НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
(назва кафедри)
_____ Голуб Б. Л.
(підпис) (ПІБ)
"__" червня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

"Програмне забезпечення інформаційної системи оренди транспортних засобів"

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми

к.н.т., доцент _____ Вайганг Г.О.
(наукова ступінь та вчене звання) (підпис) (ПІБ)

Керівник Бакалаврської кваліфікаційної роботи

к.ф.-м.н., доцент _____ Кириченко В. В.
(наукова ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Трохименко Дмитро Володимирович
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
(назва кафедри)
_____ Голуб Б. Л.
(підпис) (ПІБ)
"16" грудня 2024 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Трохименко Дмитро Володимирович
(прізвище, ім'я, по батькові)
Спеціальність F2 – «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи Програмне забезпечення інформаційної системи оренди транспортних засобів

Затверджена наказом ректора НУБіП України від "16" грудня 2024 р. № 2249 "С"

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)
Вихідні дані до бакалаврської кваліфікаційної роботи
Опис предмету дослідження, опис програмного забезпечення
Перелік питань, які потрібно розробити:
Системний аналіз предметної області
Аналіз предметної області
Розробка програмного забезпечення
Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання "16" січня 2025 р.

Керівник Бакалаврської кваліфікаційної роботи
к.ф.-м.н., доцент _____ Кириченко В. В.
(наукова ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання
_____ Трохименко Дмитро Володимирович
(підпис) (ПІБ студента)

ЗМІСТ

ВСТУП

У сучасному світі стрімкого розвитку технологій та зростання мобільності, потреба в ефективних, зручних і надійних транспортних рішеннях стає все більш актуальною. Зі збільшенням кількості населення та транспортних засобів на дорогах, традиційні методи оренди автомобілів вже не завжди відповідають вимогам користувачів щодо швидкості, зручності та безпеки. Саме тому виникає необхідність у створенні сучасного веб-застосунку для оренди транспортних засобів.

Метою дипломного проєкту є розробка веб-застосунку, що забезпечить автоматизований, швидкий та зручний процес орендування автомобілів. Такий застосунок дасть змогу користувачам легко знаходити та бронювати транспортні засоби, здійснювати оплату, отримувати необхідну інформацію та керувати орендою онлайн. Це дозволить компаніям-орендодавцям оптимізувати свої бізнес-процеси, зменшити витрати часу та підвищити прибутковість, а користувачам — отримати якісні послуги з максимальним комфортом.

Під час реалізації проєкту планується використання сучасних технологій веб-розробки та баз даних для створення надійної, масштабованої та безпечної інформаційної системи. Система дозволятиме автоматизувати ключові етапи оренди, зокрема бронювання, облік транспорту, оплату та аналітику, враховуючи при цьому вимоги користувачів до функціональності, ефективності та простоти використання.

Кваліфікаційна робота бакалавра на тему «Програмне забезпечення інформаційної системи оренди транспортних засобів» включає розрахунково-

пояснювальну записку, яка виконана на 78 сторінках аркушах, включає в себе 7 таблиць та 25 рисунків. Розрахунково-пояснювальна записка складається із вступу, 6 розділів, висновків, переліку посилань з 11 найменувань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

У сучасному світі, який переживає постійні соціальні, економічні та технологічні трансформації, питання мобільності населення набуває особливої актуальності. Швидкий темп життя, урбанізація, зростання населення та інтенсивне використання особистого транспорту створюють додаткове навантаження на інфраструктуру, погіршують екологічну ситуацію та підвищують витрати часу на пересування. У цих умовах сервіси оренди транспортних засобів виступають ефективним і гнучким рішенням.

Особливої актуальності ця проблема набула у зв’язку з пандемією COVID-19, яка суттєво змінила підходи до користування транспортом. Люди почали уникати громадського транспорту через ризик зараження, що спричинило зростання попиту на індивідуальні транспортні засоби. Водночас обмеження контактів і карантинні заходи стимулювали перехід більшості послуг в онлайн-середовище. Саме під час пандемії стало очевидно, наскільки важливо мати можливість отримувати необхідні сервіси дистанційно, швидко та безпечно.

Традиційна модель оренди автомобілів із відвідуванням офісу, оформленням паперових документів і фізичною взаємодією з персоналом втратила актуальність у нових реаліях. Це викликало необхідність створення цифрових платформ, які дозволяють повністю автоматизувати процес — від

вибору автомобіля до його бронювання, оплати й управління замовленням через особистий кабінет.

Веб-застосунки, що забезпечують ці функції, не лише відповідають очікуванням сучасного користувача, а й сприяють підвищенню ефективності бізнесу: дозволяють краще керувати автопарком, оптимізувати логістику, зменшити витрати на персонал і підвищити рівень задоволеності клієнтів. Крім того, завдяки збиранню та аналізу даних, компанії можуть краще розуміти поведінку користувачів і ухвалювати обґрунтовані стратегічні рішення.

Таким чином, у контексті глобальних змін, викликаних як технологічним прогресом, так і пандемією, створення веб-застосунку для оренди транспортних засобів є вкрай актуальним і необхідним. Це рішення дозволяє не лише адаптуватися до нових умов, а й забезпечити високу якість сервісу, зручність для користувачів та конкурентоспроможність бізнесу на ринку.

1.2 Аналіз продуктів комерційних компаній по оренді автомобілів (потенційних конкурентів)

Аналіз ринку програмного забезпечення є важливим етапом при розробці та впровадженні інформаційної системи. Мета аналізу полягає в ідентифікації потенційних аналогів та конкурентів пропонованої системи оренди ТЗ, оцінці їхніх функціональних можливостей, переваг і недоліків. Ця інформація є важливою для визначення стратегії розробки та підвищення конкурентоздатності пропонованого рішення.

У рамках даного аналізу будуть розглянуті:

- Локальні програмні продукти, пропоновані українськими компаніями.
- Функціональні можливості: можливості з точки зору бронювання, оренди, управління транспортними засобами, обслуговування клієнтів, звітності тощо.
- Відповідність сучасним вимогам: відповідність стандартам, нормам та найкращим практикам у галузі оренди ТЗ.
- Можливості інтеграції: можливості інтеграції з іншими інформаційними системами, такими як бухгалтерські програми, CRM-системи тощо.

Отримані результати допоможуть:

- Визначити найбільш перспективні напрямки розвитку та удосконалення пропонованої інформаційної системи.
- Підвищити конкурентоздатність на ринку.
- Запропонувати клієнтам системи з оптимальним набором функцій та можливостей.

Аналіз ринку є важливим інструментом, який може допомогти розробити та впровадити успішну систему.

За основу пошуку інформації в інтернеті було виділено 3 основних інформаційних та українських систем оренди транспортних засобів:

SEVEN CARS – Прокатна компанія створена в 2013 році двома амбіційними людьми. З часом компанія розросталася і перетворилася на одного з лідерів столиці у своєму сегменті ринку. Сьогодні автопарк seven cars складається понад 100 автомобілів від «Економ» до «Преміум» класу. В

цій компанії можна знайти великий вибір іномарок, які хороші не лише своїми технічними характеристиками, але й чудовим станом Сайт компанії: 7cars.com.ua/ (Рис. 1.1).

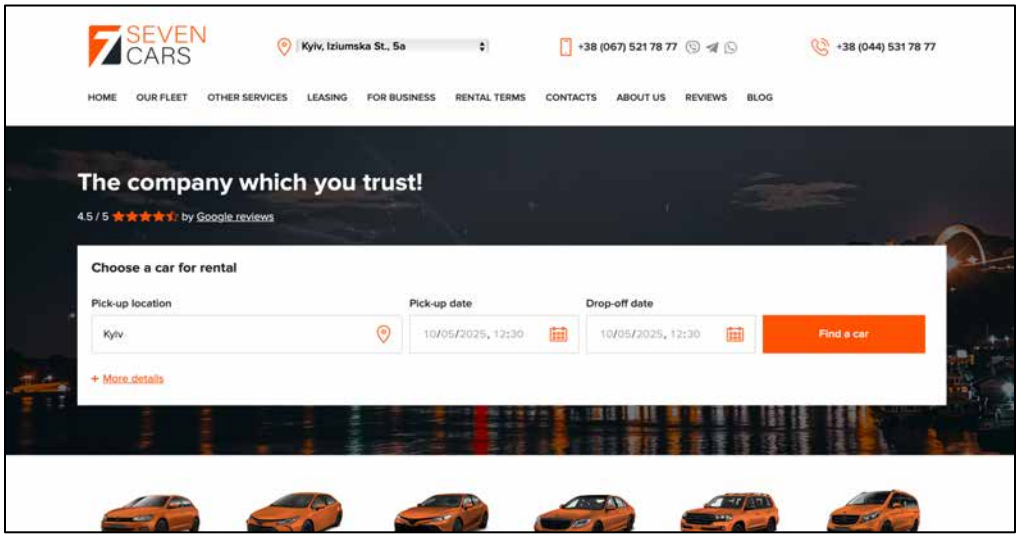

Рис. 1.1 – Головна сторінка SEVEN CARS

Основні функціональні можливості сервісу:

- Пошук та бронювання авто;
- Перегляд доступних авто та їх характеристик;
- Порівняння тарифів;
- Оплата онлайн;
- Управління бронюванням;
- Перегляд історії оренди;
- Зв'язок з підтримкою;

Додаткові функціональні можливості:

- Персоналізація профілю;

- Зберігання улюблених транспортів;
- Відстеження статусу бронювання;
- Доступ до спеціальних пропозицій;
- Можливість залишити відгук;

Недоліки:

- Складність інтерфейсу: деяким користувачам може здатися складним інтерфейс веб-додатку, через велику кількість функцій та меню.
- Відсутність онлайн-чату: Підтримка користувачів здійснюється лише через телефонний зв'язок або електронну пошту, що може бути незручно.
- Незручна фільтрація авто: фільтри для пошуку авто не завжди зручні та логічні, що може ускладнювати пошук потрібного авто.

RENTAL – компанія, яка надає послуги з прокату авто у всіх найбільших містах України. Локації з оренди авто знаходяться у Києві, Харкові, Дніпрі, Львові, Одесі, Вінниці, Запоріжжі та Івано-Франківську. Оренда авто у даній компанії - це величезний вибір автомобілів, починаючи від економ класу, на зразок Chevrolet Spark, закінчуючи преміальними моделями седанів типу Mercedes S класу та позашляховиків - Audi Q7 і Toyota Land Cruiser 200, якими можна керувати самостійно без водія. Сайт компанії: rental.ua. (Рис. 1.2).

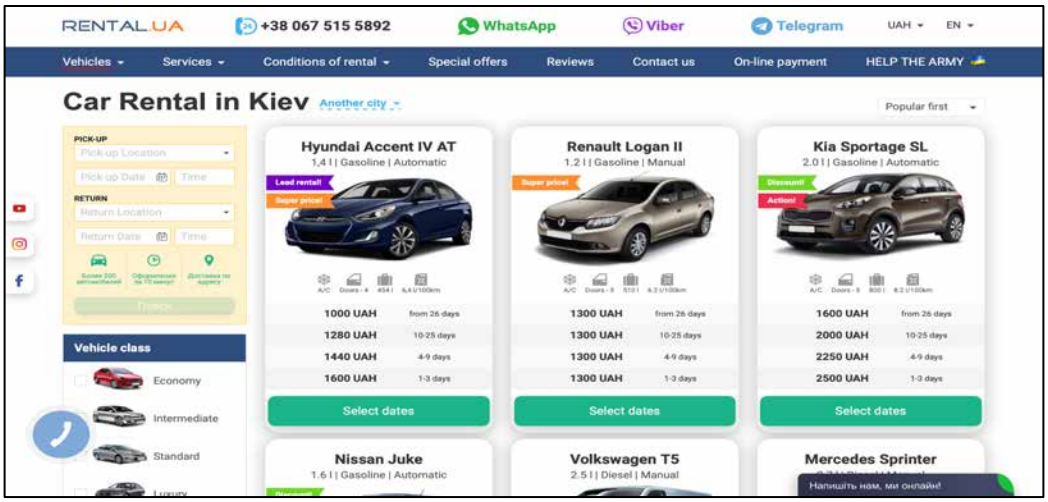

Рис. 1.2 – Головна сторінка RENTAL

Основні функціональні можливості:

- Пошук та бронювання авто;
- Перегляд доступних авто та їхні характеристики;
- Оплата онлайн;
- Управління бронюванням;
- Перегляд історії оренди;
- Зв'язок з підтримкою;

Додаткові функціональні можливості:

- Самообслуговування (відкриття/закриття);
- Можливість орендувати велосипед;
- Карта з пунктами самообслуговування;
- Доступ до FAQ;

Недоліки:

- Обмежена функціональність: веб-додаток RENTAL пропонує менший спектр функцій порівняно з SEVEN CARS, наприклад, відсутня можливість відстеження статусу бронювання.
- Немає можливості зберегти улюблені авто: це може бути незручно, якщо ви часто орендуєте авто однієї моделі.
- Не завжди коректне відображення доступних авто: на веб-сайті можуть відображатися недоступні авто, що може призвести до плутанини.

NarsCars - це українська компанія, що динамічно розвивається, відповідаючи трендам і крокуючи в тандемі з часом, розвивається на вітчизняному і міжнародному ринках. На сьогодні NarsCars, входить в топ три найбільших українських компаній по оренді авто. Має більше 170 автомобілів, філії компанії розташовані по всій Україні в найбільших містах. Сайт: narscars.com.ua (Рис. 1.3).

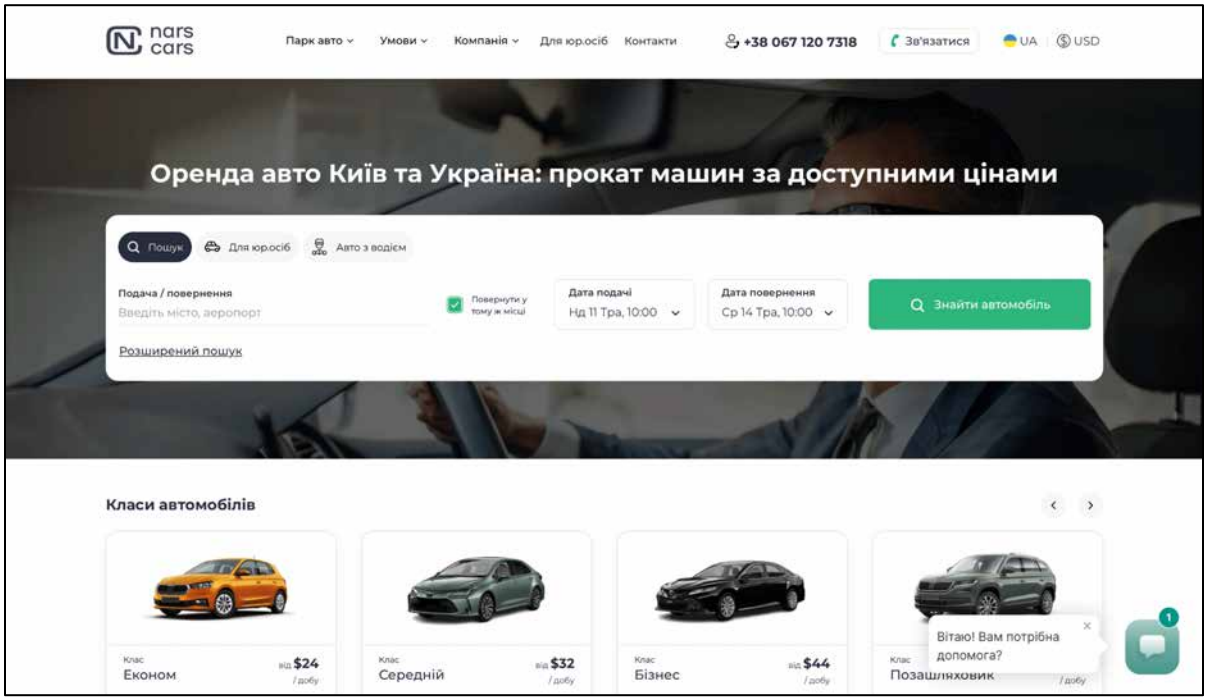

Рис. 1.3 – Головна сторінка NarsCars

Основні функціональні можливості:

- Пошук та бронювання;
- Перегляд доступних (з детальними фото);
- Вибір персонального менеджера;
- Оплата онлайн;
- Перегляд історії оренди;
- Цілодобова підтримка VIP-клієнтів;

Додаткові функціональні можливості:

- Бронювання готелів та ресторанів;
- Доступ до ексклюзивних пропозицій;
- Можливість залишити відгук;

Недоліки:

- Відсутність онлайн-бронювання: бронювання авто на веб-сайті NarsCars неможливо, для цього потрібно зв'язатися з менеджером.
- Немає детальної інформації про авто: на веб-сайті NarsCars представлена лише коротка інформація про авто, без фото та детальних характеристик.

Після збору необхідної інформації для даних компаній та порівняння програм-аналогів було сформовано таку таблицю:

Таблиця 1.1 - Порівняльна характеристика обраних компаній

	SEVEN CARS	RENTAL	NarsCars
Власний кабінет (реєстрація, авторизація)	+	+	+
Оформлення замовлення	+	+	+
Історія оренди	+	+	+
Онлайн підтримка	+	+	+
Калькулятор оренди	+	+	+
Самообслуговування	-	+	-
Доставка транспорту	-	-	+
VIP-сервіс	-	-	+
Система відгуків	+	+	+
Онлайн оплата	+	+	+
Персоналізація	-	+	-
Мобільний додаток	+	+	-

1.3 Постановка завдання

Метою дипломного проєкту є створення веб-застосунку для підбору та оренди автомобілів, який буде зручним у користуванні, інтуїтивно зрозумілим і не перевантаженим зайвою інформацією.

Після аналізу вже наявних рішень на ринку було прийнято рішення реалізувати веб-застосунок за моделлю електронної комерції, з урахуванням

сучасних вимог до UX/UI дизайну та функціональності. Такий підхід дозволить зробити взаємодію між користувачем і системою максимально ефективною.

Користувач зможе ознайомитися з усім переліком доступних тз, переглянути детальні описи, технічні характеристики та фотографії, після чого — здійснити вибір і заповнити форму для бронювання. Після надсилання заявки вона автоматично надходитиме в адмін-панель із відповідним статусом "в очікуванні".

Менеджер матиме змогу підтвердити, відхилити або видалити заявку, а також змінити її дані при необхідності. Крім того, він матиме повний контроль над сайтом і базою даних: додавання, редагування і видалення ТЗ, оновлення інформації, додавання типів двигунів, КПП, марок, моделей, категорій тощо. При створенні нового авто адміністратор вказує доступну кількість машин для оренди та встановлює вартість. Система підтримує можливість прикріплення медіа контенту для кожного ТЗ.

Також в адмін-панелі доступна статистика щодо загальної кількості авто, заброньованих машин і тих, що очікують на видачу. Користувач несе відповідальність за правильність введених даних у формі, щоб адміністратор міг зв'язатися з ним. Своєю чергою, адміністратор зобов’язаний оперативно реагувати на нові заявки. Веб-застосунок міститиме також додатковий функціонал, зокрема реєстрацію та авторизацію користувачів, калькулятор оренди, можливість створення звітів, відстеження термінів оренди, керування контентом, використання знижок та додаткових послуг, а також чітке розмежування прав доступу між ролями користувача та адміністратора. Реалізація проєкту передбачає розробку дизайну, створення клієнтської та серверної частин, формування бази даних та повне тестування системи.

1.4 Функціональні та нефункціональні вимоги до проектованої системи

Перелік функціональних вимог до проектованої системи:

- Реєстрація та авторизація.

Опис: Реєстрація нового користувача в системі.

Властивості: Вимагає введення ім'я, електронної пошти, пароля та ін.

- Відображення та редагування контенту.

Опис: Функції для перегляду та зміни інформації в системі оренди ТЗ.

Властивості:

- Список доступних транспортних засобів.
- Деталі транспортного засобу.
- Інформація про поточні та минулі оренди.
- Зміна інформації про транспортний засіб.
- Додавання нових транспортних засобів.
- Видалення транспортних засобів.
- Редагування статусу оренди.
- Управління інформацією про клієнтів.

- Пошук та фільтрація транспортних засобів

Опис: Надання можливості користувачам здійснювати пошук і фільтрацію доступних ТЗ.

Властивості: Пошук за маркою, моделлю, типом, ціною та іншими параметрами. Фільтрація за статусом доступності, класом ТЗ, типом пального тощо.

- Оформлення замовлення.

Опис: Оформлення замовлення на оренду транспортного засобу.

Властивості: Вимагає вибору транспортного засобу, вказання дати та тривалості оренди, вибір додаткових послуг.

- Калькулятор оренди.

Опис: Обчислення вартості оренди транспортного засобу.

Властивості: Враховує вартість оренди за годину, день або інший період, додаткові послуги, персональні знижки

- Створення звітів.

Опис: Створення звітів для адміністраторів системи.

Властивості: Формування звітів зі статистикою по оренді, популярності та попиту.

- Відстеження термінів оренди.

Опис: Відстеження термінів дії оренди транспортних засобів.

Властивості: Повідомлення про наближення до кінця терміну оренди.

- Історія дій користувача

Опис: Збереження журналу дій користувача в системі. *Властивості:* Перегляд історії бронювань.

Перелік нефункціональних вимог до проектованої системи:

1. Безпека

Опис: Система повинна забезпечувати надійний захист конфіденційної інформації користувачів.
Властивості:

- Захист паролів за допомогою хешування.
- Аутентифікація та авторизація користувачів з використанням токенів (JWT).
- Обмеження доступу на основі ролей.

- Захист від атак типу SQL-ін’єкцій, CSRF.
- Логи безпеки для виявлення підозрілої активності.

2. Надійність

Опис: Система повинна стабільно функціонувати у будь-яких умовах.
Властивості:

- Високий рівень доступності (мінімальний час простою).
- Обробка неочікуваних помилок без втрати даних.

3. Швидкодія

Опис: Система повинна оперативно реагувати на запити користувача.
Властивості:

- Час відповіді інтерфейсу — не більше 2 секунд для стандартних операцій.
- Оптимізація запитів до бази даних.
- Кешування часто використовуваних даних.

4. Ефективність

Опис: Система повинна оптимально використовувати апаратні та програмні ресурси.
Властивості:

- Використання асинхронної обробки для довготривалих запитів.
- Мінімізація споживання пам’яті та CPU.

5. Масштабованість

Опис: Система повинна бути готова до збільшення навантаження.
Властивості:

- Горизонтальна та вертикальна масштабованість.
- Підтримка розподіленої архітектури.

- Можливість додавання нових функцій без суттєвих змін у системі.

6. Сумісність

Опис: Система повинна коректно працювати на різних пристроях та в різних середовищах.

Властивості:

- Підтримка сучасних браузерів (Chrome, Firefox, Edge, Safari).
- Адаптивний дизайн для мобільних пристроїв.

7. **Документація**

Опис: Повна та актуальна документація для розробників. *Властивості:* API-документація (Swagger).

1.5 Висновок до розділу 1

У межах даного розділу було виконано моделювання предметної області, що дало змогу глибше зрозуміти структуру майбутньої системи та особливості її функціонування. Побудовано діаграму прецедентів, яка відобразила ключові функції системи з погляду користувачів та допомогла визначити ролі основних акторів — клієнта, менеджера та адміністратора. Також було розроблено діаграму діяльності, яка наочно продемонструвала послідовність виконання дій під час процесу оренди транспортного засобу — від авторизації до завершення оренди або компенсації.

Моделювання дозволило структурувати функціональність, виявити логіку основних бізнес-процесів, а також сформувати основу для проєктування архітектури, бази даних і взаємодії між компонентами системи. Отримані результати є важливим етапом у забезпеченні цілісності, зручності та надійності майбутнього веб-застосунку.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Моделювання предметної області є важливим етапом під час розробки будь-якого програмного забезпечення, оскільки дозволяє глибше зрозуміти специфіку задачі, виявити ключові об’єкти, їхні властивості та взаємозв’язки. Це допомагає структуровано описати, як працює реальний процес, який має бути реалізований у системі. Завдяки моделюванню можна на ранньому етапі виявити можливі помилки, суперечності або пропущені вимоги, що значно знижує ризик проблем у подальшій розробці. Також воно слугує основою для правильного проєктування архітектури системи, створення бази даних і реалізації бізнес-логіки. Крім того, модель предметної області є універсальним засобом комунікації між замовником, аналітиками, розробниками та тестувальниками. У межах цього розділу будуть детально розглянуті діаграми прецедентів, діяльності та послідовності, які дозволяють відобразити функціональні можливості системи, сценарії взаємодії користувачів із системою та порядок виконання основних процесів.

2.1 Діаграма прецедентів

Основна ідея діаграми прецедентів полягає в тому, щоб наочно показати, які функції виконує система з точки зору її взаємодії з користувачами або іншими зовнішніми об’єктами. Така діаграма дозволяє зрозуміти, що саме система повинна робити, не заглиблюючись у технічні деталі реалізації. Вона фокусується на поведінці системи, яку очікує кінцевий користувач або зовнішній актор (Рис. 2.1).

У центрі діаграми знаходяться **прецеденти (випадки використання)** — окремі дії або сценарії, які система повинна реалізувати, наприклад, «Увійти в систему», «Оформити оренду, «Застосувати фільтри». Кожен з них пов’язаний із **актором** — користувачем чи зовнішньою системою, яка ініціює взаємодію. Такий підхід дозволяє легко виявити функціональні вимоги, визначити ролі користувачів та встановити межі системи.

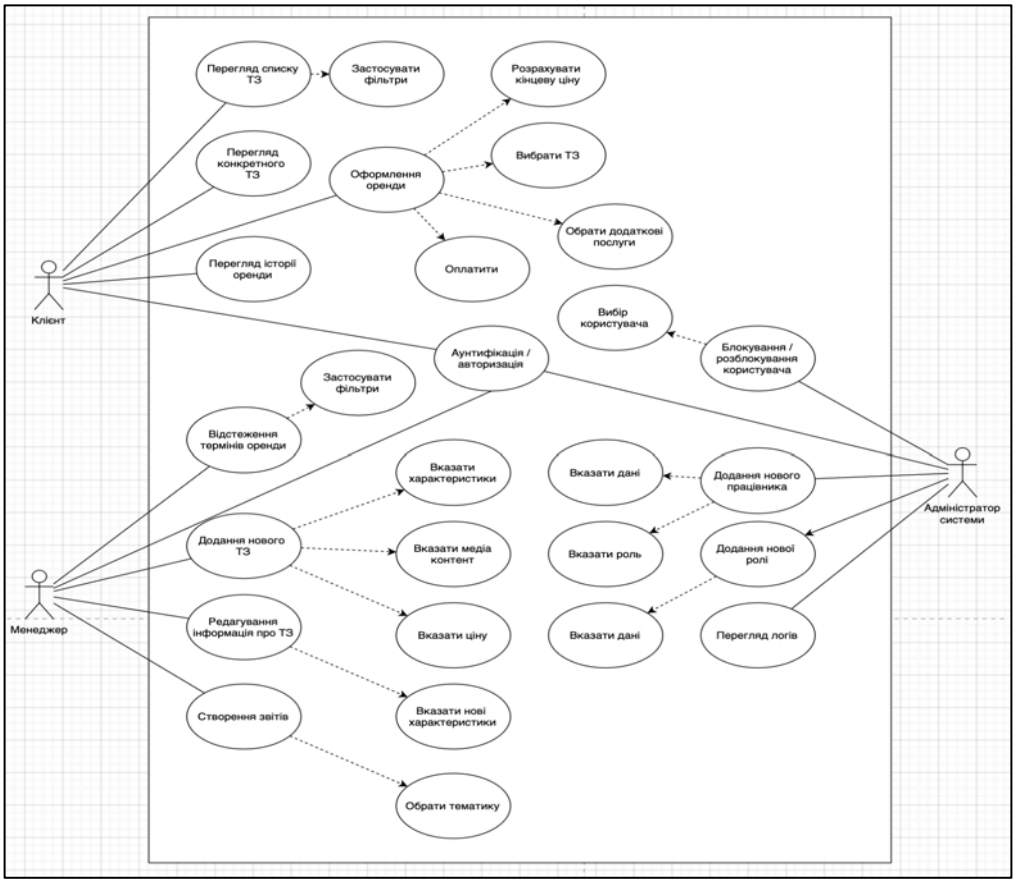

Рис. 2.1 – Діаграма Use case

Розглянемо інформацію про акторів створеної системи (табл.2.1) та опис варіантів використання (табл.2.2).

Таблиця 2.1 – Детальний опис акторів діаграми Use case

Актор	Опис
Клієнт	Користувач системи для оренди авто.
Менеджер	Працівник, який додає нові ТЗ, редагує дані, створює звіти, відстежує терміни оренди.
Адміністратор системи	Користувач з повним доступом: керує ролями, користувачами, додаванням працівників, має доступ до логів.

Таблиця 2.2 – Детальний опис прецедентів діаграми Use case

Назва прецеденту	Опис дії
Перегляд списку ТЗ	Клієнт переглядає всі доступні транспортні засоби.
Застосувати фільтри	Вибір параметрів для точнішого пошуку ТЗ.
Перегляд конкретного ТЗ	Перегляд детальної інформації про вибраний транспорт.
Оформлення оренди	Процес оформлення заявки на оренду.
Обрати додаткові послуги	Вибір додаткових послуг (напр. GPS, дитяче крісло).
Розрахувати кінцеву ціну	Автоматичний підрахунок вартості оренди.

Оплатити	Проведення оплати за оренду.
Перегляд історії оренди	Перегляд попередніх замовлень.
Аутентифікація / авторизація	Вхід у систему.
Відстеження терміну оренди	Контроль за тривалістю оренди.
Додавання нового ТЗ	Додавання нового транспорту до системи.
Редагування інформації про ТЗ	Зміна даних про наявні ТЗ.
Створення звітів	Генерація звітності щодо оренди, клієнтів тощо.
Вказати нові характеристики	Додавання нових параметрів до вже існуючого ТЗ.
Блокування / розблокування користувача	Управління доступом до системи.
Додавання нового працівника	Додавання облікового запису для менеджера.
Додавання нової ролі	Створення нових ролей для системи.
Перегляд логів	Аналіз дій користувачів у системі.

Діаграма прецедентів є простим, але потужним інструментом на етапі аналізу та проєктування, особливо на початку розробки програмного

забезпечення. Вона сприяє кращому розумінню очікувань замовника, допомагає формалізувати вимоги та створює спільну мову між аналітиками, розробниками та клієнтами.

2.2 Діаграма діяльності

Діаграма діяльності показує логіку виконання певного процесу — які дії виконуються, у якому порядку, які умови визначають перехід між діями, та хто бере участь у цих діях. Вона зручна для візуалізації робочих процесів, бізнес-процедур або алгоритмів у програмному забезпеченні.

У межах дипломної роботи було розроблено діаграму діяльності, яка відображає повний процес оренди транспортного засобу (Рис. 2.1). Ця діаграма демонструє основні етапи взаємодії користувача з системою — від моменту авторизації, перегляду доступних ТЗ, вибору додаткових послуг та надсилання запиту на оренду, до оплати, оформлення договору та завершення оренди.

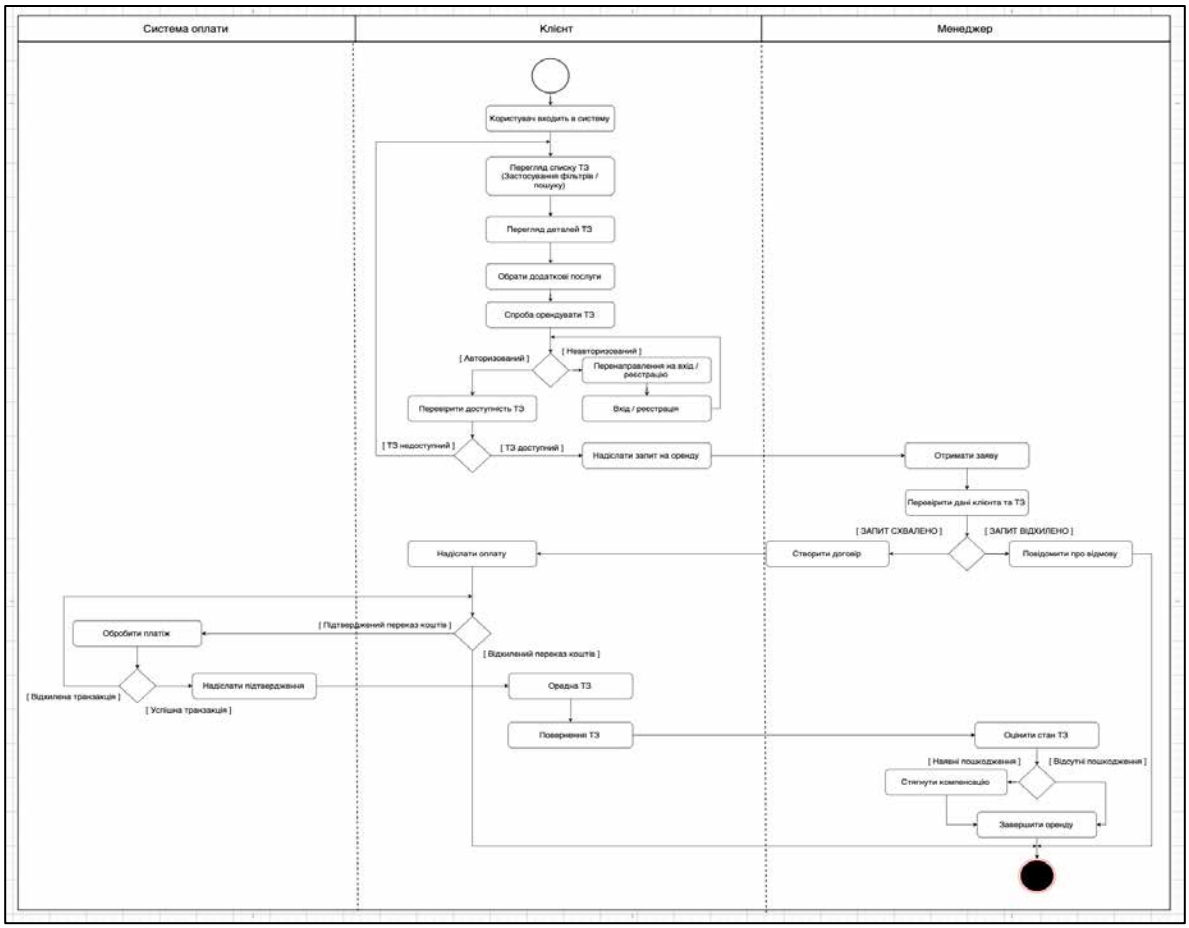

Рис. 2.1 – Діаграма діяльності activity diagram

Сформуємо приклад процесу оформлення оренди транспортного засобу

Опис процесу:

1. Клієнт починає з авторизації в системі.

2. Потім клієнт переглядає список доступних ТЗ, може застосовувати фільтри чи пошук, переглядати деталі конкретного ТЗ та обирати додаткові послуги.

3. Після цього він ініціює спробу оформлення оренди, але спочатку перевіряється, чи клієнт авторизований.

- Якщо не авторизований – перенаправляється на сторінку входу/реєстрації.

4. Далі система перевіряє доступність ТЗ:

- Якщо недоступний – оренду неможливо продовжити.
- Якщо доступний – клієнт надсилає запит на оренду.

5. Менеджер отримує запит, перевіряє дані клієнта і ТЗ:

- Якщо запит схвалено – менеджер створює договір.
- Якщо відхилено – клієнт отримує повідомлення про це.

6. Клієнт здійснює оплату через систему оплати:

- Якщо транзакція відхилена – процес переривається.
- Якщо успішна – підтвердження надсилається клієнту.

7. Після оплати відбувається оренда ТЗ.

8. По завершенню оренди клієнт повертає ТЗ, а менеджер оцінює його стан:

- Якщо є пошкодження – запускається процес компенсації.
- Інакше – оренда завершується.

2.3 Висновок до розділу 2

У межах даного розділу було виконано моделювання предметної області, що дало змогу глибше зрозуміти структуру майбутньої системи та особливості її функціонування. Побудовано діаграму прецедентів, яка відобразила ключові функції системи з погляду користувачів та допомогла визначити ролі основних акторів — клієнта, менеджера та адміністратора. Також було розроблено діаграму діяльності, яка наочно продемонструвала послідовність виконання дій під час процесу оренди транспортного засобу — від авторизації до завершення оренди або компенсації.

Моделювання дозволило структурувати функціональність, виявити логіку основних бізнес-процесів, а також сформувати основу для проєктування архітектури, бази даних і взаємодії між компонентами системи. Отримані результати є важливим етапом у забезпеченні цілісності, зручності та надійності майбутнього веб-застосунку.

3 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування інформаційного та програмного забезпечення є важливим етапом у процесі створення інформаційної системи, що визначає її логічну структуру, архітектуру та механізми функціонування. Саме на цьому етапі відбувається деталізація рішень, прийнятих під час моделювання предметної області, із подальшим формалізованим описом компонентів системи, їхніх взаємозв’язків і способів обробки даних. Якісне проєктування дозволяє забезпечити відповідність програмної реалізації вимогам замовника, спростити процес розробки, полегшити подальше тестування та супровід системи.

В межах цього розділу будуть розглянуті логічна модель даних, діаграма класів та діаграма пакетів, які дозволяють формалізувати структуру інформації, описати програмні компоненти, їхню поведінку та ієрархію. Ці моделі слугують основою для побудови бази даних, реалізації бізнес-логіки та визначення архітектурного підходу до розробки. Крім того, вони є ефективним засобом комунікації між розробниками, аналітиками й іншими учасниками проєкту, забезпечуючи узгоджене бачення системи перед початком етапу програмування.

3.1 Логічна модель даних

Перед початком створення бази даних у системі управління базами даних (СУБД) необхідно побудувати її структуру у вигляді **ER-діаграми** (англ. *Entity-Relationship diagram*), яка є зручним і наочним способом моделювання концептуальної схеми даних. Вона дозволяє на ранньому етапі проєктування описати ключові сутності предметної області, їхні атрибути та зв’язки між ними.

Модель "сутність-зв'язок" (ER-модель) — це модель даних, яка забезпечує формалізований підхід до опису логічної структури даних за допомогою узагальнених конструкцій у вигляді блоків і зв’язків. Її також називають мета-моделлю, оскільки вона виступає як інструмент для побудови інших моделей даних, які згодом реалізуються в конкретній СУБД.

Під час побудови ER-діаграми особливу увагу слід приділяти коректній нормалізації даних. Зокрема, важливо перевіряти, щоб кожен неключовий атрибут нетранзитивно залежав від первинного ключа, що відповідає вимогам **третьої нормальної форми (3NF)**. Дотримання цього принципу дозволяє уникнути надмірності даних і забезпечити цілісність інформації в базі даних.

Таким чином, логічна модель даних у вигляді ER-діаграми є важливим інструментом проєктування, який використовується для створення інформаційних систем, баз даних, а також архітектури прикладного програмного забезпечення. Приклад створення ER діаграми наведено нижче (Рис. 3.1).

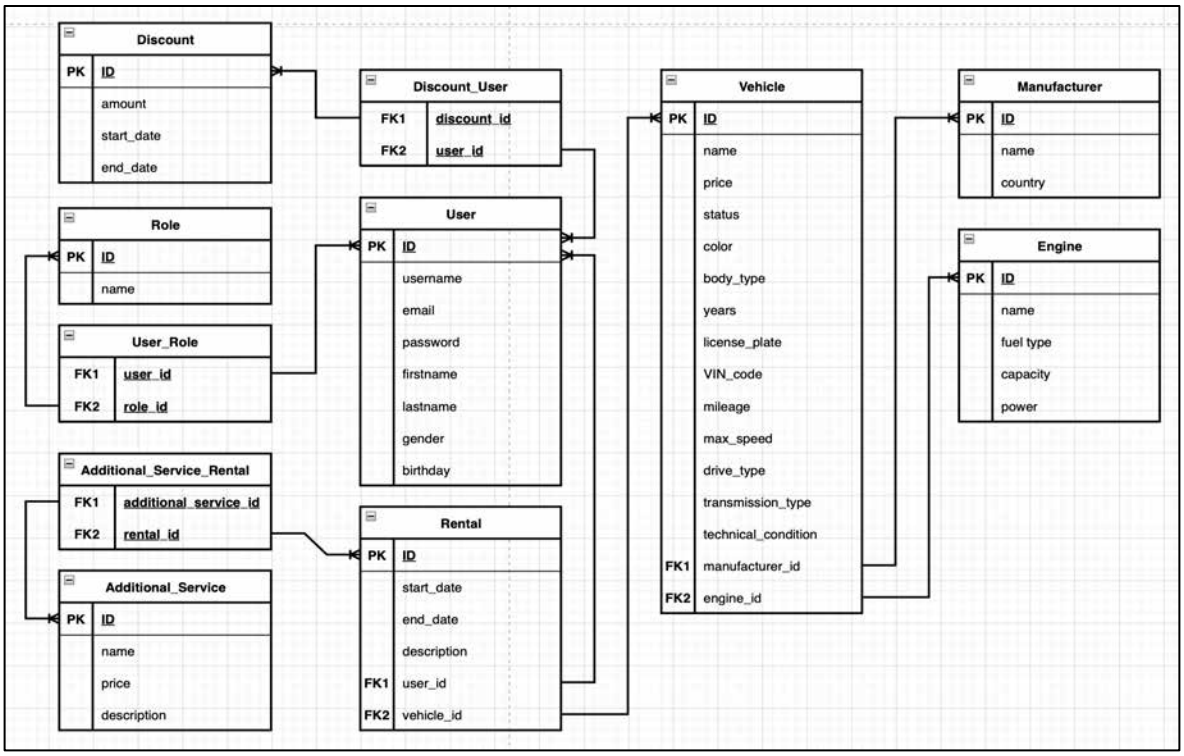

Рис. 3.1 – Приклад ER діаграми

Наступним етапом було виділено десять основних сутностей, більше інформації про них представлено у табл. 3.1.

Таблиця 3.1 – Опис сутностей ER діаграми

Назва таблиці	Опис
User	Зберігає інформацію про користувачів системи
Role	Містить ролі користувачів (наприклад, адміністратор, клієнт)

User_Role	Реалізує зв’язок "багато до багатьох" між користувачами та ролями
Vehicle	Інформація про транспортні засоби
Manufacturer	Дані про виробника транспортного засобу
Engine	Описує технічні характеристики двигуна
Rental	Зберігає інформацію про оренду (дата початку, завершення, опис)
Discount	Інформація про знижки (розмір, дата дії)
Discount_User	Реалізує зв’язок "багато до багатьох" між знижками та користувачами
Additional_Service	Перелік додаткових послуг (назва, ціна, опис)
Additional_Service_Rental	Реалізує зв’язок "багато до багатьох" між орендами та додатковими послугами

Логічна модель відповідає вимогам третьої нормальної форми (3NF), оскільки всі неключові атрибути залежать виключно від первинного ключа відповідної сутності й немає транзитивних залежностей. Це забезпечує оптимальну організацію даних, уникає надлишковості та підвищує узгодженість у системі.

3.2 Діаграма класів

Діаграма класів (Рис. 3.2) є складовою частиною об’єктно-орієнтованого аналізу та проєктування, яка дозволяє формалізовано представити структуру системи у вигляді класів, їхніх атрибутів, методів та зв’язків між ними. На відміну від ER-діаграми, що зосереджується переважно на логічних структурах даних, діаграма класів охоплює як дані (атрибути), так і поведінку об'єктів (методи), що особливо актуально під час розробки програмного забезпечення.

У даному проєкті діаграма класів відображає ключові сутності предметної області, пов’язаної з системою оренди транспортних засобів, а також взаємозв’язки між ними.

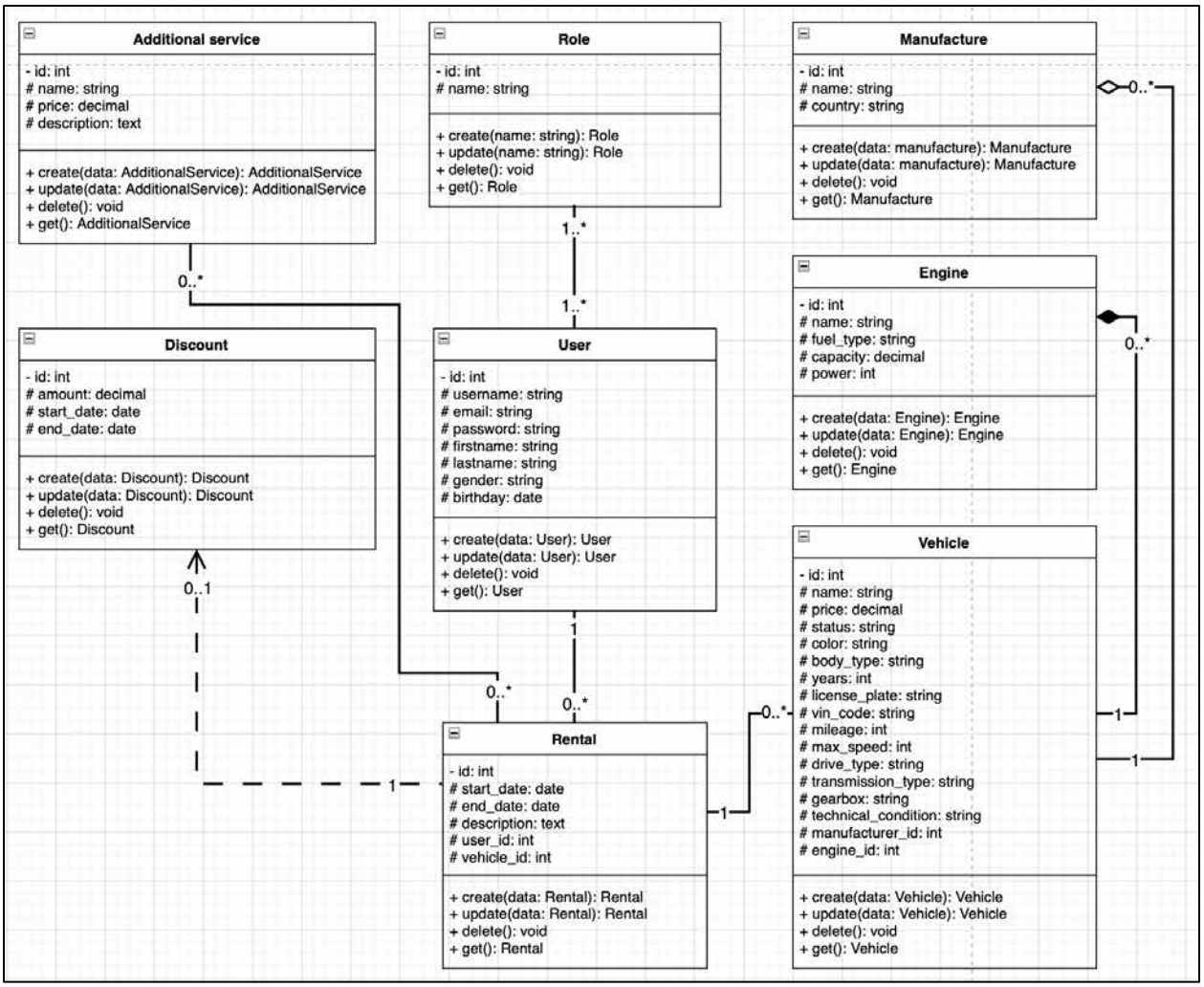

Рис. 3.2 – Діаграма класів

Кожен клас включає стандартні методи CRUD (create, read, update, delete), що дозволяє інтегрувати його у програмну логіку застосунку та забезпечує можливість обробки даних на рівні бізнес-логіки.

Важливо, що діаграма класів узгоджена з принципами третьої нормальної форми (3NF), оскільки всі атрибути класів функціонально залежать лише від первинного ключа, а зв’язки між класами дозволяють уникати дублювання інформації. Наприклад, двигун моделюється окремо,

оскільки він може бути спільним для кількох транспортних засобів, що запобігає надмірності даних.

Таким чином, діаграма класів виконує роль логічної моделі в об’єктно-орієнтованій парадигмі та слугує основою для подальшої реалізації архітектури системи, включаючи базу даних, API-інтерфейси та бізнес-логіку.

3.3 Діаграма пакетів

У процесі проєктування великих інформаційних систем важливим етапом є структуризація системи на логічні підсистеми або модулі. Для цього використовується діаграма пакетів (англ. *Package Diagram*) — один з типів діаграм UML, який дозволяє візуально представити організацію системи на рівні модулів, залежності між ними та межі відповідальності.

Діаграма пакетів (Рис. 3.3) дає змогу групувати класи, інтерфейси та інші структурні елементи в пакети відповідно до їхньої функціональності. Це сприяє кращому розподілу обов’язків, підвищує читабельність архітектури та спрощує підтримку й масштабування системи. Особливо це актуально для командної розробки, коли різні частини проєкту можуть розроблятися паралельно.

У межах даного проєкту діаграма пакетів відображає логічне розбиття системи оренди транспортних засобів на основні компоненти.

Кожен пакет може містити власні класи, інтерфейси та залежності. Зв’язки між пакетами відображають напрямок використання або імпорту

функціоналу одного пакета іншим, що дозволяє зберігати модульність і низький рівень зв’язності між компонентами.

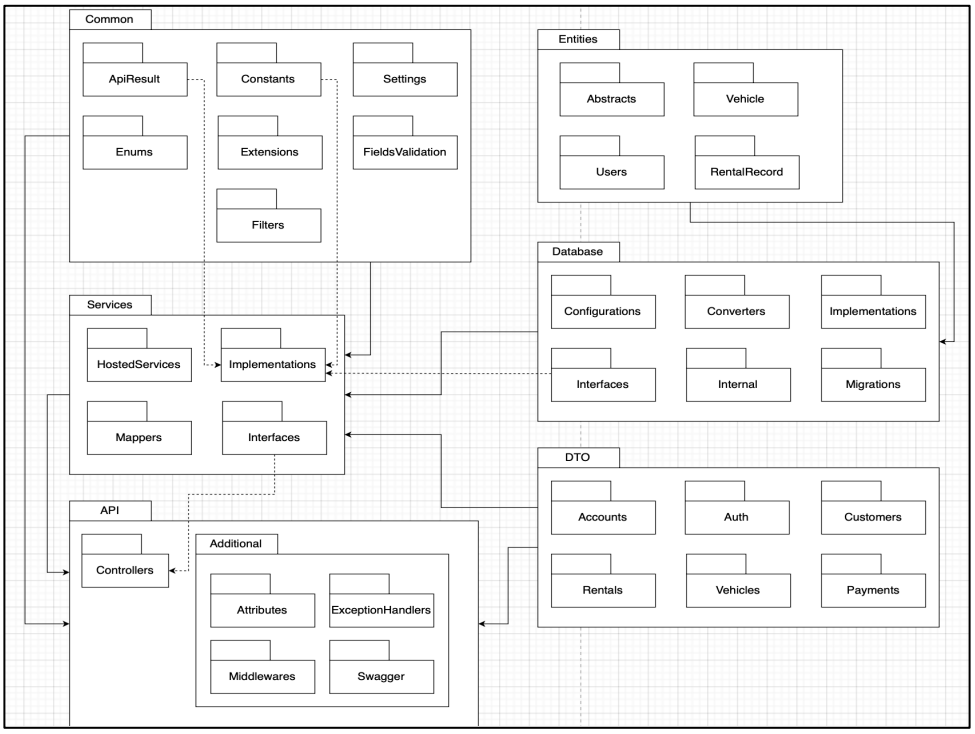

Рис. 3.3 - діаграми пакетів мови UML

1. Common (загальні утиліти, які можуть використовуватися в усій системі):

○ ApiResult — для повернення стандартного формату відповіді API.

○ Constants — зберігання констант, таких як максимальна кількість оренд або знижки.

○ Enums — перерахування типів транспортних засобів, статусів оренд тощо.

○ Extensions — розширення для роботи з основними об’єктами та колекціями.

- FieldsValidation — валідація полів, як-от перевірка водійських прав або документів.
- Filters — фільтри для обмеження доступу до певних ресурсів.
- Settings — загальні налаштування додатка.

2. Entities (сутності домену):

- Abstracts — базові абстрактні класи для сутностей.
- Vehicle — клас, що представляє транспортний засіб.
- RentalRecord — клас для запису інформації про оренду.
- Customer — клас для представлення користувачів системи (орендарів).

3. Database (рівень доступу до бази даних):

- Configurations — конфігурації для підключення до бази даних.
- Converters — конвертери для специфічних типів даних.
- Implementations — реалізації для роботи з базою даних.
- Interfaces — інтерфейси для доступу до бази даних.
- Internal — допоміжні класи для внутрішніх операцій з даними.
- Migrations — міграції для оновлення структури бази даних.

4. Services (логіка бізнес-процесів):

- HostedServices — послуги для обробки фонового завдання (наприклад, очищення завершених оренд).
- Implementations — реалізації послуг для бізнес-логіки, як-от обробка нових оренд.
- Interfaces — інтерфейси для послуг бізнес-логіки.
- Mappers — мапери для перетворення даних між сутностями та DTO.
- StaticMappers — статичні мапери для специфічних перетворень.

5. DTOs (Data Transfer Objects, об’єкти передачі даних):

- Accounts — DTO для інформації облікового запису користувача.
- Auth — DTO для авторизації та аутентифікації.
- Customers — DTO для даних клієнтів.
- Rentals — DTO для інформації про оренду.
- Vehicles — DTO для інформації про транспортні засоби.
- Payments — DTO для інформації про платежі.

6. WebAPI (веб-інтерфейс):
   - Controllers — контролери для обробки запитів користувача.
7. Additional (додаткові компоненти для поліпшення API):
   - Attributes — атрибути для додаткових перевірок і обмежень.
   - ExceptionHandlers — обробка виключень.
   - Middlewares — проміжні обробники для обробки запитів і відповідей.
   - Swagger — документація API (автоматично генерується).

Взаємозв'язки між компонентами:

- Controllers у API взаємодіють з Services для обробки бізнес-логіки.
- Services працюють з Database для збереження або отримання даних із бази.
- DTOs використовуються для передачі даних між рівнями (зокрема між WebAPI та Services).
- Common та Additional забезпечують допоміжний функціонал для всієї системи.

Ця архітектура підтримує модульність, безпеку та масштабованість системи оренди транспортних засобів, дозволяючи легко додавати нові функції або оновлювати існуючі.

3.4 Висновок до розділу 3

У цьому розділі було виконано проєктування логіки та структури системи. Сформовано ER-діаграму, діаграму класів і пакетів. Особливу увагу приділено нормалізації даних та модульності архітектури, що забезпечило основу для реалізації стабільної та масштабованої системи.

4 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

4.1 Архітектура застосунку

У контексті створення сучасної, зручної та ефективної інформаційної системи важливим етапом є вибір відповідної архітектури, яка забезпечить стабільність, масштабованість та легкість у підтримці. Зважаючи на ці вимоги, було прийнято рішення реалізувати систему на основі REST API (Рис. 4.1).

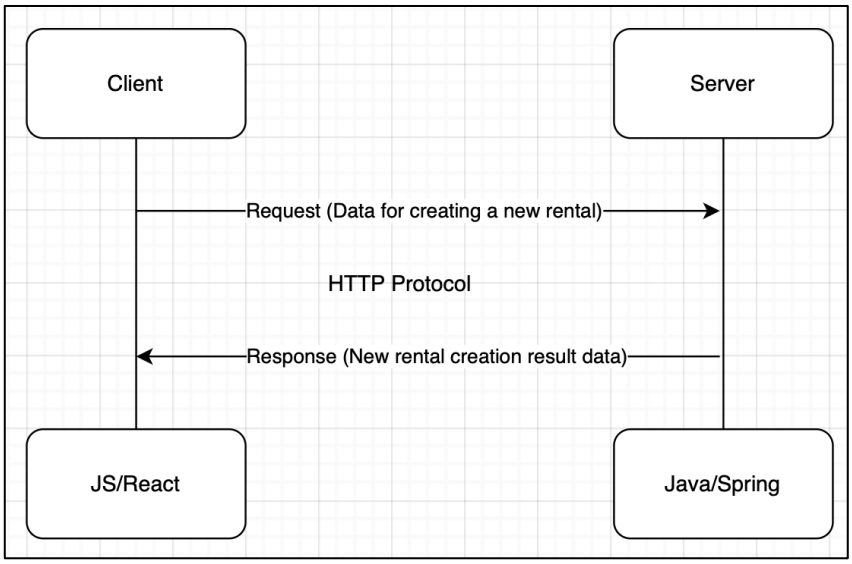

Рис. 4.1 - Схематичний приклад взаємодії клієнта та сервера.

REST API (Representational State Transfer) дозволяє будувати легкі, швидкі та масштабовані веб-служби, в основі яких лежить чітка структура ресурсів та взаємодії через HTTP-методи: GET, POST, PUT, DELETE.

Основні переваги REST API, що визначили мій вибір:

- Простота реалізації: REST API використовує стандартні та зрозумілі HTTP-методи. Це дозволяє легко розробляти й підтримувати функціонал, навіть у невеликих командах або в умовах обмежених ресурсів.
- Масштабованість: Завдяки клієнт-серверній природі REST API, додаток легко масштабується, що дозволяє обробляти велику кількість запитів без втрати продуктивності. Також є можливість реалізувати кешування та балансування навантаження.
- Незалежність компонентів: Клієнт та сервер працюють незалежно один від одного — сервер реалізує логіку, а клієнт (веб, мобільний додаток тощо) лише звертається до API. Це спрощує підтримку, оновлення й інтеграцію з іншими платформами.
- Універсальність і сумісність: REST API не залежить від мови програмування чи платформи. Його можна інтегрувати з будь-якими сервісами, що підтримують HTTP — від мобільних додатків до зовнішніх платіжних шлюзів.
- Легкість тестування: Через використання стандартного протоколу HTTP REST API легко тестується за допомогою таких інструментів, як Postman, curl або Swagger.

4.2 Складова клієнтського рівня (Frontend)

Клієнтська частина інформаційної системи реалізована з використанням технології React — популярної JavaScript-бібліотеки для створення інтерактивних інтерфейсів користувача. Основне призначення цього рівня — забезпечення зручної та ефективної взаємодії користувача із системою через графічний інтерфейс.

Для побудови елементів інтерфейсу використовується бібліотека Material UI (MUI) — набір компонентів інтерфейсу, що реалізує принципи Material Design від Google. Це дозволяє створювати сучасний, адаптивний та візуально привабливий UI з уніфікованим стилем і поведінкою елементів.

Основні функціональні компоненти клієнтського рівня:

- **Форми реєстрації та авторизації.** Даний компонент забезпечує вхід користувача в систему. Аутентифікація реалізована за допомогою JWT-токенів (JSON Web Token), що дозволяє зберігати стан сесії користувача та захищати доступ до обмежених ресурсів. Для оформлення форм використовуються компоненти Material UI, такі як TextField, Button, FormControl тощо.
- **Головна сторінка.** Містить каталог транспортних засобів з можливістю фільтрації за типом, маркою, ціною тощо, а також функцію сортування (наприклад, за ціною чи рейтингом). Візуальне відображення списку реалізоване за допомогою компонентів Card, Grid, Select з бібліотеки MUI.
- **Кошик бронювань.** На цій сторінці користувач може переглядати обрані для оренди транспортні засоби, редагувати список та

переходити до підтвердження бронювання. Компоненти бібліотеки використовуються для структурованого та інтуїтивного відображення інформації.

- **Панель адміністратора.** Реалізований окремий модуль для адміністраторів системи, що дозволяє здійснювати керування базою транспортних засобів та користувачів: додавання, редагування, видалення записів. Інтерфейс також побудований з використанням компонентів Material UI, зокрема DataGrid, Dialog, Snackbar для повідомлень про результати операцій.
- **Роутинг.** Для навігації між різними сторінками застосовується бібліотека react-router. Це забезпечує SPA-архітектуру (Single Page Application), за якої перехід між сторінками здійснюється без повного перезавантаження веб-застосунку.

Використання бібліотеки Material UI у поєднанні з React забезпечує не лише естетично привабливий вигляд застосунку, а й прискорює розробку завдяки готовим адаптивним компонентам і стилям, що відповідають сучасним стандартам веб-дизайну.

4.3 Складова серверногорівня (Backend)

Серверна частина інформаційної системи розроблена з використанням фреймворку Java Spring Boot, що є потужним інструментом для створення масштабованих, безпечних та продуктивних веб-додатків. Backend відповідає за обробку бізнес-логіки, збереження та обробку даних, а також забезпечує взаємодію з клієнтською частиною через REST API.

Основні компоненти серверної архітектури:

- Spring Boot. Фреймворк Spring використовується для конфігурації та запуску веб-застосунку. Spring Boot спрощує процес налаштування, дозволяючи швидко розгортати та тестувати серверну частину завдяки вбудованому вебсерверу (Tomcat).
- Робота з базою даних (PostgreSQL + JPA). Для зберігання даних використовується реляційна база даних PostgreSQL, яка відома своєю надійністю та продуктивністю. Для доступу до бази даних застосовується Spring Data JPA, що забезпечує ORM (Object-Relational Mapping) через бібліотеку Hibernate. Це дозволяє працювати з сутностями Java як з таблицями бази даних без необхідності писати SQL-запити вручну.
- JWT-аутентифікація. Система аутентифікації реалізована на основі JWT (JSON Web Token). Після входу користувача система генерує токен, який використовується для підтвердження авторизованого доступу до захищених ресурсів API. Це забезпечує безпечну та безстейтову (stateless) взаємодію між клієнтом і сервером.
- Контейнеризація за допомогою Docker. Для полегшення розгортання, масштабування та тестування застосунку використовується Docker. Серверна частина, база даних та інші сервіси ізольовано упаковані в контейнери, що забезпечує стабільність середовища виконання на різних етапах життєвого циклу застосунку. Конфігурація контейнерів виконується у файлах Dockerfile та docker-compose.yml.

Реалізація серверного рівня на базі Java Spring у поєднанні з PostgreSQL, JWT та Docker дозволяє створити надійну, безпечну та легко масштабовану систему, що відповідає сучасним вимогам до веб-застосунків.

4.4 Висновок до розділу 4

Було обґрунтовано вибір сучасних технологій для реалізації як клієнтської, так і серверної частини застосунку. Реалізовано REST API, інтегровано JWT-аутентифікацію, застосовано Docker для контейнеризації. Такий технологічний стек забезпечив продуктивність, безпеку та зручність у розгортанні.

5. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПЗ СИСТЕМИ

5.1 REST API

REST API (Representational State Transfer Application Programming Interface) є основним способом взаємодії між клієнтською частиною системи (React-додаток) і серверною частиною (Spring Boot-додаток). API побудовано відповідно до архітектурного стилю REST (Табл. 5.1), який забезпечує простоту, масштабованість та незалежність між клієнтом і сервером.

Основні принципи REST API:

- Клієнт-серверна архітектура – клієнт надсилає HTTP-запити до API, сервер обробляє їх та повертає відповіді.
- Безстанність (stateless) – кожен запит містить усю необхідну інформацію, сесії не зберігаються на сервері.
- Ідентифікація ресурсів через URI – кожен ресурс доступний за унікальним шляхом.
- HTTP методи використовуються відповідно до CRUD-операцій:
    - GET – отримання даних
    - POST – створення нового ресурсу
    - PUT/PATCH – оновлення ресурсу
    - DELETE – видалення ресурсу

Таблиця 5.1 – Опис шляхів REST API

Шлях (Endpoint)	Метод	Опис
/api/v1/rentals	POST	Створення нової оренди
/api/v1/rentals/{id}	PUT	Оновлення існуючої оренди за ID
/api/v1/rentals/{id}/change-status	PATCH	Зміна статусу оренди
/api/v1/rentals	GET	Отримання списку всіх оренд (опційно – за статусом)
/api/v1/rentals/personal	GET	Отримання списку персональних оренд поточного користувача
/api/v1/rentals/statuses	GET	Отримання списку доступних статусів оренди

5.2 Приклад реалізації на backend частині

Для реалізації backend частини програмного забезпечення було використано фреймворк Spring Boot, який забезпечує зручне створення RESTful веб-сервісів. Нижче наведено приклад реалізації контролера RentalController, який відповідає за обробку запитів, пов’язаних з орендою об'єктів (Рис. 5.2).

```java
@RestController
@RequestMapping("/api/v1/rentals")
@RequiredArgsConstructor
public class RentalController {

    private final RentalService rentalService;
    private final ModelMapper modelMapper;

    @PostMapping
    public ResponseEntity<Object> create(@RequestBody RentalRequest request) {
        return generateSuccessfulResponse(
                modelMapper.map(rentalService.create(request), RentalResponse.class),
                HttpStatus.CREATED
        );
    }

    @PutMapping("/{id}")
    public ResponseEntity<Object> update(@PathVariable UUID id, @RequestBody RentalRequest request) {
        return generateSuccessfulResponse(
                modelMapper.map(rentalService.update(id, request), RentalResponse.class),
                HttpStatus.OK
        );
    }

    @PatchMapping("/{id}/change-status")
    public ResponseEntity<Object> changeStatus(@PathVariable UUID id, @RequestParam ERentalStatus status) {
        return generateSuccessfulResponse(
                modelMapper.map(rentalService.changeStatus(id, status), RentalResponse.class),
                HttpStatus.OK
        );
    }

    @GetMapping
    public ResponseEntity<Object> getAll(@RequestParam(required = false) ERentalStatus status) {
        List<RentalDto> rentals;
        if (status != null) {
            rentals = rentalService.getAllByStatus(status).stream()
                    .map(rental -> modelMapper.map(rental, RentalDto.class))
                    .toList();
        } else {
            rentals = rentalService.getAll().stream()
                    .map(rental -> modelMapper.map(rental, RentalDto.class))
                    .toList();
        }
        return generateSuccessfulResponse(rentals, HttpStatus.OK);
    }
}
```

Рис. 5.1 - Приклад реалізації контролера RentalController

Опис реалізації:

- Контролер обробляє запити на створення, оновлення, зміну статусу та отримання переліку оренд.
- Для перетворення між сутностями та DTO використовується бібліотека ModelMapper.
- Кожен метод повертає відповідь з HTTP статусом та відповідними даними.

- Методи @GetMapping, @PostMapping, @PutMapping, @PatchMapping реалізують основні CRUD-операції.

Структура проєкту:

- RentalService — сервісний шар для бізнес-логіки.
- RentalRequest, RentalResponse, RentalDto — DTO класи.
- ERentalStatus — enum для збереження можливих статусів оренди.
- generateSuccessfulResponse(...) — утилітний метод для формування стандартної відповіді API.

5.3 Притримання єдиного стилю

Для забезпечення зручності супроводу, розширення та командної розробки програмного забезпечення, в процесі реалізації backend-частини було притриманося єдиного стилю програмування, який включає в себе наступні принципи:

1. Іменування

- Класи мають іменування згідно з Java-конвенціями: CamelCase (наприклад, RentalService, RentalController).
- Методи називаються дієсловами, що відповідають їхній функціональності: create(), update(), getAllPersonal().
- Змінні називаються зрозуміло та відповідно до контексту, наприклад: request, status, rentals.

2. Організація коду

- Код розділено на шари: контролер, сервіс, репозиторій, DTO, конфігурації.
- Кожен клас виконує одну чітко визначену роль, відповідно до принципу Single Responsibility (SRP).

3. Форматування

- Використовується автоформатування коду за допомогою IDE (наприклад, IntelliJ IDEA) з увімкненим перевіренням форматування згідно зі стандартами Java.
- Всі відступи, дужки та відстані розміщено згідно з прийнятими конвенціями.

4. Анотації та ін’єкції

- Використовується Lombok (@RequiredArgsConstructor, @Getter, @Setter) для зменшення шаблонного коду.
- Ін’єкція залежностей здійснюється через конструктор — без використання @Autowired прямо в полі.

5. Коментарі та документація

- Коментарі додаються лише там, де логіка складна або нетипова.
- Перевага віддається чистому коду, де назви методів і змінних самі пояснюють свою суть.

6. Обробка відповідей

- Всі відповіді REST API уніфіковані через загальний формат — метод generateSuccessfulResponse(...)повертає стандартну структуру відповіді з HTTP-статусом.

7. Конвенції для REST API

- URI структуровані згідно з REST-архітектурою: /api/v1/rentals, /api/v1/rentals/{id}/change-status.
- HTTP-методи відповідають своїм діям: GET, POST, PUT, PATCH.

5.4 Валідація даних та DTO.

У веб-додатку, що розробляється, важливою складовою є перевірка коректності вхідних даних, що надходять через REST API. Це дозволяє забезпечити захищеність та цілісність даних, які потрапляють у систему. Для цього застосовуються DTO (Data Transfer Object) та механізми валідації.

DTO — це об’єкти, що використовуються для передачі даних між клієнтською частиною (frontend) та сервером (backend). Вони дозволяють відокремити внутрішню модель домену від представлення, що використовується в API (Рис. 5.2).

```java
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class RentalRequest {

    private ERentalStatus status;

    @NotNull(message = "Start date is required")
    @FutureOrPresent(message = "Start date must be in the present or future")
    private LocalDateTime startDate;

    @NotNull(message = "End date is required")
    @Future(message = "End date must be in the future")
    private LocalDateTime endDate;

    @NotNull(message = "Vehicle ID is required")
    private UUID vehicleId;

    private List<UUID> additionalServicesIds;

    private UUID discountId;

}
```

Рис. 5.2 - DTO для запиту створення нової оренди

Також використовується DTO для відповіді API (Рис.5.3):

```java
@Data
public class RentalResponse {
    private LocalDateTime startDate;
    private LocalDateTime endDate;
    private VehicleDto vehicle;
    private double total;
    private List<AdditionalServiceDto> additionalServices;
}
```

Рис. 5.3 - DTO для відповіді створення нової оренди

Для реалізації валідації використовується специфікація Bean Validation, що підтримується у Spring за допомогою бібліотеки hibernate-validator. Анотації, такі як @NotNull, @Size, @Pattern, @Min, @Max тощо,

використовуються безпосередньо в DTO класах. Контролер, що приймає запит, містить анотацію @Valid.

У випадку некоректних даних система автоматично генерує відповідь із кодом помилки 400 Bad Request та повідомленням про помилку.

Для кращого UX реалізовано глобальний обробник винятків (Рис. 5.4):

```java
@Slf4j
@RestControllerAdvice
@RequiredArgsConstructor
public class ExceptionHandlingController {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Object> handleMethodArgumentNotValidException(MethodArgumentNotValidException ex) {
        var fieldErrors = ex.getBindingResult().getFieldErrors().stream()
                .map(error -> "Field '" + error.getField() + "' " + error.getDefaultMessage())
                .toList();
        var globalErrors = ex.getBindingResult().getGlobalErrors().stream()
                .map(DefaultMessageSourceResolvable::getDefaultMessage)
                .toList();
        var allErrors = new ArrayList<>();
        allErrors.addAll(fieldErrors);
        allErrors.addAll(globalErrors);
        log.warn(allErrors.toString(), ex);
        return ResponseHandler.generateUnsuccessfulResponse(
                allErrors.toString(),
                HttpStatus.BAD_REQUEST,
                HttpStatus.BAD_REQUEST.getReasonPhrase()
        );
    }

    @ExceptionHandler({
            EntityNotFoundException.class,
            IllegalArgumentException.class,
            HttpRequestMethodNotSupportedException.class
    })
    public ResponseEntity<Object> handleCommonBadRequestExceptions(Exception ex) {
        log.warn(ex.getMessage(), ex);
        return generateUnsuccessfulResponse(
                ex.getMessage(),
                HttpStatus.BAD_REQUEST,
                HttpStatus.BAD_REQUEST.getReasonPhrase()
        );
    }

}
```

Рис. 5.4 - Глобальний обробник винятків

Для забезпечення єдиного стилю відповіді було використано кастомний обробник (Рис. 5.5):

```java
public class ResponseHandler {

    public static ResponseEntity<Object> generateSuccessfulResponse(Object responseObj, HttpStatus status) {
        var map = new HashMap<String, Object>();
        map.put("success", true);
        map.put("data", responseObj);
        return new ResponseEntity<>(map, status);
    }

    public static ResponseEntity<Object> generateUnsuccessfulResponse(String message, HttpStatus status, Object error) {
        var map = new HashMap<String, Object>();
        map.put("success", false);
        map.put("message", message);
        map.put("error", error);
        return new ResponseEntity<>(map, status);
    }

}
```

Рис. 5.5 - Кастомний обробник

5.5 Використання ORM

Однією з ключових частин розробки серверної частини програмного забезпечення є взаємодія з базою даних. У даному проєкті для цієї мети використовується JPA (Java Persistence API) — офіційна специфікація Java для реалізації об'єктно-реляційного відображення (ORM).

JPA дозволяє працювати з реляційними базами даних через Java-об'єкти, не вдаючись до прямого написання SQL-запитів. Замість цього використовуються анотації, що описують, як класи й поля зіставляються з таблицями й стовпцями бази даних.

Основні переваги використання JPA:

- Абстракція над базами даних: JPA надає абстрактний рівень для роботи з базами даних, що дозволяє змінювати СУБД без значних змін у коді додатку.

- Зменшення кількості коду: Завдяки JPA розробники можуть уникати написання великої кількості SQL-запитів, замість цього використовуючи об'єктні моделі. Це значно зменшує обсяг коду, підвищує його читабельність та підтримуваність.
- Об'єктно-реляційне відображення (ORM): JPA автоматично зберігає об'єкти Java в базі даних, забезпечуючи відповідність між об'єктами в пам'яті та записами в базі даних. Це полегшує збереження та отримання даних.
- Кешування: JPA підтримує кешування, що може значно підвищити продуктивність додатку, зменшуючи кількість запитів до бази даних.
- Спрощення транзакцій: JPA підтримує управління транзакціями на рівні додатку, що робить роботу з транзакціями більш зручною та надійною.
- Гнучкість та розширюваність: JPA дозволяє налаштовувати поведінку збереження та завантаження даних, що дає розробникам велику гнучкість у налаштуванні роботи з даними.

Для прикладу розглянемо реалізацію сутності оренди (Рис. 5.6):

```java
@Entity
@Table(name = "rentals")
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(callSuper = true)
public class Rental extends BaseEntityAudit {

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private ERentalStatus status;

    @Column(nullable = false)
    private LocalDateTime startDate = LocalDateTime.now();

    @Column(nullable = false)
    private LocalDateTime endDate;

    @Column(columnDefinition = "TEXT")
    private String description;

    private double total;

    @ManyToOne
    @JoinColumn(name = "vehicle_id")
    private Vehicle vehicle;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToMany
    @JoinTable(
            name = "rentals_services",
            joinColumns = @JoinColumn(name = "rental_id"),
            inverseJoinColumns = @JoinColumn(name = "service_id")
    )
    private List<AdditionalService> additionalServices;

}
```

Рис. 5.6 - Сутність оренди з використанням ORM

У наведеному прикладі демонструється повноцінне використання JPA-анотацій для опису сутності Rental, яка представляє інформацію про оренду транспорту. Зв’язки між таблицями реалізуються через анотації @ManyToOneта @ManyToMany, що дає змогу органічно будувати складні об’єктні зв’язки між сутностями.

Завдяки ORM-архітектурі, розробники отримують інструменти для зручної, безпечної та ефективної роботи з даними у базі PostgreSQL, використовуючи зрозумілі об’єктні моделі.

5.6 Реалізація бізнес логіки

У процесі розробки програмного забезпечення важливу роль відіграє реалізація бізнес-логіки, яка визначає, як саме працює система відповідно до поставлених вимог. Для ефективної побудови, розширюваності та підтримуваності коду в серверній частині було використано шаблон проєктування "Стратегія" (Strategy pattern).

Патерн "Стратегія" дозволяє визначити родину алгоритмів, інкапсулювати кожен з них і зробити їх взаємозамінними. Цей шаблон дозволяє алгоритмам змінюватися незалежно від клієнтів, які їх використовують. У контексті даної системи, реалізація інтерфейсу RentalCalculator та метод calculateTotalCost є прикладом використання цього патерну (Рис. 5.7).

```
1   public interface RentalCalculator {
2       double calculateTotalCost(Rental rental, UUID discountId);
3   }
```

Рис. 5.7 - Інтерфейс для реалізації калькулятора оренди

Інтерфейс RentalService оголошує сигнатури методів, що включають створення, оновлення, зміну статусу та обчислення вартості оренди (Рис. 5.8):

```java
public interface RentalService {
    Rental get(UUID id);
    Rental create(RentalRequest request);
    Rental update(UUID id, RentalRequest request);
    Rental changeStatus(UUID id, ERentalStatus status);
    List<Rental> getAllPersonal();
    List<Rental> getAllByStatus(ERentalStatus status);
    List<Rental> getAll();
}
```

Рис. 5.8 - Інтерфейс для реалізації CRUD операцій оренди

У реалізації цього інтерфейсу (RentalServiceImpl) вбудовано логіку обробки оренди, зокрема — реалізацію методу обчислення загальної вартості оренди (Рис. 5.9):

```java
@Service
@AllArgsConstructor
public class RentalServiceImpl implements RentalService, RentalCalculator {

    private final RentalRepository rentalRepository;
    private final VehicleService vehicleService;
    private final AdditionalServiceService additionalServiceService;
    private final DiscountService discountService;
    private final UserService userService;
    private final ModelMapper modelMapper;

    @Override
    @Transactional
    public Rental create(RentalRequest request) {
        var user = userService.getCurrent();
        var vehicle = vehicleService.get(request.getVehicleId());
        vehicle.setStatus(EVehicleStatus.HIDDEN);
        vehicleService.update(vehicle.getId(), modelMapper.map(vehicle, VehicleDto.class));
        var additionalServices = additionalServiceService.getByIds(request.getAdditionalServicesIds());
        var rental = Rental.builder()
                .user(user)
                .status(ERentalStatus.AWAITING_CONFIRMATION)
                .vehicle(vehicle)
                .additionalServices(additionalServices)
                .startDate(request.getStartDate() != null ? request.getStartDate() : LocalDateTime.now())
                .endDate(request.getEndDate())
                .build();
        rental.setTotal(
                calculateTotalCost(rental, request.getDiscountId())
        );
        return rentalRepository.save(rental);
    }

    @Override
    public double calculateTotalCost(Rental rental, UUID discountId) {
        var priceOfVehiclePerDay = rental.getVehicle().getPrice();

        double priceOfEntirePeriodOfTime = priceCalculationInTimeInterval(
                priceOfVehiclePerDay,
                rental.getStartDate(),
                rental.getEndDate()
        );

        var priceOfAllAdditionalServices = rental.getAdditionalServices()
                .stream()
                .mapToDouble(AdditionalService::getPrice)
                .sum();

        double totalPriceBeforeDiscounts = priceOfEntirePeriodOfTime + priceOfAllAdditionalServices;
        double discount = discountId != null
                ? totalPriceBeforeDiscounts * (discountService.get(discountId).getAmount() / 100)
                : 0;

        return totalPriceBeforeDiscounts - discount;
    }

    private double priceCalculationInTimeInterval(double priceOfVehiclePerDay,
                                                  LocalDateTime startDate,
                                                  LocalDateTime endDate) {
        long days = ChronoUnit.DAYS.between(startDate.toLocalDate(), endDate.toLocalDate());
        return days * priceOfVehiclePerDay;
    }
}
```

Рис. 5.9 - Реалізація CRUD операцій для оренди

5.7 Використання REST Api

У процесі розробки було реалізовано REST API, яке забезпечує взаємодію між frontend- та backend-частинами системи. Для тестування та демонстрації роботи API використовувався інструмент **Postman**, що дозволяє виконувати HTTP-запити, передавати токени авторизації (Рис. 5.10), переглядати відповіді від сервера (Рис. 5.12) та відлагоджувати запити на етапі розробки (Рис. 5.11).

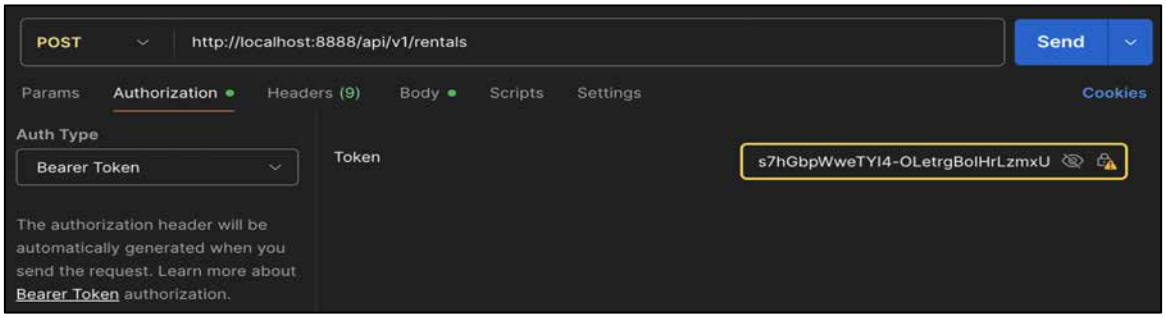

Рис. 5.10 - Передача токену

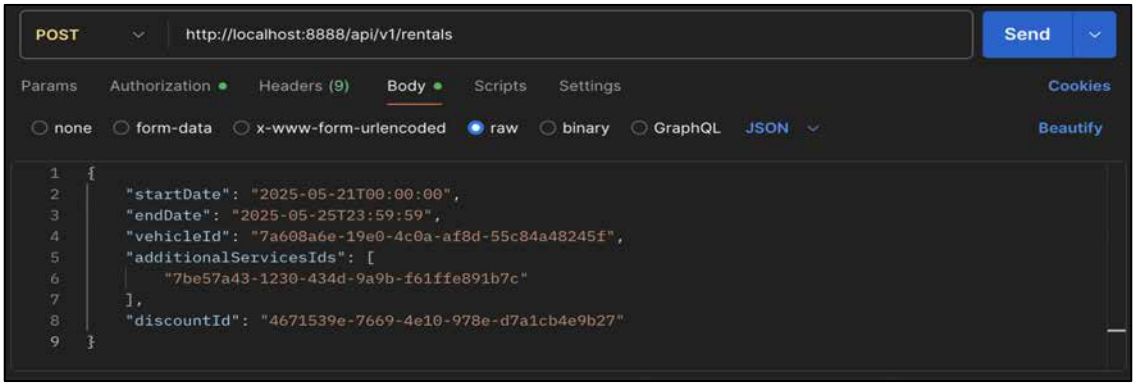

Рис. 5.11 - Тіло запиту

```
{
  "data": {
    "startDate": "2025-05-21T00:00:00",
    "endDate": "2025-05-25T23:59:59",
    "vehicle": {
      "id": "7a608a6e-19e0-4c0a-af8d-55c84a48245f",
      "name": "Superb",
      "price": 20.0,
      "status": "HIDDEN",
      "color": "GRAY",
      "bodyType": "LIFTBACK",
      "drivetrainType": "FWD",
      "technicalCondition": "GOOD",
      "transmissionType": "AUTOMATIC",
      "years": 2015,
      "licensePlate": "AI1234PO",
      "vinCode": "1HGCM82633A004352",
      "mileage": 120000,
      "maxSpeed": 220,
      "description": "",
      "manufacturerId": "9fe5b465-3dd5-42fc-a303-593154ac4720",
      "manufacturer": {
        "id": "9fe5b465-3dd5-42fc-a303-593154ac4720",
        "createdAt": "2025-05-18 16:39:53",
        "updatedAt": "2025-05-18 16:39:53",
        "name": "Skoda",
        "country": "Czech Republic"
      },
      "engineId": "bad4842f-e379-452b-b3eb-bd7e549c35eb",
      "engine": {
        "id": "bad4842f-e379-452b-b3eb-bd7e549c35eb",
        "name": "AT",
        "fuelType": "GASOLINE",
        "capacity": 1.5,
        "power": 150
      }
    },
    "total": 90.0,
    "additionalServices": [
      {
        "id": "7be57a43-1230-434d-9a9b-f61ffe891b7c",
        "name": "Дитяче крісло",
        "price": 20.0,
        "description": ""
      }
    ]
  },
  "success": true
}
```


Рис. 5.12 - Відповідь від REST API

Використання Postman дозволило:

- Тестувати коректність реалізації API без необхідності запуску клієнтської частини;
- Швидко і зручно перевіряти логіку авторизації та обробку запитів;
- Аналізувати отримані відповіді від сервера.

Цей етап був важливим для перевірки надійності API, валідації даних, коректного використання HTTP-методів (GET, POST, PUT, PATCH) і статус-кодів (200 OK, 201 Created, 400 Bad Request, 401 Unauthorized тощо).

5.8 Приклад реалізації на frontend частині

Фронтенд-частина системи розроблена з використанням **React** — сучасної бібліотеки для створення динамічних користувацьких інтерфейсів. Для оформлення інтерфейсу застосовано **Material UI (MUI)** — популярну бібліотеку компонентів, що реалізує дизайн-систему Material Design.

Одним із ключових елементів інтерфейсу є сторінка перегляду автомобіля та оформлення оренди (Рис. 5.13). Вона реалізована як компонент CarInfoPage, який відповідає за:

- відображення інформації про транспортний засіб (Рис. 5.14);
- вивід галереї зображень автомобіля (через компонент Carousel);
- вибір додаткових послуг;
- вибір дати початку та завершення оренди;
- застосування персональних знижок (якщо доступні);
- розрахунок повної вартості оренди (Рис. 5.15);
- створення запиту на оренду через REST API.

```
const handleContinue = async () => {
    const startDateTime = startDate ? `${startDate}T00:00:00` : '';
    const endDateTime = endDate ? `${endDate}T23:59:59` : '';
    try {
        await axios.post(
            `${API_BASE_URL}/rentals`,
            {
                startDate: startDateTime,
                endDate: endDateTime,
                vehicleId: vehicle.id,
                additionalServicesIds: selectedServices,
                discountId: selectedDiscount || null
            },
            {
                headers: {
                    Authorization: `Bearer ${authState.token}`
                }
            }
        );
        navigate('/account');
    } catch (error) {
        alert('Failed to create rental');
    }
};
```

Рис. 5.13 - Запит на створення оренди від клієнтської частини

Рис. 5.14 - Запит на створення оренди від клієнтської частини

Рис. 5.15 - Вибір додаткових послуг, застосування знижки та розрахунок повної вартості оренди

5.9 Висновок до розділу 5

У розділі здійснено реалізацію функціональних можливостей системи. Розроблено API-ендпоінти, реалізовано основну логіку бізнес-процесів, налаштовано валідацію даних та обробку помилок. Проведено тестування з використанням Postman, що підтвердило стабільність і відповідність системи поставленим вимогам.

6 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

6.1 Діаграма розгортання

Для забезпечення масштабованості, ізоляції сервісів і зручності розгортання програмного забезпечення була спроєктована архітектура клієнт-серверної вебсистеми, яка реалізована з використанням контейнерів Docker. Структура взаємодії між компонентами представлена на діаграмі розгортання (Рис. 6.1).

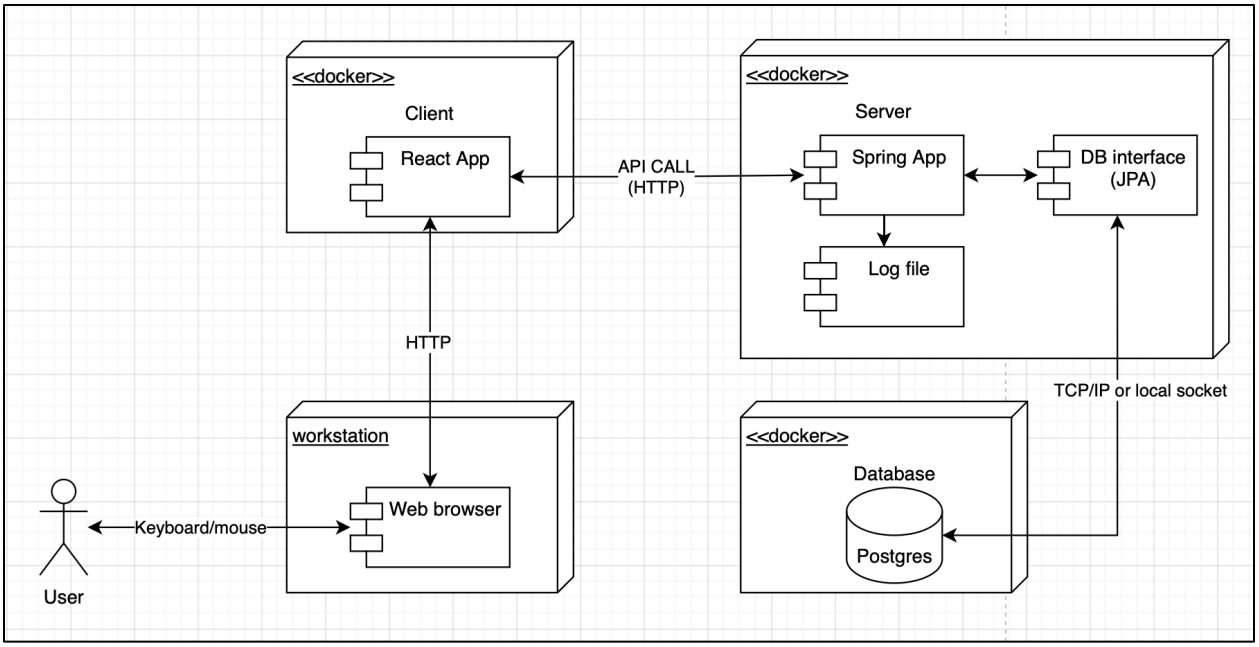

Рис.6.1 – приклад діаграма розгортання

Архітектура системи включає такі основні компоненти:

1. Користувач (User). Кінцевий користувач взаємодіє із застосунком за допомогою клавіатури та миші через веббраузер, встановлений на персональній робочій станції.

2. Робоча станція (workstation)
Містить веббраузер, через який здійснюється доступ до клієнтської частини системи. Запит на клієнт надсилається за протоколом HTTP.

3. Клієнтський контейнер (Client)
Запущений у середовищі Docker та містить фронтенд-застосунок, реалізований на React. Цей компонент відповідає за відображення інтерфейсу користувача та взаємодіє з серверною частиною через API-запити.

4. Серверний контейнер (Server)
Також ізольований у Docker-контейнері. Основним компонентом є застосунок, реалізований з використанням Spring Boot. Він відповідає за обробку бізнес-логіки, авторизацію, валідацію запитів, логування та доступ до бази даних через інтерфейс JPA (Java Persistence API).

5. Інтерфейс до бази даних (DB Interface / JPA)
Вбудований у Spring-застосунок інтерфейс, який забезпечує зручний і безпечний доступ до реляційної бази даних. Взаємодія з PostgreSQL здійснюється за допомогою SQL-запитів.

6. Система зберігання логів (Log file)
Призначена для запису логів серверної частини. Це можуть бути як текстові лог-файли всередині контейнера, так і зовнішні сервіси логування.

7. Контейнер бази даних (Database)
Містить інстанс PostgreSQL. Доступ до бази даних здійснюється по TCP/IP або через локальний сокет. У контейнері зберігаються всі дані, необхідні для функціонування системи.

6.2 Вимоги до апаратного та програмного забезпечення

Нижче наведено апаратні вимоги, які поділяються на мінімальні (для локального тестування та розробки) та рекомендовані (для розгортання серверної частини в продуктивному середовищі або на хмарній платформі) (Табл. 6.1).

Таблиця 6.1 – Вимоги до програмного забезпечення

Параметр	Мінімальні вимоги	Рекомендовані для сервера
Процесор (CPU)	4 ядра (Intel Core i5 або аналог)	4 ядра і більше
Оперативна пам’ять (RAM)	Від 8 ГБ	8–16 ГБ
Жорсткий диск (HDD/SSD)	Вільне місце: не менше 20 ГБ	SSD, обсяг: від 50 ГБ
Мережеве підключення	Обов’язкове для клієнт-серверної взаємодії	Стабільне інтернет-з’єднання

Операційна система	-	Linux (Ubuntu Server 20.04 LTS або інша)

Таким чином, навіть базовий комп’ютер середнього рівня здатен забезпечити ефективну розробку та тестування, у той час як для стабільного функціонування у виробничому середовищі потрібна більш потужна інфраструктура.

Програмна частина включає кілька незалежних компонентів: сервер, клієнт, база даних та супутні інструменти. Для кожного з них існують власні вимоги до середовища виконання. Нижче подано перелік необхідного програмного забезпечення для повноцінного функціонування проєкту (Табл. 6.2).

Таблиця 6.2 – Перелік неодхідного програмного забезперечення, для повноцінного функціонування проєкту

Категорія	Компонент	Версія / Примітка
Операційна система	Windows 10/11, Linux, macOS	Для розробки
	Ubuntu 20.04 LTS	Для розгортання сервера
Серверна	Java Development Kit	Версія 21

частина	(JDK)	
	Spring Boot Framework	Версія 3.x
	PostgreSQL	Версія 14 або вище
	Apache Maven	Для збірки проєкту
Клієнтська частина	Node.js	Версія 18.x або вище
	npm	Менеджер пакетів
	React.js	Версія 18
Інші інструменти	Git	Система контролю версій
	Docker	Опціонально, для контейнеризації
	IntelliJ IDEA / VS Code	Середовище розробки

	Postman	Для тестування API
	pgAdmin	Для керування базою даних PostgreSQL

Усі вищевказані інструменти є поширеними у професійному середовищі та мають відкритий вихідний код або безкоштовні версії для некомерційного використання, що значно спрощує їх використання у навчальних і дипломних проєктах.

Наявність чітко визначених апаратних та програмних вимог дозволяє забезпечити стабільну роботу як у процесі розробки, так і після розгортання програмного забезпечення. Використання перевірених технологій, таких як Java Spring, React.js та PostgreSQL, забезпечує гнучкість, масштабованість та безпеку системи. Завдяки застосуванню відкритих або умовно безкоштовних рішень, досягається оптимальний баланс між функціональністю та економічною доцільністю.

6.3 Вимоги до апаратного та програмного забезпечення

Для забезпечення надійної, безпечної та стабільної експлуатації програмного забезпечення, створеного в рамках даної дипломної роботи, важливо організувати відповідне технічне та програмне середовище, здійснювати регулярне обслуговування системи, а також передбачити засоби її масштабування і резервного відновлення. З огляду на використання

сучасних технологій, таких як Java Spring Boot, React.js та PostgreSQL, варто дотримуватись наступних рекомендацій.

Передусім, під час початкового розгортання необхідно ретельно налаштувати серверне оточення, включаючи встановлення відповідних версій JDK (Java Development Kit 21), системи управління залежностями Maven, а також бази даних PostgreSQL. Конфігураційні файли повинні містити актуальні значення параметрів з'єднання з БД, налаштування портів, шифрування та логування. Клієнтська частина розробляється з використанням React та збирається за допомогою Node.js, після чого може бути розміщена на окремому вебсервері або інтегрована в структуру Spring-додатку як статичні ресурси.

Оскільки система має клієнт-серверну архітектуру, доцільним є використання контейнеризації, що спрощує процес налаштування та експлуатації. Платформа Docker дозволяє створити ізольоване середовище для кожного компонента (бекенд, фронтенд, БД) з урахуванням усіх залежностей та налаштувань. Це особливо корисно при розгортанні проєкту на хмарних платформах або у випадках, коли необхідно швидко масштабувати інфраструктуру.

З метою поліпшення організації експлуатації системи рекомендовано дотримуватись таких принципів:

- Контейнеризація з Docker: розгортання додатку у вигляді контейнерів з використанням Dockerfile для кожного модуля та docker-compose для оркестрації компонентів. Це забезпечує гнучкість, зменшує залежність від ОС та спрощує розгортання на інших машинах.
- Моніторинг і логування: використання систем логування Spring Boot, а також додаткових засобів моніторингу (Prometheus,

Grafana) для спостереження за навантаженням, помилками та стабільністю роботи.

- Безпечне середовище: впровадження автентифікації користувачів, захисту API, використання HTTPS у продуктивному середовищі та обмеження доступу до БД і сервісів ззовні.
- Резервне копіювання: регулярне створення бекапів бази даних PostgreSQL із зберіганням на захищених носіях. Необхідно також періодично перевіряти працездатність процесу відновлення даних.
- Оновлення та супровід: планове оновлення залежностей (Java, бібліотек Spring, Node.js, React), підтримка структури проєкту у системі контролю версій (Git), ведення журналу змін (changelog) для кожного релізу.
- Документація і автоматизація: створення технічної документації з описом архітектури та інструкціями з розгортання; автоматизація процесів за допомогою скриптів або CI/CD (опційно).

Загалом, розроблене програмне забезпечення не вимагає надмірних ресурсів, однак забезпечення коректного налаштування середовища та дотримання описаних рекомендацій дозволяє досягти високої надійності, масштабованості та безпеки системи у реальних умовах експлуатації. Також варто передбачити процедури супроводу на випадок передачі проєкту іншому розробнику або команді.

Особливістю даного проєкту є його відкритість та доступність — усі вихідні коди, а також інструкції з розгортання, зберігаються у публічному репозиторії GitHub. Це дозволяє легко ознайомитись із внутрішньою

структурою системи, використовувати її як базу для подальших розробок або ж інтегрувати у більші програмні комплекси. Посилання на репозиторій:

6.4 Висновок до розділу 5

Систему підготовлено до розгортання у продуктивному середовищі. Реалізовано рекомендації щодо розгортання додатку, визначено вимоги до апаратного забезпечення. Побудована архітектура забезпечила гнучкість, надійність та готовність системи до використання.

ВИСНОВКИ

У процесі виконання дипломної роботи було реалізовано повний цикл розробки веб-застосунку для оренди транспортних засобів. Проведено аналіз актуальності проблеми, визначено основні функціональні вимоги та проведено порівняння з існуючими комерційними рішеннями. На основі цього було сформульовано технічне завдання та обрано оптимальні інструменти й технології для реалізації системи.

Під час роботи спроєктовано архітектуру системи, створено логічну модель бази даних, побудовано діаграми прецедентів, класів і діяльності. Розроблено клієнтську частину на основі React з використанням бібліотеки Material UI, а також серверну частину на базі Spring Boot із застосуванням PostgreSQL, JWT-аутентифікації та REST API. Забезпечено належний рівень безпеки, масштабованості та зручності у використанні.

Система протестована й підготовлена до впровадження. Здійснено підготовку розгортання та супровід. У результаті розроблено надійний,

функціональний та сучасний веб-застосунок, який здатен задовольнити потреби як кінцевих користувачів, так і бізнесу.

Поставлені завдання були повністю виконані, а мета дипломного проєкту — досягнута. Отримані результати можуть бути використані як основа для подальшого розвитку продукту та його впровадження в реальне середовище.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. – Prentice Hall, 2017. – 432 p.
2. Fowler, M. *Patterns of Enterprise Application Architecture*. – Addison-Wesley, 2002. – 560 p.
3. Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. – Addison-Wesley, 1994. – 395 p.
4. What is a RESTful API? [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/what-is/restful-api – Дата звернення: 15.04.2025.
5. React Documentation. [Електронний ресурс]. – Режим доступу: https://reactjs.org/docs/getting-started.html – Дата звернення: 10.04.2025.
6. Spring Boot Reference Documentation. [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ – Дата звернення: 12.04.2025.
7. PostgreSQL Official Documentation. [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/– Дата звернення: 13.04.2025.
8. Docker Documentation. [Електронний ресурс]. – Режим доступу: https://docs.docker.com/ – Дата звернення: 14.04.2025.

9. OpenAPI Specification (Swagger). [Електронний ресурс]. – Режим доступу: https://swagger.io/specification/ – Дата звернення: 15.04.2025.

10. Hibernate ORM User Guide. [Електронний ресурс]. – Режим доступу: https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html – Дата звернення: 14.04.2025.

11. Material UI Documentation. [Електронний ресурс]. – Режим доступу: https://mui.com/material-ui/getting-started/overview/ – Дата звернення: 11.04.2025.

Додаток А

GitHub репозиторій. [Електронний ресурс]. – Режим доступу: https://github.com/dmtryii/diploma-vehicle-rental-system