрібно визначити структуру таблиць. Для прикладу, нижче наведено міграцію для таблиці users. Якщо поглянути на код видно, що кожна міграція складається з двох методів: up та down. Метод up використовується власне коли таблиця створюється, а down коли відбувається відкат бази даних.

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
```

```
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('users');
    }
};
```

З коду можна побачити, що Laravel має клас Schema, який відповідає за створення таблиці або використання вже існуючої. Будь-який метод приймає функцію, де передається параметр класу Blueprint. Даний метод відповідає за створення атрибутів таблиці. А власне в методі down, відбувається видалення таблиці, якщо вона існує.

Всі міграції створюються в папці database/migrations/tenant, оскільки всі стосуються саме tenant. Після створення та визначення структур всіх міграцій, їх потрібно запустити, що робиться за допомогою Artisan команди:

```
php  artisan migrate
```

Після успішного створення таблиці, залишається заповнити відповідні моделі. Як зазначалося раніше, моделі це класи які успадковується від класу Model. Приклад моделі можна побачити нижче:

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class Contract extends Model
```

```
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'customer_id',
        'deadline',
        'total'
    ];

    protected $with = [
        'customer',
        'products'
    ];

    public function customer() : BelongsTo
    {
        return $this->belongsTo(Customer::class);
    }

    public function products() : HasMany
    {
        return $this->hasMany(ContractProduct::class, "contract_id");
    }

    public function transportartions(): HasMany
    {
        return $this->hasMany(Transportartion::class, "contract_id");
    }
}
```

На початку класу, використовується клас бібліотеки Tenancy for Laravel, щоб зазначити, що даний клас має бути ізолюваними, тобто об’єкти належать тільки одному tenant. Наступним є важливе поле fillable, яке визначає, які атрибути можна заповнювати. Далі нижче створені відношення між класами і використовується поле with, яке визначає, які відношення автоматично добавляти до даних цієї моделі. Тепер, коли буде повертатися моделі Contract, будуть повертатися дані моделі і дані про Customer та ContractProduct. Таким чином була реалізована вся база даних відповідно до ER-діаграми.

3.2 Опис UI інтерфейсу вебсистеми

Однією з нефункціональних вимог є створення інтуїтивно-зрозумілий інтерфейс. Для того, щоб правильно спроектувати вебсистему і зробити її зручною, буде корисним спроектувати дизайн. Проєктування інтерфейсів

відбувається в програмі Figma. Figma — це потужний інструмент, який дозволяє створювати інтерфейси для різних платформ і створювати різні діаграми. Основною функцією Figma є можливість командної роботи, що дозволяє працювати над проєктом командно.

Створення дизайну — це комплексний процес. Він включає не тільки створення візуальної частини, але й аналіз предметної області, що вже зроблено, створення прототипів, тестування та проводження інтерв'ю з користувачами. І це не повний список процесів, які включає створення дизайну інтерфейсу. У даній кваліфікаційній роботі деякі процеси опущені, через обмежений час. Тому на основі вимог, було створено одразу дизайн. Інтерфейс включає велику кількість сторінок.

При вході у вебсистему, користувач спершу має заповнити форму входу (рисунок 3.1). Якщо користувач не має аккаунта, тоді він повинен створити профіль за допомогою форми реєстрації (рисунок 3.2).

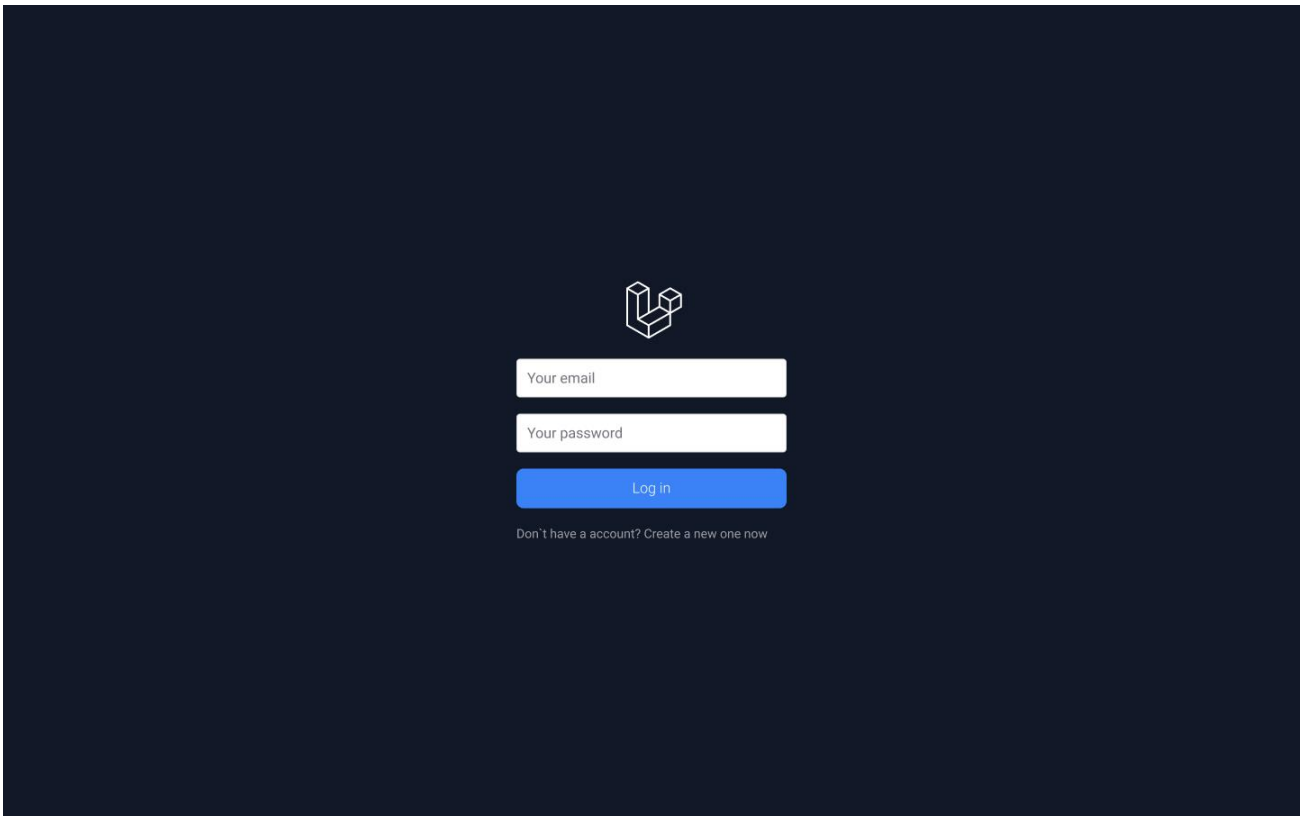


Рисунок 3.1 — Форма входу у систему

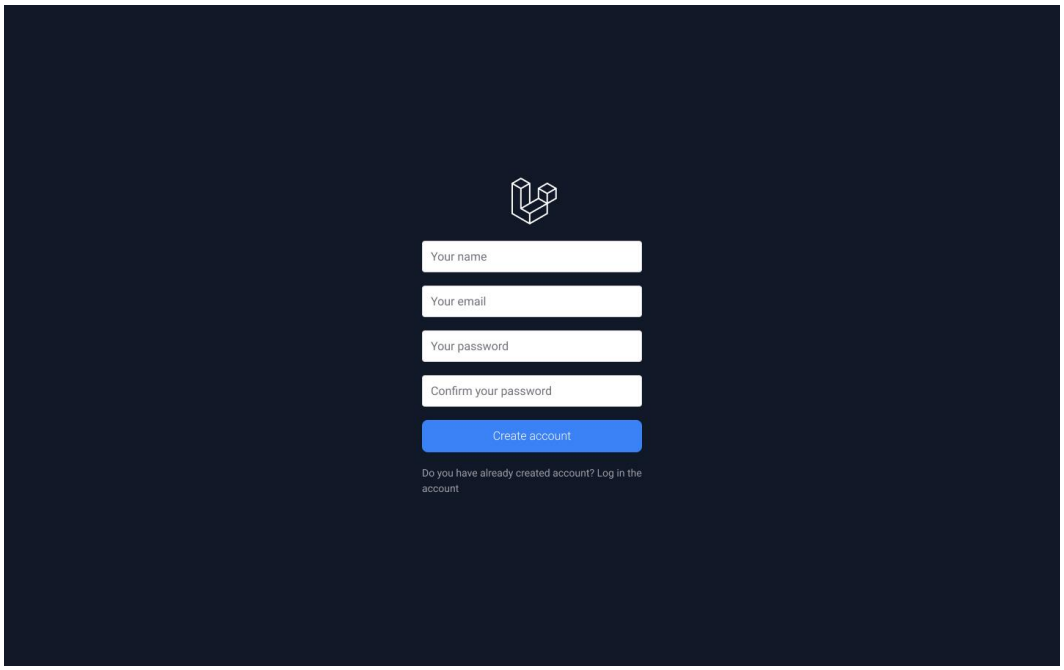


Рисунок 3.2 — Форма реєстрації аккаунта

Після успішної реєстрації аккаунта, користувач має вибрати тип профіля: або власник логістичної компанії, або працівник (рисунок 3.3). Якщо це власник компанії, тоді він повинен ввести базову інформацію про компанію (рисунок 3.4).

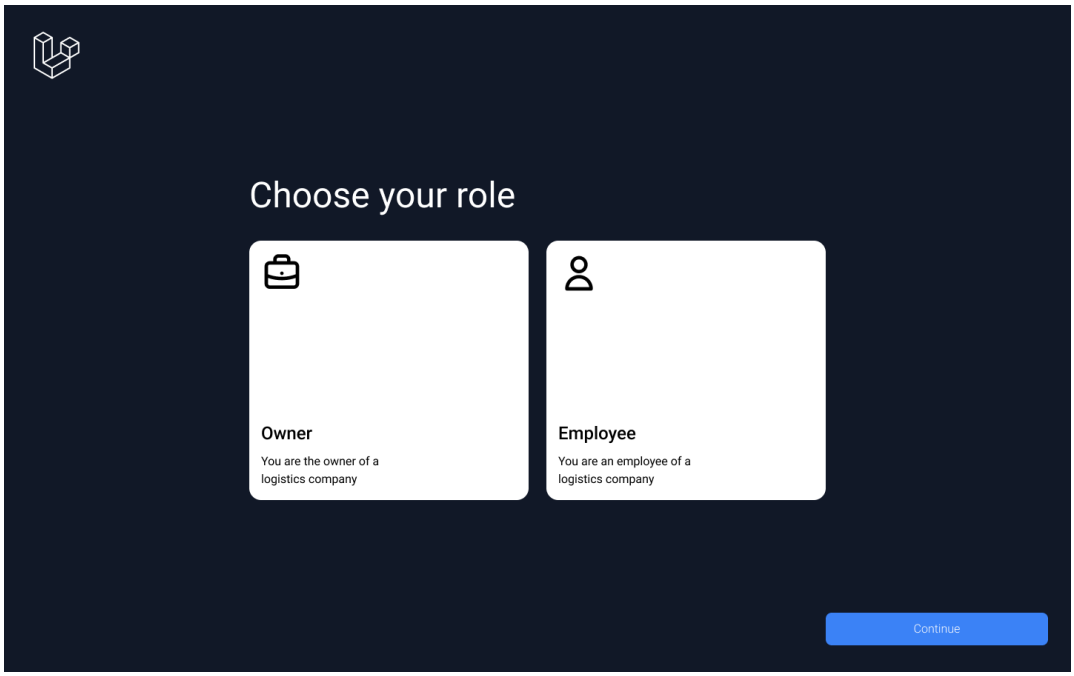


Рисунок 3.3 — Вибір тип профілю після реєстрації

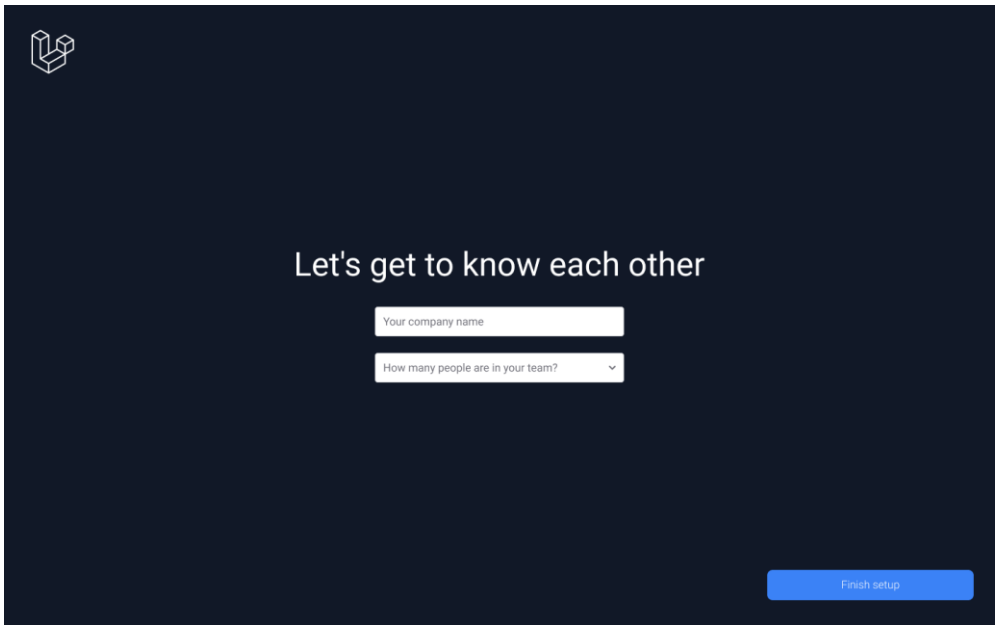


Рисунок 3.4 — Заповнення інформації про компанію

Після успішної авторизації, користувач попадає у профіль. На всіх сторінках з лівою сторони розміщена навігаційна панель, а справа відображається контент відповідної сторінки. Першою сторінкою, коли користувач попадає у профіль є сторінка, яка містить загальну інформацію, яка може знадобитися (рисунок 3.5).

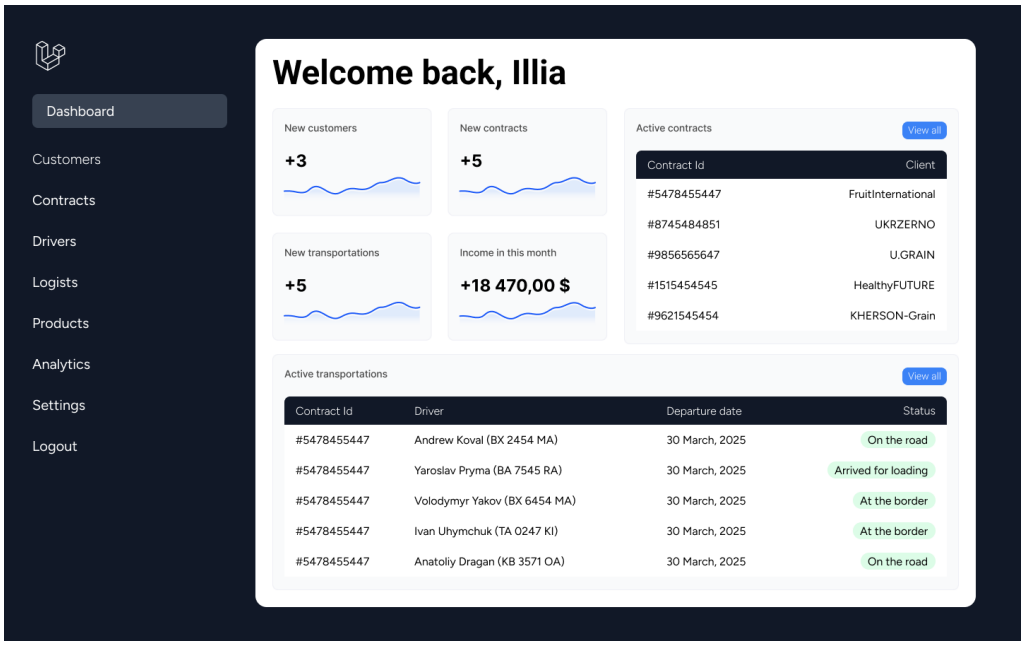


Рисунок 3.5 — Головна сторінка вебсистеми

Далі користувач може перейти на інші сторінки, що зробити певні дії. Більшість сторінок на сайті реалізовують CRUD і мають схожі елементи управління. На сторінка відображення списку записів, використовується таблиця (рисунок 3.6). Кожна така сторінка має кнопку, яка веде на сторінку створення запису. Дані сторінки містять форми для створення запису в базі даних (рисунок 3.7).

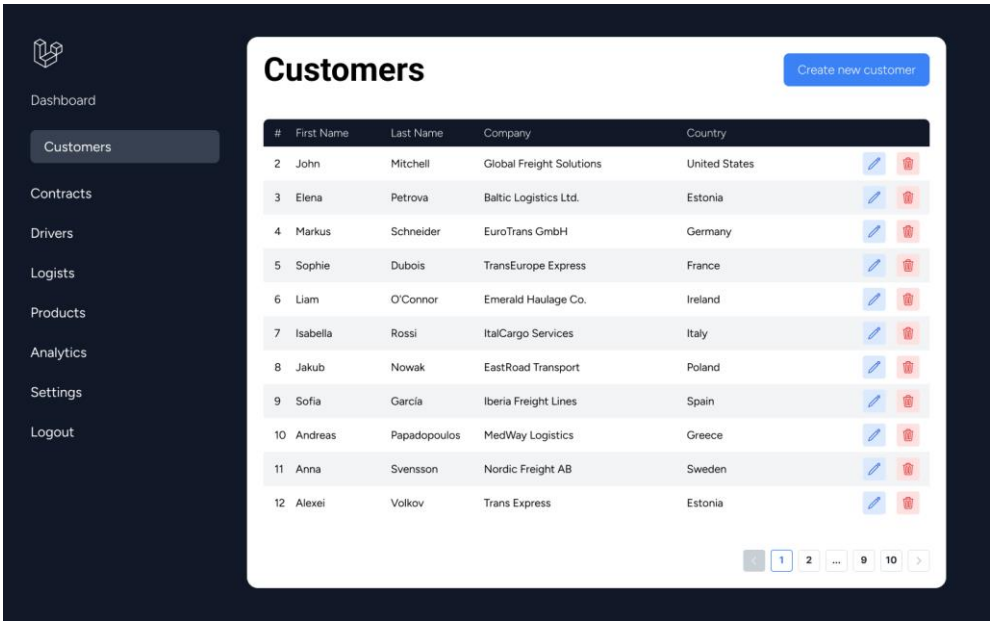


Рисунок 3.6 — Сторінка всіх клієнтів

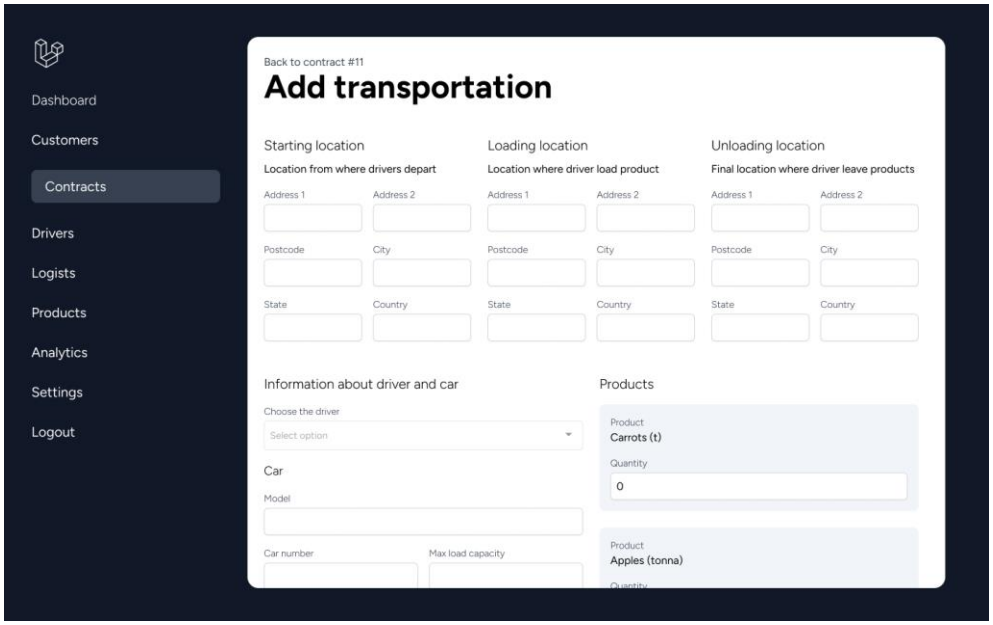


Рисунок 3.7 — Сторінка створення перевезення

Кожен запис можна видалити і редагувати. Сторінки редагування ідентичні до сторінок створення, лише мають заповненні дані (рисунок 3.8).

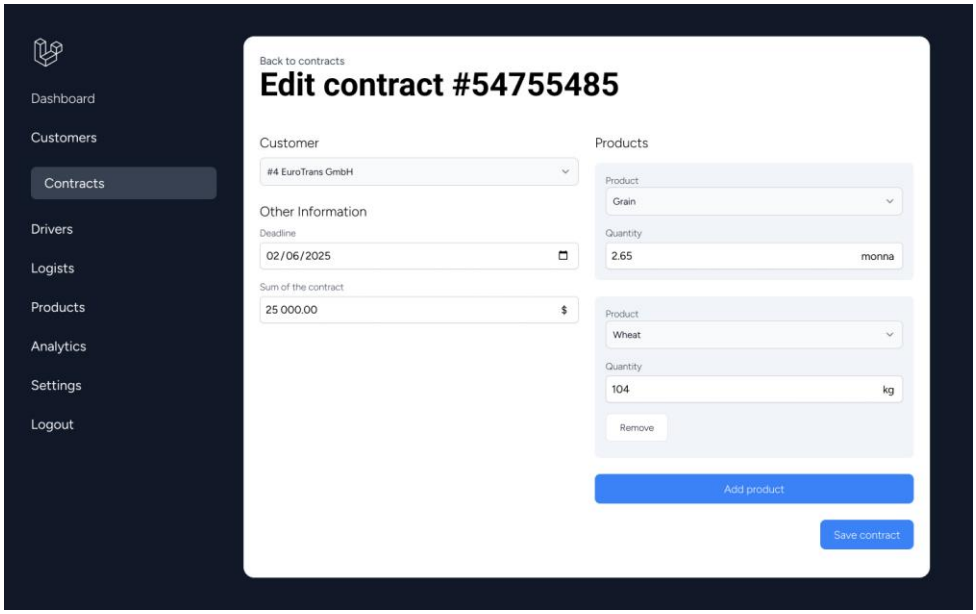


Рисунок 3.8 — Сторінка редагування даних про контракт

Останньою сторінкою, яка була спроєктованою є сторінка аналітики. Основна мета даної сторінки — це відображення даних у графічній спосіб для кращого розуміння. Дана сторінка містить дані про різні параметри компанії та графіки для порівняння результатів з попередніми місяцями (рисунок 3.9).

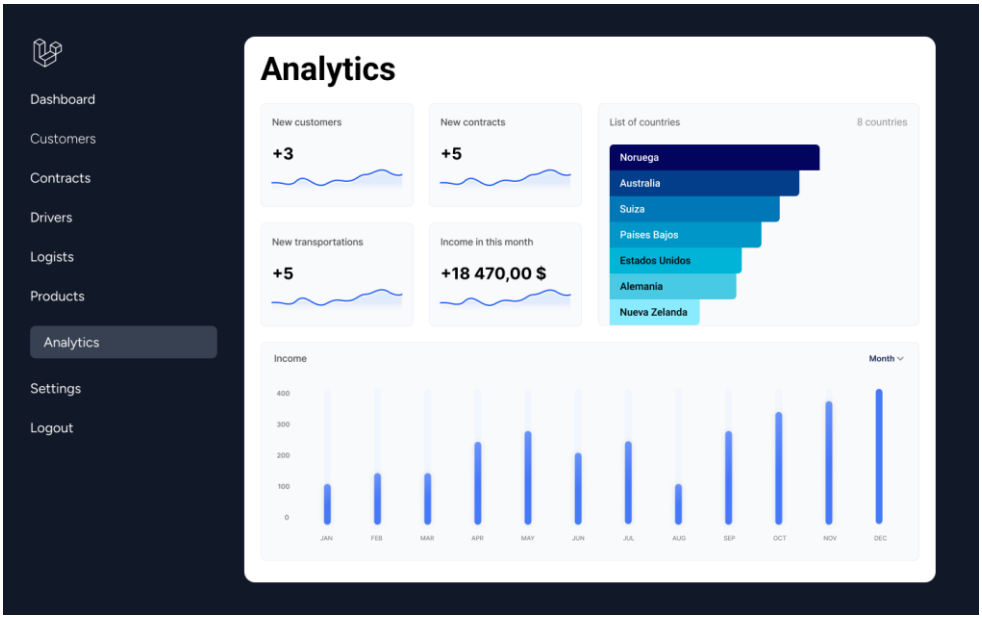


Рисунок 3.9 — Сторінка аналітики

У результаті, було спроєктована мінімалістичний та інтуїтивно зрозумілий дизайн вебсистеми у Figma. Протягом створення дизайну, було дотримано єдиний стиль і вебсистема виглядає цілісною.

3.3 Реалізація вебсистеми

Після успішного створення бази даних, наступним етапом є реалізація клієнтської та серверної частини вебсистеми. Створення моделей та міграцій можна вважати серверною частиною. Для роботи з даними моделей не достатньо. Для повноцінної роботи вебдодатка, використовуючи шаблон MVC, потрібно створити представлення та контролери. Згідно шаблону MVC, основне призначення представлень є відображення даних та отримання даних від користувачів. Дані вже опрацьовує контролер, а не модель, як можна подумати.

По замовчуванню, представлення в Laravel створюється за допомогою шаблонізатора Blade. Blade — це простий, але потужний механізм шаблонів [34]. На відміну від деяких механізмів шаблонів PHP, Blade не обмежує використання простого коду PHP у ваших шаблонах. Фактично, всі шаблони Blade компілюються у простий код PHP і кешуються до моменту їх модифікації, що означає, що Blade практично не додає навантаження на вашу програму. Файли шаблонів Blade мають розширення .blade.php і зазвичай зберігаються в каталозі resources/views.

Як можна зрозуміти, Vue.js не є вбудованим в Laravel і для роботи з ним потрібно окреме рішення. Таким рішенням є бібліотека Inertia.js. Inertia — це бібліотека, яка дозволяє будувати SPA, або Single Page Application, на основі Vue.js або React.js без зміни стеку в Laravel. Inertia виступає своєрідним «містом» між бекендом та фронтендом. Тобто замість того, щоб повертати чистий PHP код, Inertia повертає набір компонентів, які потрібні для сторінки і Vue.js генерує код на основі цих компонентів. Всі файли зберігаються в папці resources/js.

Для створення привабливого інтерфейсу використовувалася CSS бібліотека стилів Tailwind. Дана бібліотека має привабливий дизайн та легко налаштовані стилі, що значно скорочує час створення інтерфейсу.

Майже всі сторінки вебсистеми, мають схожу загальну структуру. На рисунку 3.10 можна побачити графічне зображення структури сторінок.

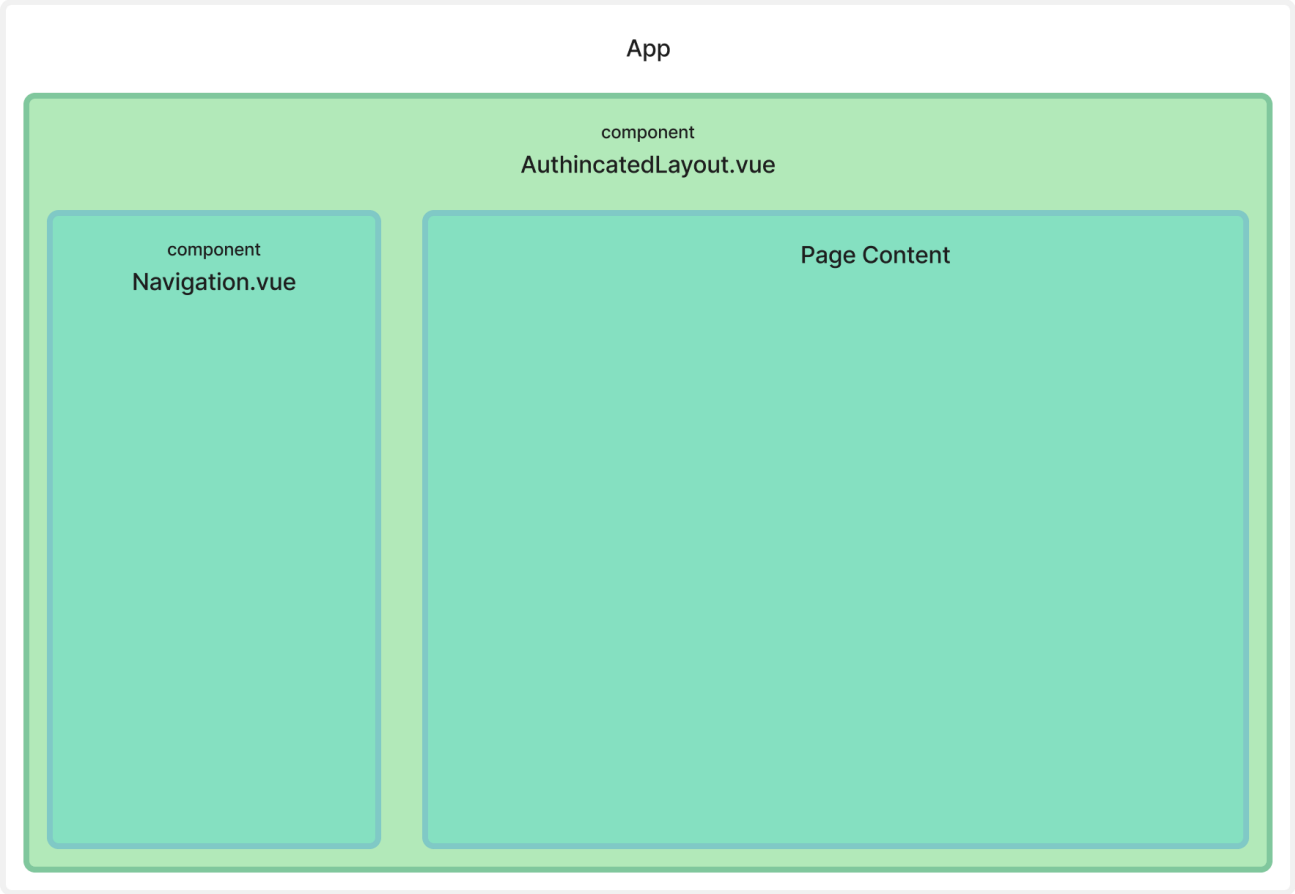


Рисунок 3.10 — Загальна структура сторінка вебсистеми

З рисунку можна побачити, що кожна сторінка обертається в обгортку у вигляді AuthincatedLayout.vue. Даний компонент містить елементи, які можуть повторюватися протягом всіх сторінок. Для сторінки авторизації, замість AuthincatedLayout використовується GuestLayout, який не містить компонент Navigation.vue, оскільки на даних сторінках немає навігації. У обгортку передається контент конкретної сторінки, що вставляється за допомогою slot.

Контент кожної сторінки зберігається у папці Pages. Кожна сторінка має свій окремий файл, де зберігається весь контент сторінки. Нижче наведено приклад коду для сторінки створення водіїв.

```
<script setup>
import InputError from "@/Components/InputError.vue";
// підключення бібліотек

const props = defineProps({
    drivers_companies: Array | Object,
});

const showCompanyCreatingForm = ref(!props.drivers_companies.length);
//

const form = useForm({
    // тут вказуються дані, які потрібні для відправки
});

const submitForm = () => {
    isSubmitted.value = true;

    let validated = true;

    // валідація даних

    if (validated) {
        form.post(route("drivers.store"));
    }
};
</script>

<template>
    <AuthenticatedLayout>
        <div>
            <div class="flex flex-col">
                <a :href="route('drivers.index')" class="text-gray-700"
                    >Back to drivers</a
                >
                <h2 class="text-5xl font-bold">Create driver</h2>
            </div>
            <div class="mt-12 h-full">
                <form class="flex gap-6" @submit.prevent="submitForm">
                    <div class="basis-1/2 grow shrink">
                        <div class="mb-6">
                            <Multiselect
                                :options="driversCompanies"
                                v-model="choosenDriverCompany"
                                // вказуються різні props
                            ></Multiselect>

                           // продовження контенту
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </AuthenticatedLayout>
</template>
```

```
<style src="vue-multiselect/dist/vue-multiselect.min.css"></style>
```

З вищенаведеного коду можна побачити, що є три основних компоненти: script, template та style (може опускатися). У тегах script прописується весь код взаємодії користувача з елементами інтерфейсу. Елемент template ж включає розмітку компонента, або просто кажучи модифікований HTML код. Теги style включають таблицю стилів, або підключають зовнішній файл. Така структура використовується по всіх файлах vue.

Якщо запустити проект, то нічого не буде відображатися. Причиною цьому буде, де що не прописано маршрути. Laravel має зручну систему маршрутизації, що дозволяє легко визначити маршрути, які будуть присутніми у вебсистемі. Всі маршрути зберігаються в папці routes, де є декілька різних файлів. Основна робота виконується у:

– web.php — маршрути, що доступні для всіх користувачів;

– api.php — маршрути, що використовуються для REST API;

– tenant.php — маршрути, що використовуються в кожному окремому tenant.

Для роботи з маршрути, Laravel має окремий клас Route, який має статичні методи для визначення типу маршруту. Для прикладу, наведено визначення POST маршруту:

```
Route::post("contracts/",[ContractController::class, "store"]);
```

Даний рядок коду виконує, що за посиланням {домен}/contracts/ при використані POST метода — буде опрацьовуватися метод store класу ContractController. Під кожен HTTP метод, є відповідний метод класу Route, що дає змогу вказувати всі види методів для маршрутів. Таким чином прописуються всі маршрути. Але Laravel має багато методів, спростити цей процес. Якщо певний контролер містить лише базові методи для CRUD, тоді можна використати метод

```
Route::resources(["contracts" => ContractController::class]);
```

У такому разі, Laravel сам створить всі необхідні маршрути. Це значно спрощує процес маршрутизації. Після успішного створення маршрутів, необхідно прописати відповідні алгоритми для обробки даних. Це прописується в контролерах. Багато контролерів мають схожі методи, оскільки вони реалізовують CRUD і використовуються стандартний набір. Щоб створити Controller, також можна використовувати Artisan команду make:controller. Всі контролери зберігаються в папці app/Http/Controllers. Нижче наведено приклад такого контрелера:

```
class DriverController extends Controller
{

    public function index()
    {
        $drivers = Driver::all();

        return Inertia::render('Drivers/Index', [
            'drivers' => $drivers,
        ]);
    }

    public function create()
    {
        $driversCompanies = DriversCompany::all();

        return Inertia::render('Drivers/Create', [
            'drivers_companies' => $driversCompanies,
        ]);
    }

    public function store(StoreDriverRequest $request)
    {
        $validated = $request->validated();

        $driver_company_data = $validated['drivers_company'];

        $driver_company = null;
        if(!empty($driver_company_data['id'])){
            $driver_company = DriversCompany::find($driver_company_data['id']);
        }else{
            $driver_company = DriversCompany::create($driver_company_data);
        }

        $driver_data = $validated['driver'];
        $driver_data['drivers_company_id'] = $driver_company->id;

        $driver = Driver::create($driver_data);

        return Inertia::render('Drivers/Index', [
            'drivers' => Driver::all(),
        ]);
    }
```

```
    public function edit(Driver $driver){
        return Inertia::render('Drivers/Edit', [
            'driver' => $driver,
            'drivers_companies' => DriversCompany::all(),
        ]);
    }

    public function update(UpdateDriverRequest $request, Driver $driver)
    {
        $validated = $request->validated();

        $driver_company_data = $validated['drivers_company'];

        $driver_company = null;
        if(!empty($driver_company_data['id'])){
            $driver_company = DriversCompany::find($driver_company_data['id']);
        }else{
            $driver_company = DriversCompany::create($driver_company_data);
        }

        $driver_data = $validated['driver'];
        $driver_data['drivers_company_id'] = $driver_company->id;

        $driver->fill($driver_data)->save();

        return Inertia::render('Drivers/Index', [
            'drivers' => Driver::all(),
        ]);
    }

    public function destroy(Driver $driver){
        $driver->delete();

        return back()->with('data', [
            'drivers' => Driver::all(),
        ]);
    }
}
```

Як видно з коду, контролер містить стандартні RESTful методи:

– index — відображує список всіх записів моделі;

– show — відповідає за показ конкретного запису моделі;

– create — показує форму для створення запису моделі;

– store — відповідає саме за обробку даних форми create і збереження нового запису в базі даних;

– edit — показує форму для редагування певного запису;

– update — обробляє дані з форми edit та відповідає за оновлення запису в базі даних;

– delete — відповідає за видалення запису з бази даних.

Варто зазначити, що кожен метод повинен повертати якусь відповідь. У даному випадку обов'язково повинно повертатися компоненти Vue.js. Як можна побачити, за це відповідає саме клас Intertia. Або якщо не потрібно генерувати нову сторінку і треба повернути дані на ту саму сторінку, то використовується допоміжна функція back(), що повертає результат на ту саму сторінку.

Деякі методи отримують параметр request. Це інша важлива можливість Laravel [35]. Requests це спеціальні класи, які потрібні для отримання даних користувача і валідації та авторизації HTTP- запитів. У даних класах, можна зазначити правила, яким мають відповідати дані отримані від користувача. У разі успіху, дані передаються до контролера. У разі помилки, тобто якійсь дані не відповідають зазначеним правилам, тоді повертається помилка з певним повідомлення. Для прикладу, нижче наведено клас StoreDriverRequest:

```
class StoreDriverRequest extends BaseRequest
{
    public function rules(): array
    {
        return [
            'drivers_company' => ['required', 'array'],
            'drivers_company.id' => ['nullable', 'exists:drivers_companies,id'],
            // вказуються правила валідації
        ];
    }

    public function messages(): array
    {
        return [
            'drivers_company.required' => 'The driver\'s company field is
required.',
            'drivers_company.array' => 'The driver\'s company must be an
array.',
            // вказуються повідомлення, які мають повертатися для конкретного
випадку
        ];
    }
}
```

Всі Requests містять два методи: rules та messages. Метод rules повертає масив з правилами валідації. Можуть використовуватися безліч правил. Laravel має багато готових правил, але можна також створювати власні. Метод messages, повертає масив повідомлень, які мають повертатися при помилці. Ключем в

масив виступає назва атрибуту, яке було провалідовано, і правило на якому була помилка.

Таким чином створюється серверна та клієнтські частини в Laravel. Це не весь функціонал Laravel. Він має багато інших корисних функції, які можуть стати в нагоді при розробці і оптимізувати роботу вебзастосунку. На рисунку 3.11 зображено структуру проєкту і папки в яких в основному проводилася робота.

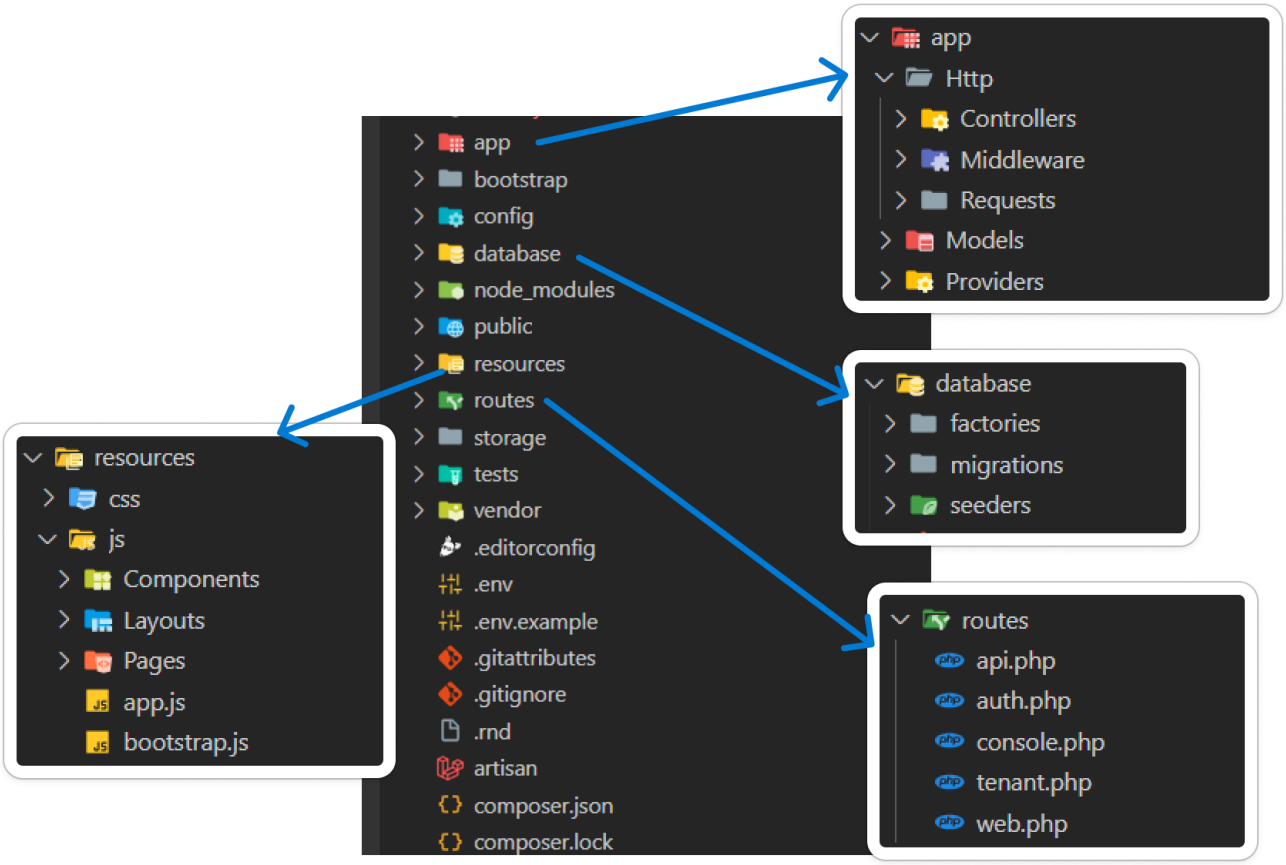


Рисунок 3.11 — структура проєкту вебсистеми

3.4 Тестування вебсистеми

3.4.1 Вибір та обґрунтування методів тестування вебсистеми

Заключним етапом циклу розробки програмного забезпечення є тестування. Саме на етапі тестування перевіряється, чи виконано та враховано

вимоги до програмного забезпечення, які були поставлені на початку проєкту. Даний процес називається валідація.

Існує багато видів тестування і класифікують по різних параметрах. У рамках даної кваліфікаційної, було здійснено функціональне та нефункціональне тестування [36]. Як зрозуміло з назви, функціональне тестування відповідає за тестування функціональних вимог, а нефункціональне — нефункціональних вимог. Серед цих видів, можна виділити наступні види тестування:

– модульне тестування — тестування ізольованого модуль програмного забезпечення обмеженого у власному середовищі, тобто зв'язки з іншими модулями не враховується;

– інтеграційне тестування — тестування, при якому перевіряється, як окрему компоненти системи можуть взаємодіяти між собою без рахування зовнішніх факторів;

– end-to-end тестування — форма тестування, яка перевіряє функціональності системи, відтворюючи реальні сценарії використання та перевіряючи роботу компонентів між собою;

– тестування сумісності — нефункціональне тестування, яке перевіряє як працює система на різних платформах;

– тестування продуктивності — процеси перевірки різних параметрів продуктивності, а саме: швидкість завантаження, робота при максимальних показниках і інші;

– тестування користувацького досвіду — перевірка інтерфейс на комфортність та зрозумілості користування для кінцевого користувача.

Для проведення тестування, існують різні інструменти, які можуть робити тестування автоматизовано так і в ручну. PHP та Laravel має наступні інструменти для тестування:

– PHPUnit – це PHP фреймворк для модульного тестування, який дозволяє перевіряти правильність роботи окремих частин коду — переважно функцій та методів [37];

– Pest – мінімалістичний фреймворк на основі PHPUnit, який в основному підходить до інтегрованого тестування [38];

– Laravel Dusk – фреймворк, який дозволяє проводити end-to-end тестування, симулюючи користування вебдодатком у браузері [39].

Нефункціональні тестування мають різні інструменти. Для тестування продуктивності, найбільш популярним є Google PageSpeed Insights. Для тестування користувацького досвіду використовують різні інструменти одночасно, оскільки це досить комплексний процес. Для прикладу, інтерв'ю з користувачами, фокус групи і інші.

Для забезпечення структурованості тестування, може підійти підхід до документування через Test Case. Test Case – це своєрідний артефакт, який описує сукупність кроків, конкретних умов та параметрів, необхідних для перевірки реалізації функції на відповідність вимогам прямим або непрямим [40]. Зазвичай test case складається з декількох атрибутів:

– унікальний ідентифікатор тесту, який використовується для позначення;

– назва тесту;

– основна ідея або короткий опис змісту;

– список дій, котрі не мають прямого відношення до функціоналу котрий перевіряється;

– порядок дій, які потрібно виконати для отримання очікуваного результат;

– результат, який очікується при успішному виконанню тесту.

Тест може мати декілька значень: позитивний (pass), негативний (fail) та виконання тесту заблоковано (blocked). Такий підхід дозволяє не лише перевіряти функціональність програмного забезпечення, а й відстежувати якість його реалізації протягом усього життєвого циклу розробки. Використання тест-кейсів забезпечує повторюваність тестів, полегшує виявлення помилок, спрощує регресійне тестування та підвищує прозорість процесу тестування для команди розробників і зацікавлених сторін.

3.4.2 Результати виконання тестів

На основі аналізу інструментів тестування, було створено потрібні тести для тестування функціоналу системи і виконано. Результат тестування можна побачити на рисунку 3.12.

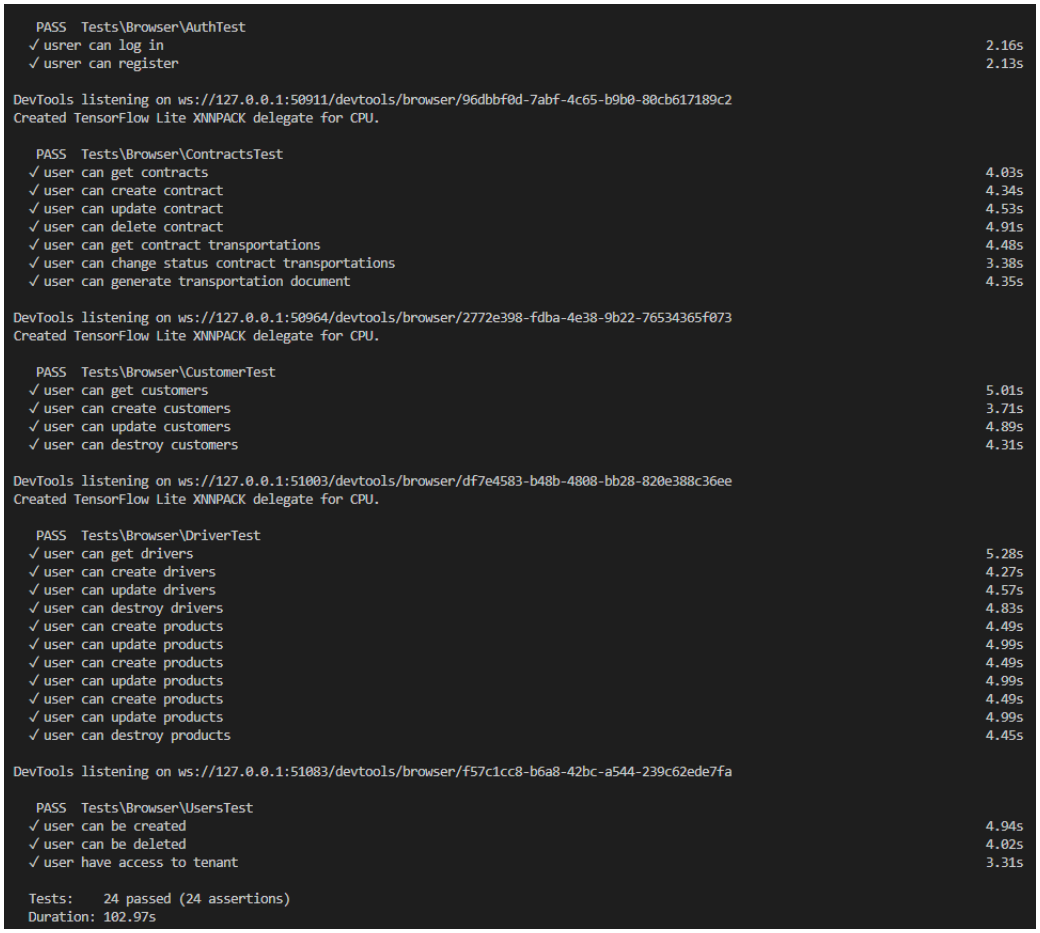


Рисунок 3.12 — Результати тестування

Першим тестом була перевірка коректності роботи авторизації та аутентифікації. Основною ціллю було перевірити чи правильно відбувається вхід та реєстрації в систему. У таблиці 3.1 описано всі тестові випадки, які перевірялися у даному тестів. Всі відповіді системи під час виконання тестування записувалося згідно стандартами Test Case Management. Таким чином вдалося зберегти структурний підхід до тестування.

Таблиця 3.1 — Тестування авторизації та аутентифікації

Дія	Очікуваний результат	Успішність
ПЕРЕДУМОВИ		
Відкриття вебсистеми через посилання	Відображається сторінка входу	Passed
ОПИС ТЕСТОВОГО ВИПАДКУ		
Натискання кнопки «Увійти» після введення даних для входу	Користувача перенаправляє на сторінку домашню сторінку панелі управління	Passed
Натискання кнопки «Увійти» з не вірно введеними даними для входу у систему	Користувачу виводитися помилка	Passed
Натискання на посилання «Не маєте аккаунта? Створіть новий аккаунт зараз»	Користувача перенаправляє на сторінка реєстрації	Passed
Натискання на кнопку «Зареєструватися» при вірно введених даних	Створюється профіль користувач і перенаправляється на сторінку налаштування	Passed
Натискання на кнопку «Завершити реєстрацію»	Завершується реєстрація компанії, створюється tenant і користувача переноситься на домашню сторінку панелі керування	Passed

Наступним етапом тестування було перевірка функціонал для авторизованих користувачів. Спершу було відбулося тестування функціоналу CRUD для клієнтів, тобто можливість перегляду, створення, редагування та видалення записів. У таблиці 3.2 описано тестові випадки для перевірки бази клієнтів.

Таблиця 3.2 — Тестування функціонал бази клієнтів

Дія	Очікуваний результат	Успішність
ПЕРЕДУМОВИ		
Відкриття сторінки бази клієнтів	Відображається сторінка з записами з бази клієнтів	Passed
ОПИС ТЕСТОВОГО ВИПАДКУ		
Натискання на певний запис	Відкривається інформація про запис	Passed
Натискання на кнопку видалення	Запис про клієнта видаляється з бази даних	Passed
Натискання на кнопку «Добавити клієнта»	Користувача перенаправляє на сторінку створення	Passed

Продовження таблиці 3.2

Натискання кнопки «Добавити» на сторінці додавання клієнта	Додається запис про клієнтів в базу даних	Passed
Натискання на кнопку редагування	Користувача перенаправляє на сторінку редагування запису	Passed
Натискання на кнопку «Зберегти» на сторінці редагування	Запис про певного клієнта оновлюється новою інформацією	Passed

Наступне було перевірено функціонал CRUD для товарів та водіїв. Дані тестування майже ідентичні, як перевірка функціоналу бази клієнтів. Опис тестових випадків можна побачити на таблицях 3.3 та 3.4.

Таблиця 3.3 — Тестування функціоналу бази товарів

Дія	Очікуваний результат	Успішність
ПЕРЕДУМОВИ		
Відкриття сторінки бази товарів	Відображається сторінка бази товарів	Passed
ОПИС ТЕСТОВОГО ВИПАДКУ		
Натискання на певний запис в списку	Відкривається інформація про товар	Passed
Натискання на кнопку видалення	Запис про товар видаляється з бази	Passed
Натискання на кнопку «Добавити товар»	Відкривається модальне вікно з формою	Passed
Натискання кнопки «Добавити» на формі додавання товару	Додається запис про товар в базу даних	Passed
Натискання на кнопку редагування	Відкривається сторінка редагування товару	Passed
Натискання на кнопку «Зберегти» на сторінці редагування	Запис про товар оновлюється новою інформацією	Passed

Таблиця 3.4 — Тестування функціоналу бази водіїв

Дія	Очікуваний результат	Успішність
ПЕРЕДУМОВИ		
Відкриття сторінки бази водіїв	Відображається сторінка бази водії	Passed
ОПИС ТЕСТОВОГО ВИПАДКУ		
Натискання на певного водія в списку	Відкривається інформація про водія	Passed
Натискання на кнопку видалення	Водій видаляється з бази даних	Passed
Натискання на кнопку «Добавити водія»	Відкривається сторінка з формою	Passed
Відправка форми додавання водія	Додається запис про водія в базу даних	Passed
Натискання на кнопку редагування	Відкривається сторінка редагування водія	Passed
Відправка форми редагування запису	Запис про водія оновлюється	Passed

Об’ємним тестуванням було перевірка роботи з контрактами та перевезеннями. В таблиці 3.5 описано тестові випадки для цього функціоналу.

Таблиця 3.5 — Тестування контрактів та перевезень

Дія	Очікуваний результат	Успішність
ПЕРЕДУМОВИ		
Відкриття сторінки бази контрактів	Відображається сторінка бази контрактів	Passed
ОПИС ТЕСТОВОГО ВИПАДКУ		
Натискання на певний контракт в списку	Відкривається інформація про контракт разом з переліком перевезень у рамках контракту	Passed
Натискання на кнопку видалення контракту	Контракт перенаправляється у корзину	Passed
Натискання на кнопку «Створити контракт»	Відкривається сторінка з формою створення контракту	Passed
Вибір клієнта з бази клієнтів при створені контракту	Відображення списку клієнтів, з якого користувач вибирає певного клієнта	Passed
Вибір товарів з бази товарів при створені контракту	Відображення списку товарів, з якого користувач вибирає товари, які потрібно перевезти з однієї локації в іншу.	Passed
Відправка форми додавання контракту	Додається запис про контракт в базу даних	Passed
Натискання на кнопку редагування	Відображається сторінка редагування контракту	Passed
Відправка форми редагування запису	Запис про контракт оновлюється даними з форми	Passed
Натискання на кнопку «Добавити перевезення» на сторінці контракту	Відкривається сторінка з формою для додавання перевезення	Passed
Натискання на кнопку видалення перевезення	Перевезення видаляється	Passed
Натискання на кнопку редагування перевезення	Відображається сторінка з формою редагування для зміни статусу перевезення	Passed
Відправка форми редагування перевезення	Статус перевезення змінюється	Passed
Натискання на кнопку «Генерація документів для перевезення»	Появляється модальне вікно з вибору виду документа	Passed
Натискання на кнопку «Генерація документа» в модальному вікні генерації документів	Перенаправлення на сторінку з попереднім переглядом документа у форматі PDF	Passed

Останнім тестом було перевірка безпеки. Перевірялося чи користувачам доступні лише їхні дані і вони не можуть переглядати дані інших. Додатково була перевірка ролів користувачів. Опис тестових випадків наведений у таблиці 3.6.

Таблиця 3.6 — Тестування безпеки

Дія	Очікуваний результат	Успішність
ПЕРЕДУМОВИ		
Користувач входить у систему у ролі логіста	Перенаправлення на головну сторінку панелі управління	Passed
ОПИС ТЕСТОВОГО ВИПАДКУ		
Перегляд даних з бази даних (для прикладу, перегляд клієнтів, контрактів)	Відображення лише тих записів, які відносяться лише для конкретного tenant	Passed
Перехід за посиланням, яке веде до запису іншого tenant	Перенаправлення на сторінку входу, з повідомленням «Ви не маєте права доступу до даного запису»	Passed
Відобразити меню панелі управління у профіля логіста	Відсутній пункт меню «Аналітика» та «Логісти»	Passed
Перехід за посиланням, яке веде до сторінки аналітики або логістів	Відображення головної сторінки панелі керування, замість сторінки аналітики	Passed

3.5. Висновки програмної реалізації

У даному розділі, було детально описано процес програмної реалізації вебсистеми для автоматизації логістичних операцій. Спершу було реалізовано базу даних, що включало створення міграцій та моделей. Щоб забезпечити ізоляцію даних та безпеку, було реалізовано шаблон бази даних multi-tenancy.

Для забезпечення зручного інтерфейс, було спроектовано користувацький інтерфейс у Figma. Було спроєктовано кожну сторінку, яка передбачалася у вебсистемі. Використовуючи макети, було створено Vue.js компонентів для представлень і контролери, де було реалізовано обробку даних та бізнес-процесів.

Заключним етап було тестування, під час якого проведено перевірку всіх модулів на відповідність функціональним вимогам, і виправити баги. Щоб впевнитися, що система відповідає вимогам безпеки, проведено тестування безпеки.

У результатів, було отримано працюючу систему, яка відповідає всім вимогам поставлених на початку та готова до впровадження. Після провадження, система готова до подальшого вдосконалення та реалізації нового функціоналу.

ВИСНОВКИ

У результаті в даній кваліфікаційній роботі було пройдено та описано всі етапи проєктування та реалізації вебсистеми для автоматизації логістичних операцій з використанням сучасних технологій розробки. Дана робота показує весь цикл розробки програмного забезпечення від етапу проєктування інтерфейсу та користувацького досвіду до його наступної реалізації.

Першим етапом розробки вебсистеми був детальний аналіз предметної області. Даний етап є крайні важливим, оскільки на цьому кроці формується загальне розуміння розроблюваного програмного забезпечення та загальні вимоги до програми.

Початково було проведено аналіз логістики, як сфери в загальному. У результаті було виділено логістичні операції, які можна автоматизувати. На основі цього, було проаналізовано наявні програмно-технічні рішення на ринку. Аналіз включав порівняльну характеристику трьох систем управління транспортом.

Завдяки аналізу предметної області було поставлено чітко завдання та сформовано функціональні та нефункціональні вимоги до вебсистеми. Додатково створено UML діаграми варіантів використання, які відображають всіх зацікавлених користувачів та функціонал, який вони повинні мати.

Наступним етапом створення вебсистеми для автоматизації логістичних операцій було детальне проєктування програмного забезпечення. Даний етап включає детальне моделювання системи в загальному, бази даних та вибір стеку технологій, які використовувався у реалізації програмного забезпечення.

Спершу було проведено аналіз та вибір архітектури програмного забезпечення. Під час аналізу проведено порівняльну характеристику всіх видів архітектури та шаблонів. Основні цього було прийнято рішення використовувати клієнт-серверну архітектуру, оскільки вона найкраще підходить для вебдодатків.

Наступним кроком було детальне проєктування всіх програмних модулів програмного забезпечення. Даний етап включав створення таких діаграм як: діяльності, станів, потоків даних та розгортання. Завдяки розробленим діаграм сформувалося чітке розуміння бізнес-процесів, які модулі будуть наявні та взаємодія між ними. Додатково, спроектовано як дані будуть передаватися між модулями та звідки дані будуть надходити. Завдяки детальному конструюванню, було чітко створено план реалізації системи.

Окрім проєктування модулів, було детально спроєктовано базу даних. На основі детального аналізу видів баз даних, було вибрано реляційну базу даних. Додатково було вибрано архітектурний шаблон для бази даних multi-tenancy, оскільки вебсистема створюється у вигляді SaaS і потрібно забезпечити ізоляцію даних. На основі цих даних, було створено ER-діаграму, яка відображає структуру бази даних.

Заключним кроком даного етапу роботи було вибір стеку технології для реалізації програмного забезпечення. На основі детального аналізу найбільш популярних технології вибрано Vue.js для фронтенду, PHP фреймворк Laravel для бекенду та PostgreSQL для бази даних.

Завершальним етапом розробки системи для автоматизації логістичних операцій було програмна реалізація. Спершу було реалізовано базу даних. Створено Eloquent моделі, міграції згідно ER діаграми, розробленої на етапі проєктування.

Перед реалізацією бекенду та фронтенду, було спроєктовано користувацький інтерфейс у програмному забезпечені Figma. На основі макетів, було створено відповідні компоненти та реалізовано серверну частину системи. У результаті було створено зручна та приваблива вебсистема для роботи логістів.

Для валідації програмного забезпечення було проведено різні види тестування. Всі тести показали успішні результати. Можна дійти висновку, що всі поставлені задачі для кваліфікаційної роботи було виконано. У кінці, було отримано працюючу вебсистему для автоматизації логістичних операцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 120 Automation Statistics: Business, Machine Learning and Challenges [Електронний ресурс] // Flair. – Режим доступу: https://flair.hr/en/blog/automation-statistics/

2. How Many Companies Use AI? (New 2025 Data) [Електронний ресурс] // Exploding Topics. – Режим доступу: https://explodingtopics.com/blog/companies-using-ai

3. Important Warehouse Automation Statistics You Can’t Ignore [Електронний ресурс] // Meteor Space. – Режим доступу: https://www.meteorspace.com/important-warehouse-automation-statistics/

4. Що таке логістика? [Електронний ресурс] // Ukrauto Logistic. – Режим доступу: https://ukrautologistic.com.ua/sho-take-logistika/

5. Що таке логістична компанія та як вона допомагає бізнесу? [Електронний ресурс] // Протокол. – Режим доступу: https://protocol.ua/ua/shcho_take_logistichna_kompaniya_ta_yak_vona_dopomagae_biznesu/

6. LBS Cloud [Електронний ресурс] // Веб-сайт LBS Cloud. – Режим доступу: https://lbs.systems/logistics

7. Система управління логістикою Qguar TMS [Електронний ресурс] // Веб-сайт Qguar TMS. – Режим доступу: https://quantum-int.com/products/tms-sistema-upravlinnya-transportom/

8. Логістична платформа Tocan TMS [Електронний ресурс] // Веб-сайт Tocan. – Режим доступу: https://tocan.com.ua/uk/sistema-upravleniya-transportom-tocan-logist-tms/

9. Основні типи архітектури програмного забезпечення [Електронний ресурс] // Art of business analysis. – Режим доступу: https://www.artofba.com/uk/post/main-types-of-software-architecture

10.Монолітна архітектура ПЗ [Електронний ресурс] // QALight. – Режим доступу: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/

11.Що таке архітектура мікросервісів? [Електронний ресурс] // Paypro Global. – Режим доступу: https://payproglobal.com/uk/відповіді/що-таке-мікросервісна-архітектура/

12.Клієнт-серверна архітектура [Електронний ресурс] // QATestLab training center. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

13.Що таке MVC? Переваги та недоліки MVC [Електронний ресурс] // AlexHost. – Режим доступу: https://alexhost.com/uk/faq/shho-take-mvc-perevagy-ta-nedoliky-mvc/

14.Діаграма діяльності в UML: символ, компоненти та приклад [Електронний ресурс] // Guru99. – Режим доступу: https://www.guru99.com/uk/uml-activity-diagram.html

15.Моделювання станів (State Modelling) [Електронний ресурс] // Maxym Zosym. – Режим доступу: https://www.maxzosim.com/state-modelling/

16.Нотація DFD [Електронний ресурс] // Електронна бібліотека «Луцький національний технічний університет». – Режим доступу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/Кондіус%202%20готовва/page18.html

17.Що таке діаграма розгортання? [Електронний ресурс] // Guru99. – Режим доступу: https://www.guru99.com/uk/deployment-diagram-uml-example.html

18.Типи баз даних: особливості, відмінності та приклади [Електронний ресурс] // Dou. – Режим доступу: https://dou.ua/lenta/articles/types-of-databases/

19.Стратегії мультитенантності баз даних для змішаних навантажень і SLA в реальному часі [Електронний ресурс] // ProIT. – Режим доступу: https://proit.ua/stratieghiyi-multitienantnosti-baz-danikh-dlia-zmishanikh-navantazhien-i-sla-v-riealnomu-chasi/

20.Поняття ER-моделі. Поняття сутності (entity). Атрибути. Види атрибутів [Електронний ресурс] // BestProg. – Режим доступу: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/

21.СУБД: які бувають, як вибрати [Електронний ресурс] // HighLoad. – Режим доступу: https://highload.tech/uk/subd-yaki-buvayut-yak-vibrati/

22.Основні переваги субд MySql [Електронний ресурс] // StudLife. – Режим доступу: https://studfile.net/preview/5607354/page:3/

23.Postgres поглинає світ баз даних [Електронний ресурс] // Data Life. – Режим доступу: https://data-life-ua.com/db/postgres-is-eating-the-database-world/

24.Oracle Database - An Introduction [Електронний ресурс] // Geeks for geeks. – Режим доступу: https://www.geeksforgeeks.org/oracle-database-an-introduction/

25.Які зараз перспективи у Node js? [Електронний ресурс] // FoxmindEd. – Режим доступу: https://foxminded.ua/node-js-perspektyvy/

26.8 ключових переваг фрейморку Laravel для веб-розробки [Електронний ресурс] // Wezom. – Режим доступу: https://wezom.com.ua/ua/blog/17-preimuschestv-ispolzovanija-laravel-v-it-industrii

27.Що розробляють на ASP.NET? [Електронний ресурс] // FoxmindEd. – Режим доступу: https://foxminded.ua/asp-net-tse/

28.Що таке React JS і для чого він потрібен? [Електронний ресурс] // Dan.IT Education. – Режим доступу: https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/

29.Vue.js як JavaScript-фреймворк [Електронний ресурс] // FoxmindEd. – Режим доступу: https://foxminded.ua/vue-js/

30.Що таке Angular? [Електронний ресурс] // Tutorial in UA. – Режим доступу: https://angular.tutorial.in.ua/what-is-angular/

31.A Practical Introduction to Laravel Eloquent ORM [Електронний ресурс] // DigitalOcean. – Режим доступу: https://www.digitalocean.com/community/tutorial-series/a-practical-introduction-to-laravel-eloquent-orm

32.What Are Migrations In Laravel? [Електронний ресурс] // Solid Brain. – Режим доступу: https://solidbrain.net/blog/what-are-migrations-in-laravel

33.Eloquent Relationship In Laravel [Електронний ресурс] // Dev. – Режим доступу: https://dev.to/karleb/eloquent-relationship-in-laravel-51jc

34.Understanding Laravel Blade and How To Use It [Електронний ресурс] // Kinsta. – Режим доступу: https://kinsta.com/blog/laravel-blade/

35.Laravel Form Request and Data Validation Tutorial [Електронний ресурс] // Inspector. – Режим доступу: https://inspector.dev/laravel-form-request-and-data-validation-tutorial/

36.Різниця між функціональним і нефункціональним тестуванням [Електронний ресурс] // QATestLab training center. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/difference-between-functional-and-non-functional-testing/

37.Що таке PhpUnit в PHP? [Електронний ресурс] // itProger. – Режим доступу: https://itproger.com/ua/spravka/php/phpunit

38.Методологія тестування коду на PHP з використанням PHPUnit та Pest [Електронний ресурс] // IT Рейтинг України. – Режим доступу: https://it-rating.ua/metodologiya-testuvannya-kodu-na-php-z-vikoristannyam-phpunit-ta-pest

39.Laravel Dusk: A Beginner's Guide to Automated Browser Testing [Електронний ресурс] // LaravelGeek. – Режим доступу: https://laravelgeek.com/posts/laravel-dusk-a-beginners-guide-to-automated-browser-testing

40.Тестова документація – що таке test case [Електронний ресурс] // IT-notes. – Режим доступу: https://www.it-notes.wiki/software-testing/what-is-test-case/

Додаток А

(Обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проєкту розробки MVP вебсистеми для автоматизації логістичних процесів транспортної компанії. Система покликана оптимізувати керування перевезеннями, контрагентами, персоналом та контрактами, забезпечуючи зручні інструменти для керівників та логістів. Рішення дозволяє створювати та обробляти заявки на перевезення, контролювати виконання завдань, переглядати аналітику, а також управляти користувачами та налаштуваннями системи.

1 Підстава для розробки

Підставою для розробки є «Завдання на кваліфікаційну роботу», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: Вебсистема для автоматизації логістичних операцій.

2 Призначення розробки

Вебсистема призначена для автоматизації основних логістичних процесів у транспортній компанії. Система забезпечує зручний та ефективний інструмент для управління перевезеннями, персоналом і контрактами. Вона дозволяє співробітникам різних ролей — логістам, керівникам відділів та директору компанії — ефективно виконувати свої обов’язки в єдиному інформаційному середовищі.

Функціональне призначення вебсистеми включає в себе наступні можливості для користувачів:

– додавання та опрацювання заявок на перевезення;

– призначення транспортних засобів та водіїв до маршрутів;

– відстеження статусу перевезень і взаємодія з водіями;

– ведення бази водіїв;

– перегляд ефективності маршрутів та історії перевезень;

– робота з контрактами;

– призначення завдань логістам і контроль їх виконання;

– аналіз ефективності перевезень за допомогою аналітичних звітів;

– управління базою товарів;

– призначення відповідальних логістів до виконання договорів;

– керування користувачами та їхніми ролями;

– перегляд фінансової аналітики та статистики роботи компанії;

– контроль платежів і заборгованостей;

– зміна системних параметрів (валюта, типи перевезень тощо);

– загальне управління роботою команди.

Користувачами системи є працівники логістичних компаній, які мають доступ до вебінтерфейсу системи через сучасні браузери.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Перелік функціональних можливостей, які має надавати вебсистема для автоматизації логістичних операцій:

1. Реєстрація та авторизація. Система повинна надавати можливість реєстрації нових користувачів та авторизації існуючих. Під час реєстрації користувач вказує ім’я, електронну пошту, пароль і підтвердження пароля. Після реєстрації доступ до системи здійснюється через авторизацію за допомогою вказаної електронної пошти та пароля.

2. Управління заявками на перевезення. Логісти мають можливість створювати заявки на перевезення, вносити до них деталі (маршрут, вантаж, дати

тощо), редагувати їх, змінювати статуси виконання, додавати коментарі та взаємодіяти з водіями в межах заявки.

3. Призначення ресурсів до маршрутів. Для кожної заявки логіст може призначати відповідний транспортний засіб і водія. В системі реалізовано перевірку доступності ресурсів на заданий період.

4. Відстеження статусу перевезень. Користувачі можуть відслідковувати поточний статус перевезень у режимі реального часу. Водії мають змогу оновлювати статус виконання безпосередньо в системі, що забезпечує оперативність та прозорість процесу.

5. Ведення бази водіїв. Система надає можливість створювати, редагувати та переглядати профілі водіїв, додавати контактні дані, документи, історію виконаних рейсів.

6. Управління контрактами. Здійснюється реєстрація та супровід контрактів із клієнтами та партнерами. Контракти можуть бути пов’язані з конкретними заявками чи маршрутами. Є можливість призначення відповідальних осіб, зберігання супровідних документів, контроль термінів виконання.

7. Аналітика та звітність. Керівники відділів і директори мають змогу переглядати звіти про ефективність логістичних операцій, історію перевезень, завантаженість транспорту, роботу логістів та водіїв. Дані виводяться у вигляді таблиць, графіків і діаграм.

8. Планування маршрутів. Система надає інструмент для побудови маршруту перевезення з урахуванням пунктів завантаження/розвантаження, відстані, часу в дорозі та наявних обмежень. Також можливе редагування вже створених маршрутів.

9. Керування користувачами та ролями. Директор компанії може створювати облікові записи співробітників, призначати їм ролі (логіст, керівник відділу, адміністратор), а також змінювати або блокувати доступ до системи.

3.2 Вимоги до надійності

Застосунок повинен виконувати наступні вимоги до надійності:

− розмежування ролів та прав користувачів під час користування системи;

− використання multitenancy для ізоляції даних користувачів між собою;

− валідація користувацьких даних на стороні клієнта і сервера;

− обробка помилок та надсилання повідомлень про це;

− блокування некоректних дій користувача при взаємодії з системою.

3.3 Умови експлуатації

Система призначена для використання в браузерах на персональних комп’ютерах, ноутбуках або планшетах з доступом до інтернету. Для стабільної роботи вебзастосунку необхідно використовувати сучасні вебоглядачі (Google Chrome, Firefox, Microsoft Edge тощо), оновлені до актуальних версій. Рекомендовані умови експлуатації повинні відповідати типовим офісним умовам — при температурі навколишнього середовища від +10 °C до +35 °C та відносній вологості не вище 80%. Система може функціонувати на мобільних пристроях через адаптивний вебінтерфейс, однак пріоритетною платформою залишається десктоп.

3.4 Вимоги до складу та параметрів технічних засобів

Система призначена для використання на пристроях з доступом до інтернету, зокрема на настільних комп’ютерах, ноутбуках і планшетах, які працюють під управлінням сучасних операційних систем (Windows, macOS, Linux). Рекомендується використовувати останні версії браузерів, таких як Google Chrome, Mozilla Firefox або Microsoft Edge, для забезпечення стабільної роботи інтерфейсу.

Для належного функціонування більшості можливостей системи необхідне стабільне з’єднання з Інтернетом. У майбутньому передбачено масштабування рішення з можливістю адаптації під мобільні пристрої через нативний

застосунок, однак наразі оптимальним є використання системи через браузер на десктопі. Рекомендовані технічні вимоги до пристрою:

– сучасна операційна система (Windows 10+, macOS 11+, Linux Kernel 5.0+);

– браузер: Google Chrome 100+, Firefox 100+, Microsoft Edge 100+;

– процесор: не менше 1.0 ГГц;

– оперативна пам’ять: від 2 ГБ;

– стабільне з’єднання з Інтернетом (мінімум 5 Мбіт/с).

3.5 Вимоги до інформаційної та програмної сумісності

Вебсистема для автоматизації логістичних процесів буде розроблена як SaaS-рішення, що реалізується на основі Laravel (бекенд) та Vue.js (фронтенд). Завдяки цій технологічній зв’язці система забезпечуватиме високу продуктивність, масштабованість і зручний інтерфейс для різних ролей користувачів: директорів, керівників відділів та логістів.

Система призначена для використання в браузерах на настільних ПК, ноутбуках та планшетах з доступом до Інтернету. У подальшому можливе створення мобільної версії або окремого мобільного додатку, але на поточному етапі стабільна робота гарантується лише у десктопному браузері.

Для зберігання та обробки даних буде використано PostgreSQL, з поділом доступу відповідно до ролей користувачів та реалізація multitenancy. Вебзастосунок забезпечить обробку великого обсягу логістичної інформації — включно з маршрутами, перевезеннями, водіями, контрактами, аналітикою тощо.

Усі дані передаватимуться за захищеним протоколом (HTTPS), з використанням токенів авторизації (наприклад, Laravel Sanctum або Passport) для забезпечення безпеки та контролю доступу.

3.6 Спеціальні вимоги

Вимоги до інтерфейсу:

– інтерфейс вебсистеми повинен бути інтуїтивно зрозумілим для всіх ролей користувачів: директорів, логістів та членів команди;

– дизайн має забезпечувати високу контрастність для зручності роботи у виробничих та офісних умовах;

– інтерфейс повинен містити статичну бічну навігаційну панель (sidebar) з іконками та назвами основних розділів системи: «Транспорти», «Контракти», «Документи», «Аналітика» тощо;

– вміст кожного розділу має чітко відображати поточний рівень вкладеності (глибину навігації)**,** наприклад, шляхом breadcrumbs або заголовків сторінок.

4 Вимоги до програмної документації

Після завершення розробки MVP вебсистеми для автоматизації логістичних процесів замовнику буде передано наступний комплект документації:

– програмний код системи з коментарями, що пояснюють логіку основних модулів (Laravel backend, React frontend);

– опис функціональних можливостей розробленої системи, включаючи розмежування доступу за ролями (директор, логіст, адміністратор), опис модулів транспорту, документів, аналітики, маршрутів тощо;

– технічне завдання на розробку (поточний документ), що містить вимоги до системи, її архітектури, функцій і не функціональних параметрів.

5 Стадії та етапи розробки

Стадії та етапи розробки проєкту «Вебсистеми для автоматизації логістичних операцій» наведено в таблиці А.1.

Таблиця А.1 — Стадії та етапи розробки проєкту

Стадія розробки	Період виконання	Етапи робіт	Зміст робіт
Формування вимог	02.01.2025 – 20.02.2025	Аналіз і постановка задачі	Аналіз проблематики логістичних компаній; обґрунтування доцільності створення MVP-системи; формування вимог до функціоналу, ролей, даних і доступу.
		Розробка технічного завдання	Складання технічного завдання на розробку вебсистеми: визначення цілей, призначення, архітектури, інтерфейсу, вимог до безпеки, надійності та масштабованості.
Проєктування	21.02.2025 – 20.03.2025	Ескізне проєктування	Попереднє проектування модулів системи (реєстрація, транспорти, документи, аналітика); формування загальної структури даних; вибір технологій.
	21.03.2025 – 10.04.2025	Технічне проєктування	Деталізація архітектури додатку; розробка ER-діаграм, маршрутизації, REST API; визначення алгоритмів логіки обробки транспортів, користувачів, статусів.
Реалізація	11.04.2025 – 30.04.2025	Розробка MVP	Створення базової версії програми (реєстрація, ролі, управління транспортами); розробка бази даних, авторизації, основних CRUD-функцій; внутрішнє тестування.
Тестування	23.04.2025 – 30.04.2025	Верифікація та валідація ПЗ	Проведення функціонального тестування; перевірка на відповідність вимогам; усунення помилок; оптимізація продуктивності.

Продовження таблиці А.1

Документування	01.05.2025 – 10.05.2025	Розробка супровідної документації	Підготовка керівництва користувача, інструкцій для логістів та директорів, технічного опису API та архітектури системи.
Впровадження	11.05.2025 – 25.05.2025	Розгортання MVP-системи	Розгортання ПЗ на сервері або хмарному середовищі; передача системи замовнику; консультації щодо використання; підготовка до подальшого масштабування.

6 Порядок контролю та приймання

Основні види тестування:

– інтеграційне тестування: передача даних між компонентами, збереження і обробка інформації в базі даних, обробка авторизації та ролей користувачів;

– end-to-End тестування (E2E): реєстрацію та логін користувача (директора, логіста); створення, редагування, перегляд транспорту; зміну статусу замовлення; перевірку доступів для кожної ролі.

Вимоги до прийняття:

– весь функціонал повинен реалізовано згідно з описаними вимогами: управління транспортами, ролями, документами, маршрутами тощо;

– вебсистема повинна стабільно працювати у різних середовищах і пристроях, коректно обробляти помилки, забезпечує збереження даних у випадках відмови мережі;

– усі основні дії (авторизація, створення об'єктів, редагування, фільтрація, генерація документів) повинні успішно бути протестовані перед здачею;

– до системи мають бути підготовлені супровідні документи: технічне завдання, опис функціоналу, інструкція користувача.

Додаток Б

(Обов’язковий)

ЛІСТИНГ КОДУ

0001_01_01_000000_create_users_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('tenant_id')->nullable();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });

        Schema::create('password_reset_tokens', function (Blueprint $table) {
            $table->string('email')->primary();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });

        Schema::create('sessions', function (Blueprint $table) {
            $table->string('id')->primary();
            $table->foreignId('user_id')->nullable()->index();
            $table->string('ip_address', 45)->nullable();
            $table->text('user_agent')->nullable();
            $table->longText('payload');
            $table->integer('last_activity')->index();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('users');
        Schema::dropIfExists('password_reset_tokens');
        Schema::dropIfExists('sessions');
    }
};
```

2019_09_15_000010_create_tenants_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateTenantsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up(): void
    {
        Schema::create('tenants', function (Blueprint $table) {
            $table->string('id')->primary();

            $table->string('name')->nullable();

            $table->timestamps();
            $table->json('data')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down(): void
    {
        Schema::dropIfExists('tenants');
    }
}
```

2025_01_01_123913_create_user_roles_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('user_roles', function (Blueprint $table) {
            $table->id();
            $table->string("slug", 255)->unique();
            $table->string("name", 255);
        });

        Schema::table('users', function (Blueprint $table) {
            $table->foreignId("role_id")->nullable();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
```

```
        Schema::dropIfExists('user_roles');
    }
};
```

2025_01_02_150321_create_customers_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('customers', function (Blueprint $table) {
            $table->id();
            $table->string('tenant_id');
            $table->string('first_name', 255);
            $table->string('last_name', 255);
            $table->string('company_name', 255);
            $table->string('country', 255);
            $table->string('region', 255);
            $table->timestamps();

            $table->foreign("tenant_id")->references('id')->on('tenants')-
>onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('customers');
    }
};
```

2025_01_02_153043_create_contracts_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('contracts', function (Blueprint $table) {
            $table->id();
            $table->string('tenant_id');
            $table->foreignId('customer_id');
            $table->dateTime("deadline");
            $table->double('total')->default(0);
            $table->timestamps();
```

```
            $table->foreign("tenant_id")->references('id')->on('tenants')-
>onDelete('cascade');
            $table->foreign("customer_id")->references('id')->on('customers')-
>onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('contracts');
    }
};
```

2025_01_12_190156_create_products_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string("name", 255);
            $table->string("measurement_unit", 255);
            $table->timestamps();
        });

        Schema::create('contract_products', function (Blueprint $table) {
            $table->id();
            $table->string('tenant_id');
            $table->foreignId('contract_id');
            $table->foreignId('product_id');
            $table->decimal("quantity");
            $table->timestamps();

            $table->foreign("tenant_id")->references('id')->on('tenants')-
>onDelete('cascade');
            $table->foreign("contract_id")->references('id')->on('contracts')-
>onDelete('cascade');
            $table->foreign("product_id")->references('id')->on('products')-
>onDelete('cascade');

        });

    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('products');
        Schema::dropIfExists('contract_products');
    }
```

```
};
```

2025_04_13_192110_create_locations_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('locations', function (Blueprint $table) {
            $table->id();

            $table->string('tenant_id');
            $table->string('address_1');
            $table->string('address_2');
            $table->string('city');
            $table->string('state');
            $table->string('country');

            $table->foreign("tenant_id")->references('id')->on('tenants')-
>onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('locations');
    }
};
```

2025_04_13_194059_create_drivers_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('drivers_companies', function (Blueprint $table) {
            $table->id();
            $table->string('company_name');
            $table->string('phone');
            $table->string('email');
            $table->string('address_1');
            $table->string('address_2');
            $table->string('city');
            $table->string('state');
```

```
            $table->string('country');
            $table->timestamps();
        });

        Schema::create('cars', function (Blueprint $table) {
            $table->id();
            $table->foreignId('drivers_company_id');
            $table->string('car_model');
            $table->string('car_number');
            $table->float('max_load_capacity')->default(0);
            $table->timestamps();

            $table->foreign("drivers_company_id")->references('id')-
>on('drivers_companies')->onDelete('cascade');
        });

        Schema::create('drivers', function (Blueprint $table) {
            $table->id();
            $table->string('tenant_id');
            $table->foreignId('drivers_company_id')->after("id");
            $table->string('first_name');
            $table->string('last_name');
            $table->string('phone')->nullable();
            $table->string('email')->nullable();
            $table->string('city');
            $table->string('state');
            $table->string('country');
            $table->timestamps();

            $table->foreign("tenant_id")->references('id')->on('tenants')-
>onDelete('cascade');
            $table->foreign("drivers_company_id")->references('id')-
>on('drivers_companies')->onDelete('cascade');

        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('drivers_companies');
        Schema::dropIfExists('cars');
        Schema::dropIfExists('drivers');
    }
};
```

2025_04_13_194059_create_transportartions_table.php

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('transportartions', function (Blueprint $table) {
            $table->id();
```

```
            $table->string('tenant_id');
            $table->foreignId('contract_id');
            $table->foreignId('logistician_id');
            $table->foreignId('starting_location_id');
            $table->foreignId('loading_location_id');
            $table->foreignId('unloading_location_id');
            $table->foreignId('driver_id');
            $table->foreignId('car_id');
            $table->foreignId('product_id');
            $table->float('amount_loaded_product');
            $table->timestamps();

            $table->foreign("tenant_id")->references('id')->on('tenants')-
>onDelete('cascade');
            $table->foreign("logistician_id")->references('id')->on('users')-
>onDelete('cascade');
            $table->foreign("starting_location_id")->references('id')-
>on('locations')->onDelete('cascade');
            $table->foreign("loading_location_id")->references('id')-
>on('locations')->onDelete('cascade');
            $table->foreign("unloading_location_id")->references('id')-
>on('locations')->onDelete('cascade');
            $table->foreign("driver_id")->references('id')->on('drivers')-
>onDelete('cascade');
            $table->foreign("product_id")->references('id')->on('products')-
>onDelete('cascade');
            $table->foreign("car_id")->references('id')->on('cars')-
>onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('transportartions');
    }
};
```

User.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Passport\HasApiTokens;

class User extends Authenticatable
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
    protected $fillable = [
        'name',
```

```
        'email',
        'password',
        'tenant_id',
        'role_id'
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var list<string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }

    public function role(): BelongsTo {
        return $this->belongsTo(UserRole::class, "role_id");
    }
}
```

UserRole.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class UserRole extends Model
{
    protected $fillable = [
        'slug',
        'name'
    ];
}
```

Customer.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class Customer extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'first_name',
        'last_name',
```

```
        'company_name',
        'region',
        'country'
    ];
}
```

DriversCompany.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class DriversCompany extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'company_name',
        'phone',
        'email',
        'address_1',
        'address_2',
        'postcode',
        'city',
        'state',
        'country'
    ];

    protected $with = [
        'cars'
    ];

    public function cars(): HasMany
    {
        return $this->hasMany(Car::class, "drivers_company_id");
    }
}
```

Driver.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class Driver extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'drivers_company_id',
        'first_name',
        'last_name',
        'phone',
        'email',
        'city',
        'state',
```

```
        'country'
    ];

    protected $with = [
        'company'
    ];

    public function company(): BelongsTo
    {
        return $this->belongsTo(DriversCompany::class, "drivers_company_id");
    }
}
```

Product.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    protected $fillable = [
        'name',
        'measurement_unit'
    ];

    protected $hidden = [
        'created_at',
        'updated_at'
    ];
}
```

Location.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class Location extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'first_name',
        'last_name',
        'company_name',
        'phone',
        'email',
        'address_1',
        'address_2',
        'city',
        'state',
        'country'
    ];
}
```

Car.php

```
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class Car extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'drivers_company_id',
        'car_model',
        'car_number',
        'max_load_capacity'
    ];

    public function company(): BelongsTo
    {
        return $this->belongsTo(DriversCompany::class, "drivers_company_id");
    }
}
```

Contract.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class Contract extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'customer_id',
        'deadline',
        'total'
    ];

    protected $with = [
        'customer',
        'products'
    ];

    public function customer() : BelongsTo
    {
        return $this->belongsTo(Customer::class);
    }

    public function products() : HasMany
    {
        return $this->hasMany(ContractProduct::class, "contract_id");
    }

    public function transportartions(): HasMany
    {
        return $this->hasMany(Transportartion::class, "contract_id");
    }
}
```

ContractProduct.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class ContractProduct extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'contract_id',
        'product_id',
        'quantity'
    ];

    protected $hidden = [
        'id',
        'tenant_id',
        'contract_id',
        'product_id',
        'created_at',
        'updated_at'
    ];

    protected $with = [
        'product'
    ];

    public function product(): BelongsTo
    {
        return $this->belongsTo(Product::class);
    }
}
```

Transportation.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class Transportation extends Model
{
    use BelongsToTenant;

    protected $fillable = [
        'tenant_id',
        'contract_id',
        'logistician_id',
        'starting_location_id',
        'loading_location_id',
        'unloading_location_id',
        'driver_id',
    ];
```

```
    public function logistician() : BelongsTo
    {
        return $this->belongsTo(User::class, "logistician_id");
    }

    public function starting_location() : BelongsTo
    {
        return $this->belongsTo(Location::class, "starting_location_id");
    }

    public function loading_location() : BelongsTo
    {
        return $this->belongsTo(Location::class, "loading_location_id");
    }

    public function unloading_location() : BelongsTo
    {
        return $this->belongsTo(Location::class, "unloading_location_id");
    }

    public function driver() : BelongsTo
    {
        return $this->belongsTo(Driver::class, "driver_id");
    }

    public function products() : HasMany
    {
        return $this->hasMany(TransportartionProduct::class,
"transportartion_id");
    }
}
```

TransportationProduct.php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Stancl\Tenancy\Database\Concerns\BelongsToTenant;

class TransportationProduct extends Model
{
    use BelongsToTenant;

    protected $hidden = [
        'transportartion_id',
        'contract_product_id',
        'quantity'
    ];
}
```

CustomerController.php

```
use App\Http\Requests\StoreCustomerRequest;
use App\Http\Requests\UpdateCustomerRequest;
use App\Models\Customer;
use Inertia\Inertia;
use Illuminate\Http\Request;

class CustomerController extends Controller
{
```

```
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $customers = Customer::all();

        return Inertia::render('Customers/Index', [
            'customers' => $customers,
        ]);
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        return Inertia::render('Customers/Create');
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(StoreCustomerRequest $request)
    {
        $validated = $request->validated();

        $customer = Customer::create($validated);

        return redirect()->route('customers.index')->with('success', 'Клієнт
створений успішно.');
    }

    /**
     * Display the specified resource.
     */
    public function show(Customer $customer)
    {
        return Inertia::render('Customers/Show', [
            'customer' => $customer,
        ]);
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit(Customer $customer)
    {
        return Inertia::render('Customers/Edit', [
            'customer' => $customer,
        ]);
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(UpdateCustomerRequest $request, Customer $customer)
    {
        $validated = $request->validated();

        $customer->update($validated);
```

```
        return redirect()->route('customers.index')->with('success', 'Клієнт
оновлений успішно.');
    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy(Customer $customer)
    {
        $customer->delete();

        return redirect()->route('customers.index')->with('success', 'Клієнт
видалений успішно.');
    }
}
```

DriverController.php

```
use App\Http\Requests\StoreDriverRequest;
use App\Http\Requests\UpdateDriverRequest;
use App\Models\Driver;
use App\Models\DriversCompany;
use Inertia\Inertia;

class DriverController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $drivers = Driver::with('driversCompany')->get(); // завантажити
компанію одразу

        return Inertia::render('Drivers/Index', [
            'drivers' => $drivers,
        ]);
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        return Inertia::render('Drivers/Create', [
            'drivers_companies' => DriversCompany::all(),
        ]);
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(StoreDriverRequest $request)
    {
        $validated = $request->validated();

        $driverCompanyData = $validated['drivers_company'];

        $driverCompany = !empty($driverCompanyData['id'])
            ? DriversCompany::find($driverCompanyData['id'])
            : DriversCompany::create($driverCompanyData);
```

```
        $driverData = $validated['driver'];
        $driverData['drivers_company_id'] = $driverCompany->id;

        Driver::create($driverData);

        return redirect()->route('drivers.index')->with('success', 'Водій
успішно створений.');
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit(Driver $driver)
    {
        return Inertia::render('Drivers/Edit', [
            'driver' => $driver->load('driversCompany'),
            'drivers_companies' => DriversCompany::all(),
        ]);
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(UpdateDriverRequest $request, Driver $driver)
    {
        $validated = $request->validated();

        $driverCompanyData = $validated['drivers_company'];

        $driverCompany = !empty($driverCompanyData['id'])
            ? DriversCompany::find($driverCompanyData['id'])
            : DriversCompany::create($driverCompanyData);

        $driverData = $validated['driver'];
        $driverData['drivers_company_id'] = $driverCompany->id;

        $driver->update($driverData);

        return redirect()->route('drivers.index')->with('success', 'Інформацію
про водія оновлено.');
    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy(Driver $driver)
    {
        $driver->delete();

        return redirect()->route('drivers.index')->with('success', 'Водія
видалено.');
    }
}
```

ContractController.php

```
use App\Http\Requests\StoreContractRequest;
use App\Http\Requests\UpdateContractRequest;
use App\Models\Contract;
use App\Models\ContractProduct;
use App\Models\Customer;
use App\Models\Product;
```

```
use Illuminate\Support\Facades\DB;
use Inertia\Inertia;

class ContractController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $contracts = Contract::with('customer')->get();

        return Inertia::render('Contracts/Index', [
            'contracts' => $contracts,
        ]);
    }

    /**
     * Show the specified resource.
     */
    public function show(Contract $contract)
    {
        $contract->load(['customer', 'products']);

        return Inertia::render('Contracts/Show', [
            'contract' => $contract,
        ]);
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        return Inertia::render('Contracts/Create', [
            'customers' => Customer::all(),
            'products' => Product::all()
        ]);
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(StoreContractRequest $request)
    {
        $validated = $request->validated();

        DB::transaction(function () use ($validated) {
            $contract = Contract::create($validated);

            foreach ($validated['products'] as $productData) {
                $productData['contract_id'] = $contract->id;
                ContractProduct::create($productData);
            }
        });

        return redirect()->route('contracts.index')->with('success', 'Контракт
успішно створено.');
    }

    /**
     * Show the form for editing the specified resource.
     */
```

```
    public function edit(Contract $contract)
    {
        $contract->load('products');

        return Inertia::render('Contracts/Edit', [
            'contract' => $contract,
            'customers' => Customer::all(),
            'products' => Product::all(),
        ]);
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(UpdateContractRequest $request, Contract $contract)
    {
        $validated = $request->validated();

        DB::transaction(function () use ($contract, $validated) {
            $contract->update($validated);

            // Оновлюємо продукти контракту
            $contract->products()->delete(); // Спочатку видаляємо старі

            foreach ($validated['products'] as $productData) {
                $productData['contract_id'] = $contract->id;
                ContractProduct::create($productData);
            }
        });

        return redirect()->route('contracts.index')->with('success', 'Контракт
оновлено.');
    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy(Contract $contract)
    {
        $contract->delete();

        return redirect()->route('contracts.index')->with('success', 'Контракт
видалено.');
    }
}
```

TransportationController.php

```
class TransportationController extends Controller
{
    public function index()
    {
        $transportations = Transportartion::with(['driver', 'car', 'products'])-
>get();

        return Inertia::render('Transportations/Index', [
            'transportations' => $transportations,
        ]);
    }

    public function show(Transportartion $transportation)
    {
```

```
        $transportation->load(['driver', 'car', 'startingLocation',
'loadingLocation', 'unloadingLocation', 'products']);

        return Inertia::render('Transportations/Show', [
            'transportation' => $transportation,
        ]);
    }

    public function create(Contract $contract)
    {
        return Inertia::render('Contracts/Transportation/Create', [
            'contract' => $contract,
            'drivers' => Driver::with('company.cars')->get(),
        ]);
    }

    public function store(StoreTransportartionRequest $request, Contract
$contract)
    {
        $validated = $request->validated();

        DB::beginTransaction();

        try {
            $transportion_data = [
                'logistician_id' => auth()->id() ?? 1,
                'contract_id' => $contract->id,
                'driver_id' => $validated['driver'],
            ];

            $transportion_data['starting_location_id'] =
Location::create($validated['starting_location'])->id;
            $transportion_data['loading_location_id'] =
Location::create($validated['loading_location'])->id;
            $transportion_data['unloading_location_id'] =
Location::create($validated['unloading_location'])->id;

            // Обробка машини
            $car_data = $validated['car'];

            if (empty($car_data['id'])) {
                // Припускаємо, що водій має компанію з машинами
                $driver = Driver::with('company.cars')-
>findOrFail($validated['driver']);
                $car = $driver->company->cars()->create($car_data);
                $transportion_data['car_id'] = $car->id;
            } else {
                $transportion_data['car_id'] = $car_data['id'];
            }

            $transportation = Transportartion::create($transportion_data);

            // Продукти
            foreach ($validated['products'] as $p) {
                $transportation->products()->attach($p['contract_product_id'], [
                    "quantity" => $p['quantity']
                ]);
            }

            DB::commit();

            return redirect()->route('contracts.show', $contract->id)-
>with('success', 'Transportation created.');
```

```
        } catch (\Throwable $e) {
            DB::rollBack();
            return back()->withErrors(['error' => 'Transportation creating is
failed']);
        }
    }

    public function edit(Transportartion $transportation)
    {
        $transportation->load(['driver', 'car', 'startingLocation',
'loadingLocation', 'unloadingLocation', 'products']);

        return Inertia::render('Transportations/Edit', [
            'transportation' => $transportation,
            'drivers' => Driver::with('company.cars')->get(),
        ]);
    }

    public function update(UpdateTransportartionRequest $request,
Transportartion $transportation)
    {
        $validated = $request->validated();

        DB::transaction(function () use ($validated, $transportation) {
            $transportation->update([
                'driver_id' => $validated['driver'],
                'car_id' => $validated['car']['id'] ?? null,
            ]);

            // Оновлюємо локації
            $transportation->startingLocation-
>update($validated['starting_location']);
            $transportation->loadingLocation-
>update($validated['loading_location']);
            $transportation->unloadingLocation-
>update($validated['unloading_location']);

            // Оновлюємо продукти
            $transportation->products()->detach();

            foreach ($validated['products'] as $p) {
                $transportation->products()->attach($p['contract_product_id'], [
                    "quantity" => $p['quantity']
                ]);
            }
        });

        return redirect()->route('transportations.show', $transportation->id)-
>with('success', 'Transportation updated.');
    }

    public function destroy(Transportartion $transportation)
    {
        $transportation->delete();

        return redirect()->route('transportations.index')->with('success',
'Transportation is delted.');
    }

    /**
     * Generate documents for the transportation (e.g. CMR, накладна).
     */
    public function generateDocuments(Transportartion $transportation)
```

```
    {
        $transportation->load([
        'driver',
        'car',
        'contract.customer',
        'products.contractProduct.product',
        'startingLocation',
        'loadingLocation',
        'unloadingLocation'
    ]);
    $pdf = Pdf::loadView('pdf.transportation', [
        'transportation' => $transportation
    ]);

    $filename = 'transportation_' . $transportation->id . '.pdf';

    return $pdf->download($filename);
    }
}
```

TransportationController.php

```
class RegisterUserController extends Controller
{
    /**
     * Display the registration view.
     */
    public function create(): Response
    {
        return Inertia::render('Auth/Register');
    }

    /**
     * Handle an incoming registration request.
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    public function store(Request $request): RedirectResponse
    {
        $request->validate([
            'name' => 'required|string|max:255',
            'email' =>
'required|string|lowercase|email|max:255|unique:'.User::class,
            'password' => ['required', 'confirmed', Rules\Password::defaults()],
        ]);

        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);

        Log::info("new userr");

        event(new Registered($user));

        Auth::login($user);

        return redirect()->route('setup');
    }
}
```

Додаток В

(Обов’язковий)

ПРЕЗЕНТАЦІЯ

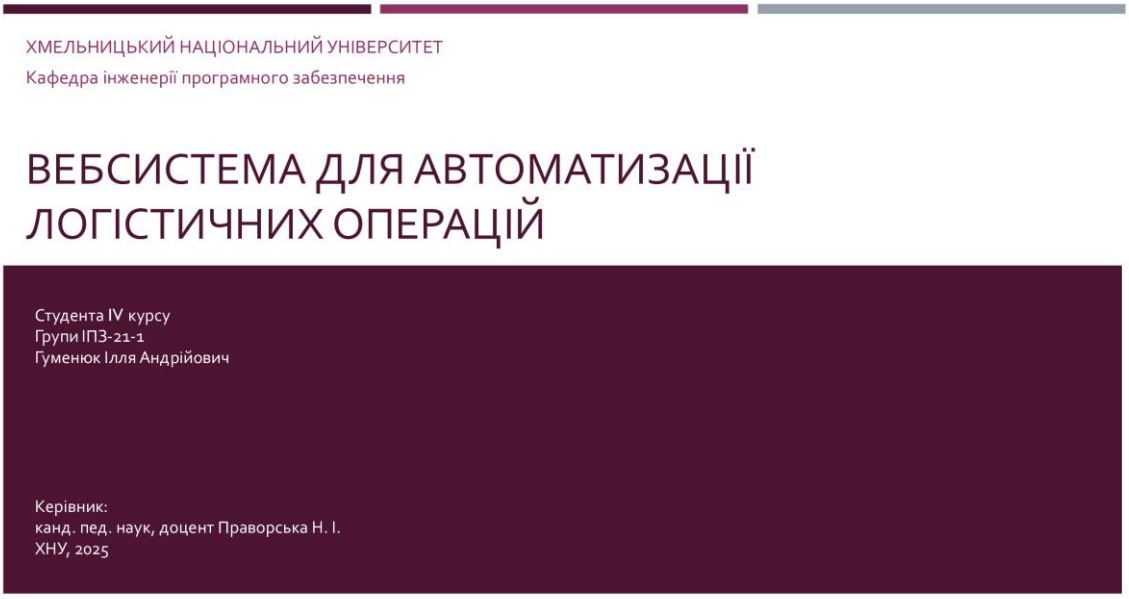


Рисунок В.1 — Слайд №1

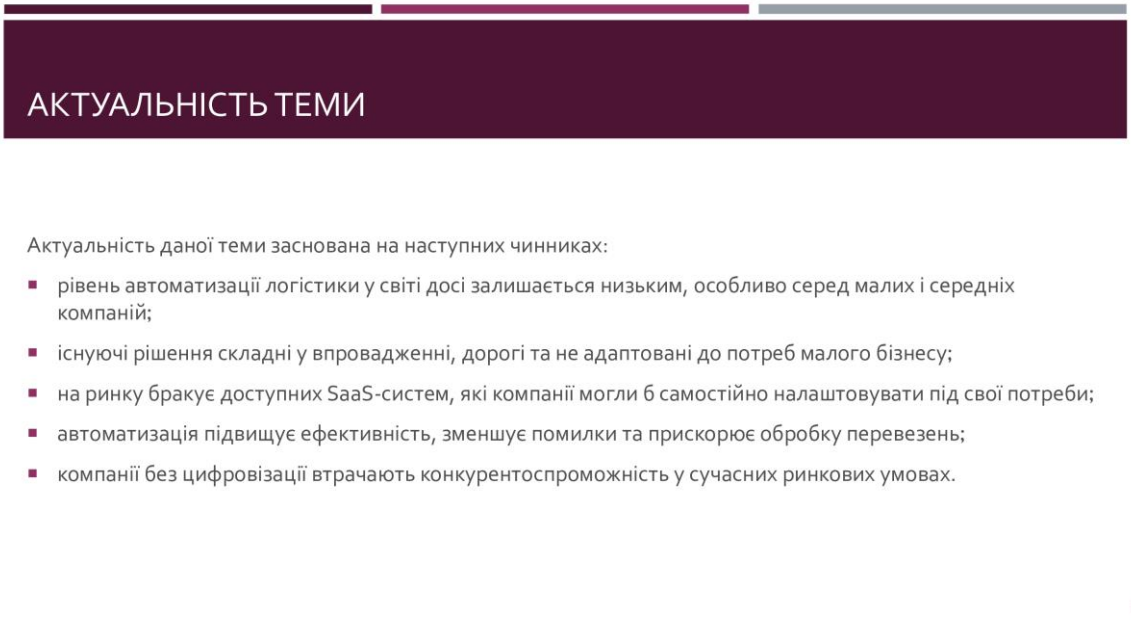


Рисунок В.2 — Слайд №2

МЕТА ТА ЗАВДАННЯ

Метою даної кваліфікаційної роботи є розробка MVP системи керування транспортом для логістичної компанія, яка дозволятиме логістам, керівникам та іншим співробітникам компанії відстежувати перебіг контрактів та перевезень, аналітику, єдину базу водії та клієнтів. Для досягнути поставленої мети, потрібно:

- дослідити детально предметну області;
- проаналізувати існуючі програмно-технічного забезпечення;
- визначити функціональні та нефункціональні вимоги;
- детально спроектувати базу даних, за допомогою ER-діаграми, та систему, використовуючи UML діаграми;
- обрати потрібний стек технологій;
- успішно програмно реалізувати систему;
- виконати тестування системи на правильність виконання та відповідність функціональних вимог.

3

Рисунок В.3 — Слайд №3

ЗМІСТОВИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ЇЇ СТРУКТУРНИХ ТА ФУНКЦІОНАЛЬНИХ ОСОБЛИВОСТЕЙ

Логістика – це процес планування, організації та управління переміщенням товарів, послуг або інформації від місця їх виробництва до кінцевого споживача.

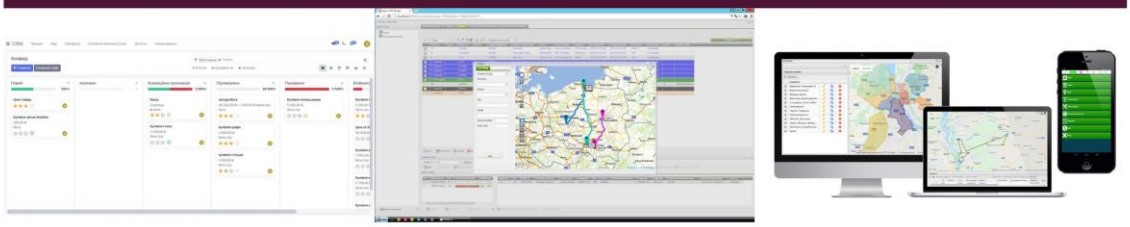
Основні функції, які виконує логістика:

- організація руху товарів між різними пунктами;
- забезпечення належних умов для продукції до її відправлення споживачам;
- регулювання кількості товарів для покриття попиту без зайвих витрат;
- управління процесом виконання замовлень, включно з поверненнями;
- використання даних для аналізу та підвищення ефективності логістики

4

Рисунок В.4 — Слайд №4

АНАЛІЗ НАЯВНОГО ПРОГРАМНО-ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ



LBS Cloud	Qguar TMS	Tocan TMS
Перевагою даного рішення є те, що вона розроблена у вигляді SaaS. Іншою важливою перевагою даної системи є інтеграція з багатьма сторонніми сервісами.	Основна перевага системи — високий рівень довіри, підтверджений позитивними відгуками та співпрацею з такими брендами, як Яготинське, Royal Canin, Епіцентр.	Розробник пропонує два варіанти: SaaS або повне придбання, що підходить як малим, так і великим компаніям. Система підтримує інтеграції та складається з взаємопов'язаних модулів, забезпечуючи повний контроль над процесами.
Система не орієнтована на логістику й може не покривати потреби логістичних компаній, зокрема відстеження перевезень, складу та інших функцій.	Система не є SaaS-рішенням, тому вимагає залучення розробників для встановлення й налаштування, що збільшує вартість і робить її недоступною для деяких компаній.	Недоліком є застарілий та доволі не зручний дизайн. При перегляді інтерфейс виникає «страх», що все дуже важко.

5

Рисунок В.5 — Слайд №5

ВИЗНАЧЕННЯ ФУНКЦІОНАЛЬНИХ ТА НЕФУНКЦІОНАЛЬНИХ ВИМОГ ДО ПЗ

Основні функції системи за ролями:

- **Директор:** керування користувачами та доступами, перегляд фінансової та операційної аналітики, налаштування системи, контроль команди та загальної ефективності.
- **Керівник відділу:** моніторинг ефективності перевезень, розподіл завдань серед логістів, керування товарами та контрактами.
- **Загальні можливості:** створення та обробка заявок на перевезення, призначення транспорту й водіїв, відстеження перевезень у реальному часі, робота з маршрутами, управління базами водіїв і контрактів.

Серед нефункціональних вимог варто виділити:

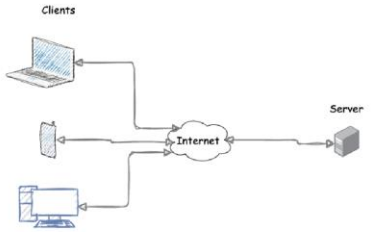
- система має бути швидко загружатися, а саме не більше 3 секунд;
- обробка одночасних запитів без втрати продуктивності;
- система має підтримувати збільшення кількості користувачів та перевезень без впливу на продуктивність;
- однакова робота на всіх девайсах та на всіх операційних системах;
- зручний, привабливий та інтуїтивно зрозумілий інтерфейс.

6

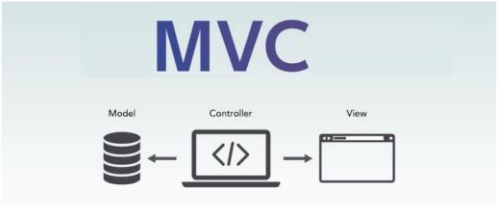
Рисунок В.6 — Слайд №6

ВИБІР ТИПУ АРХІТЕКТУРИ ТА ШАБЛОНІВ ПРОЄКТУВАННЯ

- Більшість веб-застосунків використовують клієнт-серверну архітектуру, через простоту реалізації і доступність більшості клієнтів, що забезпечує доступність для всіх, якщо користувач використовує не застарілі браузери. Саме через ці причини буде використовуватися клієнт-серверна архітектура



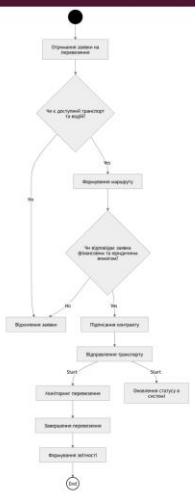
- Архітектурним шаблон для системи використовувалося. MVC (Model-View-Controller) – це архітектурний шаблон програмного забезпечення, який зазвичай використовується для розробки користувацьких інтерфейсів

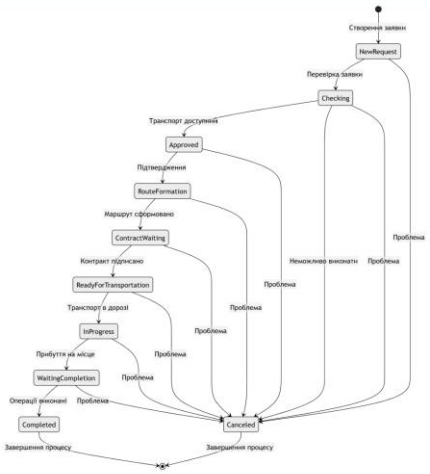


7

Рисунок В.7 — Слайд №7

ОПИС ДЕКОМПОЗИЦІЇ, ЗАЛЕЖНОСТЕЙ, ІНТЕРФЕЙСІВ

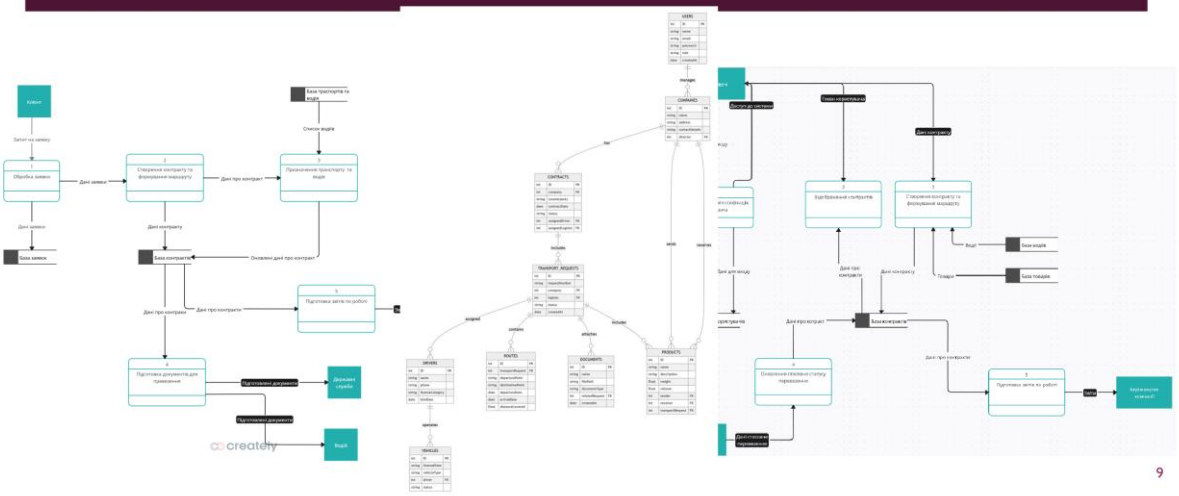




8

Рисунок В.8 — Слайд №8

ПРОЄКТУВАННЯ МОДУЛІВ І ДАНИХ



9

Рисунок В.9 — Слайд №9

АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ

PostgreSQL

1. Висока масштабованість — підтримка великих обсягів даних і великої кількості записів.
2. Гнучкість — можливість розширення за допомогою користувацьких типів, індексів, операторів і мов програмування.

Vue.js

1. Простий у використанні, але водночас продуктивний і функціональний.
2. Швидке розгортання на сервері.
3. Багато вбудованих функцій «з коробки» (на відміну від React, де потрібні сторонні бібліотеки).
4. Менша залежність від зовнішніх бібліотек, що спрощує розробку та підтримку структури проєкту.

1. Широкий набір вбудованих функцій для швидкої та зручної розробки.
2. Гнучка структура, шаблонізатор Blade, зручна маршрутизація.
3. Активна спільнота та регулярні оновлення.
4. Висока масштабованість і безпека.

10

Рисунок В.10 — Слайд №10

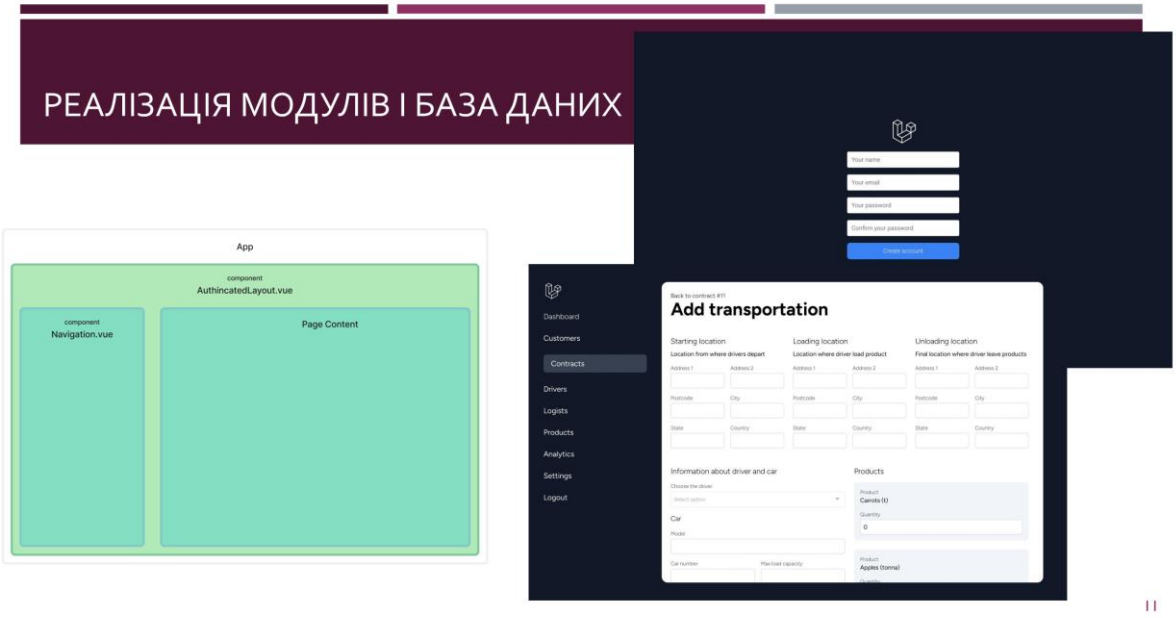


Рисунок В.11 — Слайд №11

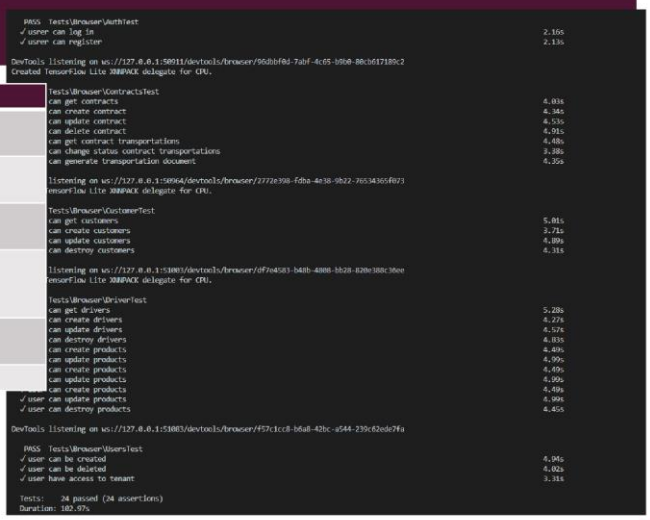
Категорія	Компонент	Вимоги
Сервер	CPU	4 ядра+ (Intel Xeon / AMD Ryzen)
	RAM	8–16 ГБ
	Диск	SSD, від 100 ГБ
	ОС	Ubuntu Server 20.04+
Клієнт	Браузер	Chrome, Firefox, Edge (останні версії)
	Екран	Від 1280×720
ПЗ (сервер)	Мова	PHP 8.1+
	Фреймворк	Laravel 10
	БД	PostgreSQL 13+
	Web-сервер	Nginx або Apache
	Пакет-менеджер	Composer
ПЗ (клієнт)	Фреймворк	Vue.js 3
	Збірка	Vite або Webpack
Інше	Контроль версій	Git
	IDE	VS Code або PhpStorm
	Розгортання	VPS або локально (опційно Docker)

Рисунок В.12 — Слайд №12

ТЕСТУВАННЯ ПЗ

Модуль	Перевірені дії	Успішність
Аутентифікація	Реєстрація, вхід з помилкою	✓✓ Passed
Multitenancy	Ізоляція даних, перевірка доступів	✓✓ Passed
Контракти	Перегляд, створення, редагування, PDF	✓✓ Passed
Клієнти	CRUD-операції, навігація між сторінками	✓✓ Passed
Товари	CRUD-операції, модальне додавання	✓✓ Passed
Водії	CRUD-операції	✓✓ Passed

Всього було протестовано 6 модулів та виконано 24 інтеграційних та end-to-end тестів



13

Рисунок В.13 — Слайд №13

ВИСНОВКИ

Завдання	Виконано
Дослідити предметну область	Проведено аналіз логістики; визначено автоматизовані процеси; вивчено вимоги користувачів
Проаналізувати існуюче програмно-технічне забезпечення	Порівняно 3 системи управління транспортом; виявлено їх сильні та слабкі сторони
Визначити функціональні та нефункціональні вимоги	Сформовано перелік вимог; створено діаграми варіантів використання
Спроєктувати базу даних та систему за допомогою ER- та UML-діаграм	Створено ER-діаграму; розроблено діаграми діяльності, станів, потоків даних, розгортання
Обрати стек технологій	Вибрано Vue.js (frontend), Laravel (backend), PostgreSQL (БД), архітектура multi-tenancy
Програмно реалізувати систему	Реалізовано базу даних, бекенд і фронтенд на основі проєктних рішень та інтерфейсів з Figma
Провести тестування системи	Проведено функціональні тести (24 тест); всі результати — **Passed**

14

Рисунок В.14 — Слайд №14

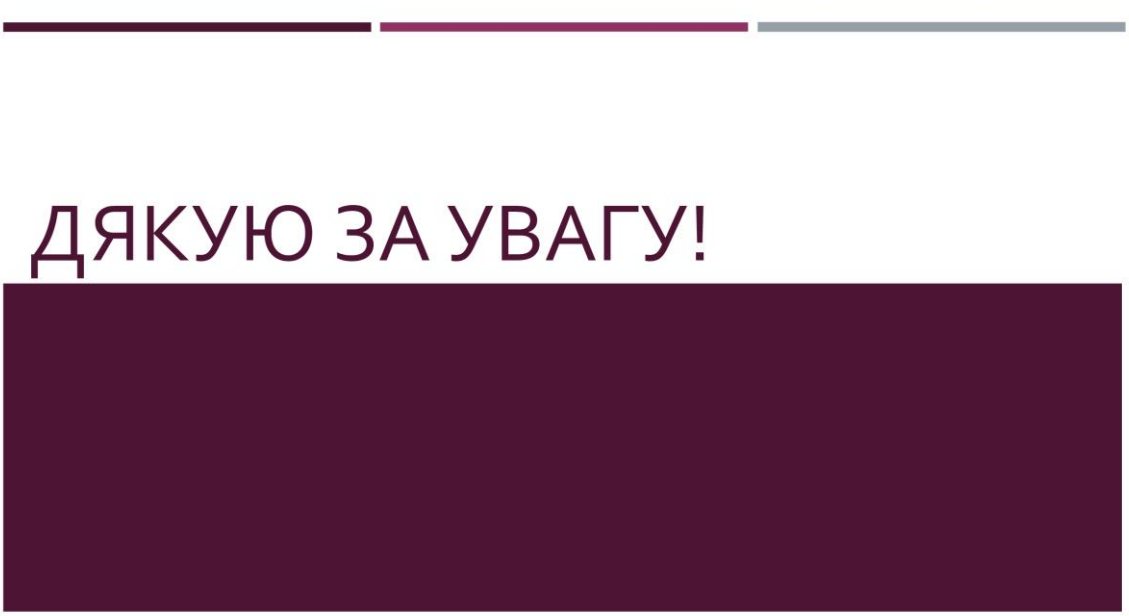
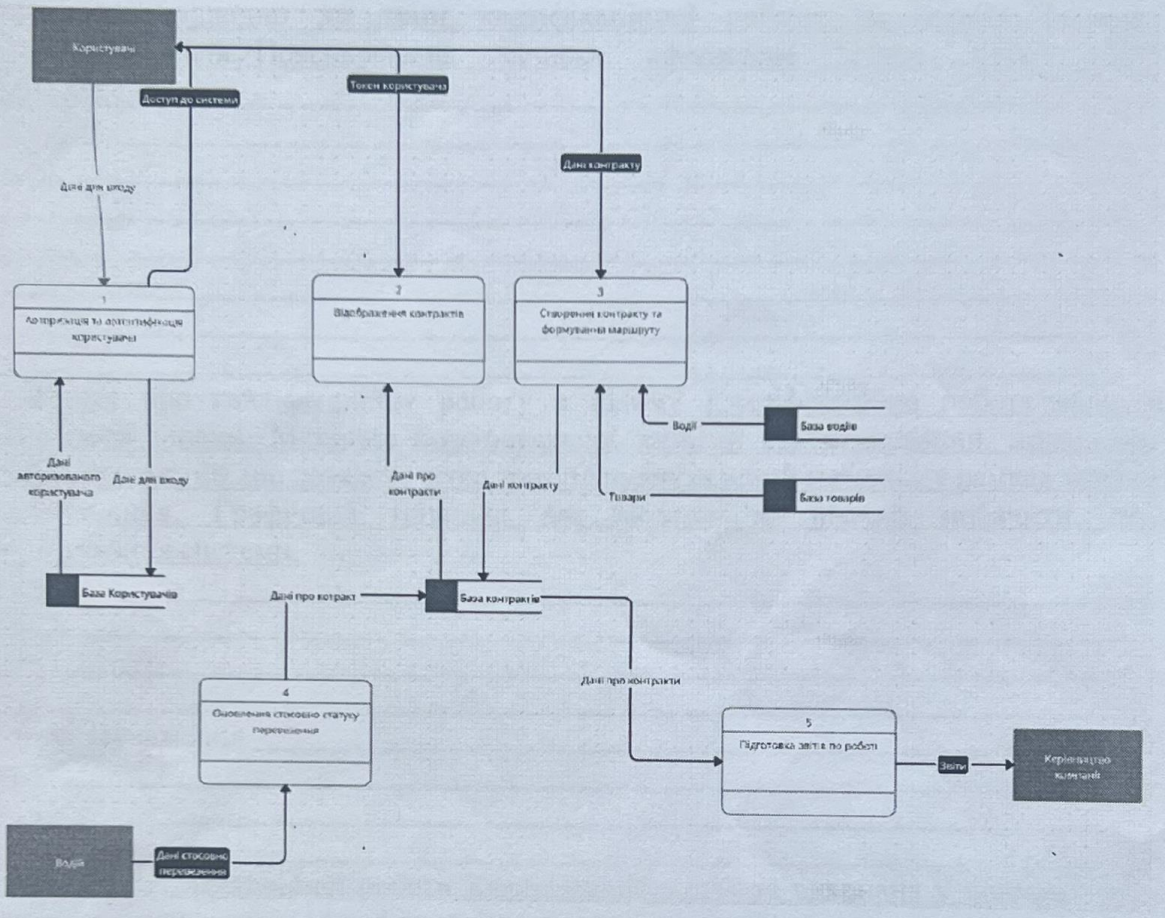
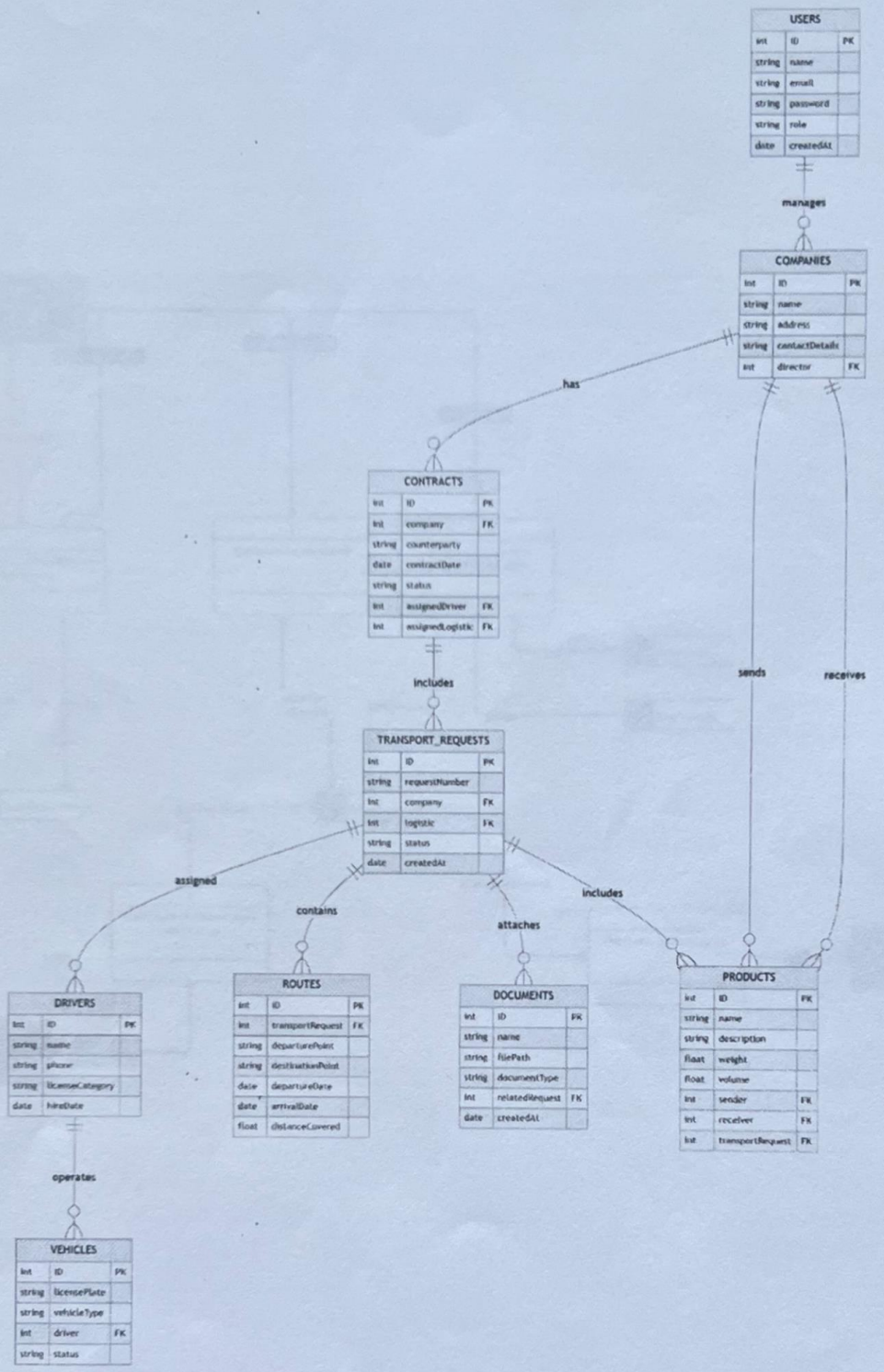


Рисунок В.15 — Слайд №15

ГРАФІЧНІ МАТЕРІАЛИ



					КвРІПЗ.2101073.01.05.Е8			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема для автоматизації логістичних операцій DFD-діаграма	Літ.	Маса.	Масштаб
Розробив		Гуменюк І. А.		02.06				
Керівник		Праворська Н.І.		02.06		Аркуш 1	Аркушів 2	
Рецензент		Кацепатенко М.В.		02.06				
Н. Контр.		Бедратюк Г.І.		02.06		ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.		02.06				



					КвРІПЗ.2101073.01.05.Е8	
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема для автоматизації логістичних операцій	Літ. / Маса. / Масштаб
Розробив		Гуменюк І. А.		02.06		
Керівник		Праворська Н.І.		02.06	ER-діаграма	Аркуш 2 / Аркушів 2
Н. Контр.		Бедратюк Г.І.		02.06		ХНУ, ІПЗ-21-1
Зав. каф.		Бедратюк Л. П.		02.06		

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Гуменюка Іллі Андрійовича
факультет ІТ, IV курс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.06.2025
дата

підпис

Anti-Plagiarism v-15.274 Educational

The maximum coincidence with one document 3.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 12%**

	Document		Sum coincidence on the DB	
ID: 241888 Title: ДП_Вебсистема для автоматизації логістичних операцій Added in a DB: 2025-05-26 Authors: І.А. Гуменюк Heads: канд. пед. наук, доцент _ Н.І. Праворська . Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	79515	1255	5890 (7%)	89 (7%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Гуменюк Ілля

Співавтор:

Назва: БКР_Вебсистема для автоматизації логістичних операцій

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:14.2%

Коефіцієнт подібності 2:7.1%

Мікропробіли: 0

Заміна букв: 1

Інтервали: 0

Білі знаки: 2

Дата створення звіту: 2025-05-28 19:47:31.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата експерт

05.28.2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Гуменюк Ілля Андрійович

Тема Вебсистема для автоматизації логістичних операцій

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 2 кількість сторінок записки 76

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі розглянуто створення вебсистеми для автоматизації логістичних операцій. Проведено аналіз предметної області, проаналізовано існуюче програмно-технічне забезпечення сформульовано функціональні та нефункціональні вимоги, обґрунтовано вибір архітектурних та технологічних рішень. Реалізовано функціональну вебсистему із можливістю введення бази клієнтів, автопарку та водії, відстеження контрактів та аналіз ефективність компанії. Проведено тестування, яке підтвердило коректну роботу програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи. У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт і предмет дослідження. У першому розділі проведено аналіз предметної області, розглянуто наявні системи керування транспортом, виявлено їхні переваги та недоліки, сформульовано функціональні та нефункціональні вимоги до застосунку. На основі вимог було створено технічне завдання. У другому розділі спроєктовано архітектуру системи на основі MVC, побудовано UML діаграми, ER-діаграму, DFD-діаграми. Проведено вибір технічного стеку. У третьому розділі спроєктовано інтерфейс вебсистеми з урахуванням UI/UX вимог, реалізовано функціонал вебсистеми за допомогою Laravel, Vue.js та PostgreSQL: облік клієнтів, автопарку та водіїв, облік контрактів, відслідковування контрактів, генерація потрібної документації. Проведено інтеграційне та End-to-end тестування. Результатом роботи стала сучасна, зручна і масштабована вебсистема у вигляді SaaS, що відповідає поставленим вимогам і має потенціал для подальшого розвитку.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки є потреба у зручному цифровому рішенні для автоматизації логістичних операцій та перебігу логістичних поставок. Робота вирізняється всебічним аналізом аналогів, продуманим архітектурним проєктуванням, використанням сучасних технологій (Laravel, Vue.js, PostgreSQL), а також орієнтацією на зручність і потреби кінцевого користувача

5. Негативні сторони роботи Вебсистема не реалізує розширену взаємодію між логістами компанії, які стежать за перебігом транспортування, та водієм.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ К.ю.н., доцент кафедри КІІС, Кашуєвська Марія Вікторівна

"02" червня 2025 р. _____
(підпис)

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

першого освітнього рівня «Бакалавр»

Студента: Гуменюк Ігор Андрійович
Група: ІПЗ-21-1

Тема: *«Вебсистема для автоматизації логістичних операцій»*

Спеціальність: 121 – Інженерія програмного забезпечення

Короткий зміст пояснювальної записки

У вступі кваліфікаційної роботи окреслюється актуальність автоматизації бізнес-процесів, зокрема в сфері логістики, де автоматизація може значно підвищити ефективність роботи компаній. Метою роботи є розробка MVP системи керування транспортом (TMS) для логістичної компанії, яка дозволить відстежувати перебіг контрактів та перевезень. Завданнями роботи є дослідження предметної області, аналіз існуючих рішень, визначення функціональних та нефункціональних вимог, проектування бази даних, вибір технологій та тестування системи.

Відповідність отриманих результатів роботи поставленим завданням

Завдання, сформульовані у вступі, включають дослідження предметної області, аналіз існуючих рішень, визначення вимог, проектування бази даних, вибір технологій та тестування системи. Висновки підтверджують, що результати роботи в основному відповідають поставленим завданням. Оцінка відповідності: в основному відповідають.

Оцінка розділів

Розділ 1: Аналіз предметної області

У цьому розділі розглядається предметна область логістики, її функції та важливість автоматизації. Описано основні функції логістики, що дозволяє зрозуміти її роль у бізнес-процесах. Проте, опис предметної області є загальним і потребує більшої глибини аналізу, зокрема конкретних прикладів та статистичних даних. Рекомендується додати детальніший опис, включаючи реальні кейси та специфічні проблеми, з якими стикаються логістичні компанії.

Розділ 2: Проєктування програмного забезпечення

У цьому розділі проведено аналіз архітектури системи, вибір шаблонів проєктування, опис модулів та їх взаємодії. Однак, опис архітектури потребує уточнення щодо критеріїв вибору, а також більшої деталізації в описі модулів та їх функцій. Рекомендується додати схеми та приклади, щоб краще проілюструвати структуру системи. Загалом, розділ містить важливі аспекти, але потребує уточнень та доповнень.

Розділ 3: Програмна реалізація та тестування

Цей розділ містить детальний опис реалізації бази даних, інтерфейсу та тестування. Опис реалізації є логічним і структурованим, але варто додати більше прикладів коду та візуалізацій результатів тестування. Висновки чітко формулюють готовність продукту до впровадження, але могли б бути більш конкретними щодо досягнутих цілей.

Позитивні сторони

Кваліфікаційна робота демонструє оригінальність у підході до автоматизації логістичних процесів, а також якість рішень, що пропонуються. Розроблений продукт має потенціал стати корисним SaaS рішенням для логістичних компаній. Описані технології реалізації є сучасними та відповідають вимогам ринку.

Недоліки

Основними недоліками є недостатня глибина аналізу предметної області, відсутність конкретних прикладів у розділі про проєктування, а також недостатня деталізація системних вимог та результатів тестування. Рекомендується переглянути структуру розділів для покращення логічності та зрозумілості викладення.

Відгук в цілому

Кваліфікаційна робота є актуальною та має практичну значущість, оскільки автоматизація в логістиці є важливим напрямком для підвищення ефективності бізнес-процесів. Зміст роботи відповідає темі та завданням, а новизна ідей та рішень є очевидною. Однак, для досягнення високої якості реалізації, необхідно внести уточнення та доповнення в окремі розділи.

Оцінка кваліфікаційної роботи

Кваліфікаційна робота заслуговує оцінки "добре". Вона виконана в повному обсязі з дотриманням основних вимог, але має деякі недоліки, які потребують доопрацювання.

Рекомендації

Рекомендується доопрацювати роботу, зокрема уточнити опис предметної області, додати більше прикладів у розділі про проєктування, а також візуалізації результатів тестування. Це дозволить підвищити якість роботи та її відповідність сучасним вимогам.

OpenAI API-асистент
Session ID: 99283e16-4489-4e40-8431-466a8b35b8e2
Підписано автоматично, модель gpt-4o-mini
Дата: 26.05.2025

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Вебсистема для автоматизації логістичних операцій
Автор Гуменюк Ілля Андрійович
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»
Рівень вищої освіти Перший (бакалаврський)
Спеціальність 121 «Інженерія програмного забезпечення»
Науковий керівник: Праворська Наталія Іванівна, канд. пед. наук, доцент
На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	**відповідає**
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, завдання, анотація, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій та у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 14,2%, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 02.06.2025

Завідувач кафедри _____ Підпис Леонід БЕДРАТЮК Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис Леонід БЕДРАТЮК Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис Наталія ПРАВОРСЬКА Ім'я, ПРІЗВИЩЕ