МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Дослідження засобів фреймворку Spring для розробки Веб-застосунків на платформі Java**

Виконав: студент 2 курсу, групи 8.1214-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Є. Т.Смирнов
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., доцент Г.П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дісітел»
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра __електроніки, інформаційних систем та програмного__ забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри _____Т.В. Критська_
“_01_” __квітня_____2025_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Смирнов Єгор Теімуразович
(прізвище, ім’я, по батькові)

1.Тема роботи Дослідження засобів фреймворку Spring для розробки Веб-застосунків на платформі Java

керівник роботи ___к.ф.-м.н., доцент Коломоєць Геннадій Павлович_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _28.03. 2025 р. № 361-е__
2. Строк подання студентом кваліфікаційної роботи_____01.12.2025 р.___
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- аналіз можливостей фреймворку Spring;
- проєктування архітектури веб-застосунку з використанням Spring Boot та MVC;
- реалізація веб-застосунку і опис його функціоналу;
- визначення вимог до програмного забезпечення для роботи застосунку;
- аналіз результатів розробки та формування висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
_____слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.04.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником.	01.04.2025	виконано
2	Літературний огляд. Перший розділ. Екосистема Spring та коротка характеристика її проєктів. Принципи інверсії управління та ін'єкції залежностей та їх використання Spring. Поняття бінів. Контекст застосунку та способи додання бінів до контексту. Поняття аспектів та їх використання Spring.	31.03 – 12.04.25	виконано
3	Літературний огляд. Другий розділ. Засоби Spring Framework для розробки Веб-застосунків. Spring Boot та його можливості. Визначення функціоналу застосунку, що розроблятиметься з використанням засобів Spring. Проєктування бази даних та створення Maven-проєкту застосунку.	01.05 – 20.05.25	виконано
4	Третій розділ. Проєктування застосунку із використанням Spring Boot, Spring MVC, Spring Data JPA та Thymeleaf.	25.05 – 20.06.25	виконано
5	Конструювання застосунку із використанням Spring Boot, Spring MVC, Spring Data JPA та Thymeleaf.	21.06 – 30.11.2025	виконано
6	Представлення практичних результатів дослідження науковому керівнику.	01.12.25	виконано
10	Оформлення звіту.	10.12.25	виконано

Студент _____ Смирнов Є. Т.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Коломоєць Г. П.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А,__

АНОТАЦІЯ

Сторінок: 101

Рисунків: 9

Таблиць: 6

Джерел: 21

Смирнов Є. Т. Веб-застосунок для оренди автомобілів із використанням фреймворку Spring : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Коломоєць Г. П. Запоріжжя : ЗНУ, 2025. 101 с.

Мета і завдання дослідження полягають у дослідженні засобів фреймворку Spring для розробки веб-застосунків на платформі Java, а також у проєктуванні та реалізації веб-застосунку для оренди автомобілів із використанням сучасних підходів до архітектури, безпеки та роботи з базами даних.

У процесі дослідження було проаналізовано сучасні підходи до веб-розробки на платформі Java, можливості екосистеми Spring та її основних модулів. На основі проведеного аналізу спроєктовано архітектуру веб-застосунку з використанням шаблону Model–View–Controller.

У результаті виконання роботи було реалізовано повнофункціональний веб-застосунок для оренди автомобілів із підтримкою реєстрації та авторизації користувачів, керування автопарком, бронювання транспортних засобів та розмежування доступу на основі ролей. Розроблений застосунок використовує Spring Boot, Spring MVC, Spring Data JPA, Spring Security та Thymeleaf.

Ключові слова: веб-застосунок, Java, Spring Boot, Spring MVC, Spring Data JPA, Spring Security, Thymeleaf, оренда автомобілів, MVC.

SUMMARY

Pages: 101

Figures: 9

Tables: 6

Sources: 21

Smyrnov Ye. T. Web application for car rental using the Spring framework : qualification work of the master of specialty 121 “Software Engineering” / Scientific supervisor H. P. Kolomoiets. Zaporizhzhia : ZNU, 2025. 101 p.

The aim of the research is to study the tools of the Spring framework for developing web applications on the Java platform, as well as to design and implement a web application for car rental using modern approaches to architecture, security, and database interaction.

In the course of the research, modern approaches to web development on the Java platform were analyzed, along with the capabilities of the Spring ecosystem and its core modules. Based on the conducted analysis, the architecture of a web application was designed using the Model–View–Controller (MVC) pattern.

As a result of the research, a fully functional web application for car rental was developed. The application supports user registration and authentication, car fleet management, vehicle booking, and role-based access control. The developed application is based on Spring Boot, Spring MVC, Spring Data JPA, Spring Security, and Thymeleaf and can be used as a basis for further extension and scaling.

Keywords: web application, Java, Spring Boot, Spring MVC, Spring Data JPA, Spring Security, Thymeleaf, car rental, MVC.

ЗМІСТ

ВСТУП 10

ВЕБ-РОЗРОБЦІ 15

1.1 Загальна характеристика фреймворку Spring 15

1.2 Інверсія управління та ін’єкція залежностей у Spring 16

1.3 Контекст застосунку та способи додавання бінів у Spring 20

1.4 Аспектно-орієнтоване програмування в Spring (AOP) 23

1.5 Spring Boot спрощення конфігурації 25

1.6 Висновки до розділу 1 27

РОЗДІЛ 2 ЛІТЕРАТУРНИЙ ОГЛЯД. ЗАСОБИ SPRING ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ 29

2.1 Аналіз сучасних підходів до веб-розробки на Java 29

2.2 Spring Framework як основа створення веб-застосунків 30

2.3 Spring Boot та його можливості в контексті веб-розробки 31

2.4 Визначення функціоналу застосунку для оренди автомобілів 32

2.5 Проєктування бази даних для веб-застосунку 33

2.5.1 Основні сутності та їх призначення 34

2.5.2 Детальний опис таблиць та їхніх полів 34

2.5.6 Опис зв’язків між таблицями 37

2.6 Створення Maven-проєкту засобами Spring Initializr 38

2.6.1 Вибір параметрів проєкту 39

2.6.2 Додавання необхідних залежностей 39

2.6.3 Формування файлу pom.xml 40

2.6.4 Структура згенерованого Spring Boot-проєкту 41
2.7 Висновки до розділу 2 42
РОЗДІЛ 3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ІЗ ВИКОРИСТАННЯМ SPRING BOOT, SPRING MVC, SPRING DATA JPA ТА THYMELEAF 44
3.1 Загальні принципи архітектури веб-застосунку 44
3.3 Архітектурне проєктування застосунку 47
3.3.1 Рівень даних (Model Layer) 48
3.3.2 Репозиторний рівень (Repository Layer) 54
3.3.3 Сервісний рівень (Service Layer) 57
3.3.4 Контролерний рівень (Controller Layer) 61
3.3.5 Рівень представлення (View Layer) Thymeleaf 67
3.4 UML-діаграма доменної моделі веб-застосунку 69
3.5 Реалізація взаємодії між компонентами 71
3.6 Представлення даних користувачу 71
3.7 Безпека та авторизація користувачів 72
3.8. Використання шаблонізатора Thymeleaf у веб-застосунку 72
3.8.1 Загальна характеристика Thymeleaf 76
3.8.2 Інтеграція Thymeleaf зі Spring Boot 77
3.8.3 Передача даних з контролера до шаблону 77
3.8.4 Основні атрибути Thymeleaf 78
3.8.5 Робота з формами 78
3.8.6 Використання фрагментів і шаблонів (layout) 79
3.8.7 Валідація даних у шаблонах 79

3.8.8 Переваги використання Thymeleaf у Spring Boot 80

3.9 Реалізація безпеки та авторизації користувачів у веб-застосунку засобами Spring Security 81

3.9.1 Модель безпеки та ролей 81

3.9.2 Реалізація UserDetailsService 82

3.9.3 Шифрування паролів 83

3.9.4 Конфігурація Spring Security 83

3.9.5 Захист адміністративних сторінок 84

3.9.6 Сторінка входу та виходу 84

3.9.7 Обробка помилок авторизації 85

3.9.8 Переваги використання Spring Security 85

3.10 Висновки до розділу 3 86

4.1 Загальні підходи до конструювання веб-застосунку 87

4.2 Створення та конфігурація Spring Boot-проєкту 88

4.3 Реалізація доменної моделі 89

4.3.1 Принципи створення JPA-сутностей 89

4.3.2 Пояснення ключових сутностей 90

4.4 Реалізація рівня доступу до даних (Repository Layer) 90

4.5 Реалізація бізнес-логіки (Service Layer) 91

4.6 Реалізація контролерів Spring MVC 93

4.8 Реалізація авторизації та автентифікації Spring Security 94

4.9 Ініціалізація тестових даних 95

4.10 Висновки до 4 розділу 96

ВИСНОВКИ.....98

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....100

ВСТУП

Актуальність теми

Розвиток інформаційних технологій та цифровізація бізнес-процесів створюють нові можливості для підвищення ефективності та зручності повсякденного життя. Однією з таких актуальних задач є створення надійних і зручних у використанні веб-застосунків для автоматизації процесів оренди автомобілів. Платформа Java, зокрема, фреймворк Spring, є потужним інструментом для вирішення таких завдань завдяки своїй масштабованості, продуктивності та широкій функціональності [1].

Веб-застосунки на базі Spring Framework дозволяють створювати інтуїтивно зрозумілі інтерфейси та забезпечують надійність роботи системи, що є особливо важливим для сервісів, які працюють з великими потоками користувачів та даними. Сучасний ринок автомобільної оренди потребує гнучких, масштабованих та ефективних IT-рішень, які здатні забезпечити комфортну взаємодію як клієнтам, так і адміністраторам сервісу.

Таким чином, актуальним є дослідження можливостей фреймворку Spring для розробки веб-застосунків, що надають користувачам зручний сервіс оренди автомобілів з можливістю перегляду, бронювання, зміни або скасування бронювань, а адміністраторам – зручні інструменти для управління парком транспортних засобів.

Мета і завдання дослідження

Метою дослідження є аналіз і вивчення сучасних засобів Spring Framework для створення веб-застосунків, а також розробка повнофункціонального веб-майданчика для оренди автомобілів.

Для досягнення мети необхідно вирішити такі завдання:

1. Провести аналіз основних компонентів екосистеми Spring та їх можливостей.

2. Вивчити особливості та переваги Spring Boot для розробки веб-застосунків.

3. Розробити веб-застосунок, що дозволяє клієнтам переглядати, бронювати, змінювати або скасовувати бронювання автомобілів, а адміністраторам керувати автопарком.

Об'єкт дослідження

Об'єктом дослідження є процеси веб-розробки на платформі Java з використанням фреймворку Spring.

Предмет дослідження

Предметом дослідження є методи та інструменти Spring Framework, що використовуються для реалізації функціоналу веб-застосунків.

Методи дослідження

В роботі використовуються наступні методи дослідження:

1. Аналіз наукової та технічної літератури, а також офіційної документації фреймворку Spring.

2. Порівняльний аналіз існуючих технологічних рішень для розробки веб-застосунків.

3. Практичне експериментування шляхом створення реального веб-застосунку.

4. Синтез та узагальнення отриманих результатів.

Наукова новизна одержаних результатів

Наукова новизна роботи полягає у комплексному дослідженні засобів Spring Framework з акцентом на практичне впровадження при створенні сучасних веб-сервісів для оренди автомобілів, а також визначення оптимальних підходів до реалізації такого типу веб-застосунків.

Практичне значення одержаних результатів

Практичне значення дослідження полягає у створенні повноцінного веб-застосунку, що може бути впроваджений у реальну діяльність компаній, які

займаються орендою автомобілів. Розроблений веб-застосунок може слугувати зразком для реалізації аналогічних рішень, а також навчальним матеріалом для студентів та спеціалістів у сфері веб-розробки.

Апробація одержаних результатів

Результати дослідження були представлені на студентській науково-практичній конференції «Молода наука – 2025», де отримали позитивні відгуки [2].

Глосарій

Веб-застосунок (англ. Web application) — програмний продукт, що працює на веб-сервері та доступний користувачам через браузер за допомогою мережі Інтернет.

Платформа Java — програмна платформа та мова програмування, призначена для розробки кросплатформних застосунків, що виконуються у середовищі Java Virtual Machine.

Фреймворк Spring — комплексна платформа для розробки корпоративних та веб-застосунків на мові Java, що забезпечує інверсію управління, модульність та масштабованість програмних рішень.

Spring Boot — модуль екосистеми Spring, який спрощує створення та запуск веб-застосунків за рахунок автоматичної конфігурації та вбудованого сервера.

Spring MVC (англ. Model–View–Controller) — модуль Spring для реалізації веб-застосунків на основі архітектурного шаблону MVC, що забезпечує розділення бізнес-логіки, представлення та керування запитами.

Model–View–Controller (MVC) — архітектурний шаблон, що передбачає розподіл застосунку на модель даних, представлення та контролер, які взаємодіють між собою.

Spring Data JPA — модуль Spring, що спрощує роботу з реляційними базами даних шляхом використання об’єктно-реляційного відображення та автоматичної генерації SQL-запитів.

JPA (англ. Java Persistence API) — специфікація Java для роботи з реляційними базами даних за допомогою ORM-підходу.

ORM (англ. Object-Relational Mapping) — технологія відображення об’єктів предметної області у таблиці реляційної бази даних.

Сутність (англ. Entity) — клас доменної моделі, який відповідає таблиці бази даних та описується за допомогою JPA-анотацій.

Репозиторій (англ. Repository) — компонент застосунку, відповідальний за доступ до даних та виконання операцій збереження, пошуку й видалення об’єктів у базі даних.

Сервісний шар (англ. Service layer) — рівень архітектури, що містить бізнес-логіку застосунку та координує взаємодію між контролерами і репозиторіями.

Контролер (англ. Controller) — компонент Spring MVC, який обробляє HTTP-запити, викликає відповідні сервіси та передає дані до представлення.

Thymeleaf — серверний шаблонізатор для формування HTML-сторінок у веб-застосунках Spring з можливістю динамічного відображення даних.

Spring Security — модуль Spring, призначений для реалізації автентифікації, авторизації та захисту веб-застосунків.

Автентифікація (англ. Authentication) — процес перевірки особи користувача за допомогою облікових даних.

Авторизація (англ. Authorization) — процес визначення прав доступу користувача до ресурсів застосунку на основі його ролі.

Роль користувача (англ. User role) — набір дозволів, що визначає рівень доступу користувача до функціоналу веб-застосунку.

Реляційна база даних — база даних, у якій інформація зберігається у вигляді таблиць, пов’язаних між собою.

MySQL — реляційна система керування базами даних з відкритим вихідним кодом, що використовується для зберігання даних веб-застосунку.

HTTP-запит — звернення клієнта до веб-сервера для отримання або передавання даних.

РОЗДІЛ 1 ЕКОСИСТЕМА SPRING ТА ЇЇ ЗАСТОСУВАННЯ У ВЕБ-РОЗРОБЦІ

1.1 Загальна характеристика фреймворку Spring

Фреймворк Spring є одним з найпоширеніших інструментів для розробки програмного забезпечення на платформі Java, особливо у сфері корпоративної (enterprise) розробки. Його гнучка та модульна архітектура забезпечує можливість створення масштабованих, підтримуваних і ефективних застосунків [1].

Основна ідея фреймворку полягає у впровадженні принципів інверсії управління (IoC — Inversion of Control) та ін’єкції залежностей (DI — Dependency Injection), що дозволяє знизити зв’язність між компонентами системи та підвищити гнучкість архітектури. Такий підхід сприяє розробці тестованого, розширюваного та незалежного коду.

Екосистема Spring включає велику кількість модулів, серед яких основними є:

- **Spring Core** — базовий модуль, що забезпечує механізм ін’єкції залежностей та управління життєвим циклом об’єктів;
- **Spring Web (Spring MVC)** — модуль для створення веб-застосунків, підтримує REST-архітектуру та шаблон Model-View-Controller;
- **Spring Boot** — модуль для спрощення конфігурації та запуску Spring-застосунків, надає автоконфігурацію та вбудований сервер (Tomcat, Jetty) [2];
- **Spring Data** — засіб для доступу до баз даних із використанням репозиторного підходу;

- **Spring Security** — фреймворк для реалізації механізмів автентифікації, авторизації та захисту веб-застосунку;
- **Spring Cloud** — модуль для створення мікросервісної архітектури, взаємодії сервісів, обробки конфігурацій, реєстрації тощо;
- **Spring AOP** — реалізація аспектно-орієнтованого програмування, що дозволяє відокремити технічну логіку (логування, безпека) від бізнес-логіки;
- **Spring Test** — модуль для тестування компонентів Spring-застосунку.

Перевагами Spring є:

- підтримка різноманітних способів конфігурації (XML, Java-конфігурація, анотації);
- висока продуктивність та масштабованість;
- активна спільнота розробників і регулярні оновлення;
- підтримка найсучасніших підходів у розробці (DevOps, REST, Docker, Kubernetes);
- сумісність із популярними інструментами — Maven, Gradle, Git, Jenkins.

Завдяки широким можливостям, Spring став де-факто стандартом у розробці Java-застосунків та є рекомендованим вибором для реалізації комплексних систем, зокрема таких, як системи управління орендою автомобілів.

1.2 Інверсія управління та ін’єкція залежностей у Spring

Однією з ключових концепцій, яка лежить в основі архітектури Spring, є **інверсія управління** (Inversion of Control, IoC). У традиційних застосунках компоненти самостійно створюють свої залежності, що призводить до високої зв’язаності коду. Spring Framework змінює цей підхід: створення і керування залежностями делегується спеціальному контейнеру — **Spring IoC Container**.

Завдяки такій інверсії, залежності не створюються вручну, а "впроваджуються" (ін’єктуються) ззовні. Цей підхід називається **ін’єкцією залежностей** (Dependency Injection, DI), і він забезпечує низький рівень зв’язності, що спрощує тестування, розширення та супровід програмного забезпечення.

У Spring є кілька способів ін’єкції залежностей:

1. **Через конструктор**

   Це найбільш рекомендований підхід, оскільки забезпечує незмінність об’єктів (immutability) та чітко позначає, які саме залежності потрібні для роботи компонента. Приклад ін’єкції залежностей через конструктор наведено у лістингу 1.

   Лістинг 1 — Ін’єкція залежностей через конструктор

```
@Component
public class BookingService {
    private final CarRepository carRepository;

    @Autowired
    public BookingService(CarRepository carRepository) {
        this.carRepository = carRepository;
```

2. **Через сеттер**

   Використовується, якщо залежність не є обов’язковою або може змінюватися після створення об’єкта. Приклад використання сеттер-ін’єкції показано у лістингу 2.

   Лістинг 2 — Ін’єкція залежностей через сеттер

```
@Component
public class CarService {
    private CarRepository carRepository;
```

```
    @Autowired
    public void setCarRepository(CarRepository carRepository) {
        this.carRepository = carRepository;
    }
}
```

3. **Через поле**

   Найменш рекомендований підхід, але широко застосовується через компактність. Його мінус — ускладнення юніт-тестування. Приклад ін’єкції залежностей через поле наведено у лістингу 3.

Лістинг 3 — Ін’єкція залежностей через поле

```
@Component
public class RentalController {
    @Autowired
    private BookingService bookingService;
}
```

Контейнери IoC у Spring

IoC реалізується за допомогою **ApplicationContext** — центрального інтерфейсу для взаємодії з контейнером Spring. Існує кілька його реалізацій, зокрема:

- **AnnotationConfigApplicationContext** — використовується для Java-конфігурації;
- **ClassPathXmlApplicationContext** — для конфігурацій на базі XML;
- **WebApplicationContext** — специфічний для веб-застосунків.

Під час запуску застосунку контейнер Spring виконує сканування класів, позначених спеціальними анотаціями, створює об’єкти (біни), визначає їхні залежності та автоматично впроваджує їх у відповідні компоненти.

Анотації для управління біном

- @Component — позначає клас як компонент, який повинен бути автоматично зареєстрований у Spring-контейнері;
- @Service, @Repository, @Controller — спеціалізовані варіанти @Component, які мають додаткову семантику;
- @Autowired — вказує Spring, що залежність повинна бути впроваджена автоматично;
- @Qualifier — використовується для уточнення, який саме бін впроваджувати, якщо є кілька кандидатів.

Приклад роботи IoC наведено у лістингу 4.

Лістинг 4 — Приклад інверсії керування (IoC)

```
@Component
public class PaymentService {
    public void processPayment() {
        System.out.println("Processing payment...");
    }
}

@Service
public class BookingService {
    private final PaymentService paymentService;

    @Autowired
    public BookingService(PaymentService paymentService) {
        this.paymentService = paymentService;
    }
}
```

У цьому прикладі Spring автоматично створить об’єкти PaymentService та BookingService, після чого впровадить залежність PaymentService у BookingService.

1.3 Контекст застосунку та способи додавання бінів у Spring

У Spring Framework **контекст застосунку (ApplicationContext)** є центральним інтерфейсом, який відповідає за створення, конфігурацію та управління життєвим циклом об'єктів — **бінів** (beans). Біни — це звичайні Java-об’єкти, які контролюються Spring-контейнером та автоматично впроваджуються в інші компоненти при потребі.

Контекст застосунку також забезпечує такі функції:

- зберігання бінів і доступ до них за іменем або типом;
- автоматичне сканування класів і виявлення анотацій;
- підтримку подій, локалізації та життєвого циклу;
- взаємодію з навколишнім середовищем.

Основні типи контекстів застосунку:

- AnnotationConfigApplicationContext — використовується для Java-конфігурації (без XML);
- ClassPathXmlApplicationContext — для конфігурацій, що описані у XML-файлах;
- WebApplicationContext — специфічний для веб-застосунків, є розширенням ApplicationContext.

Реєстрація бінів: основні підходи

У Spring існує кілька способів реєстрації бінів у контейнері:

1. Анотаційна конфігурація (найпоширеніша)

Spring підтримує автоматичне сканування класів у вказаних пакетах та виявлення класів, позначених анотаціями:

- @Component — загальне позначення біну;
- @Service, @Repository, @Controller — спеціалізовані анотації.

Приклад створення компонента з використанням анотації @Component наведено у лістингу 5.

Лістинг 5 — Приклад автоматичного виявлення компонента Spring

```
@Component
public class NotificationService {
    public void send() {
        System.out.println("Notification sent.");
    }
}
```

2. Java-конфігурація через @Bean

Інший підхід — визначити метод, що повертає бін, і позначити його анотацією @Bean. Це зручно для сторонніх бібліотек, які не можна змінювати.

Приклад оголошення біну за допомогою анотації @Bean наведено у лістингу 6.

Лістинг 6— Створення біну за допомогою анотації @Bean

```
@Configuration
public class AppConfig {
    @Bean
    public CarService carService() {
        return new CarService();
    }
}
```

3. XML-конфігурація (застарілий підхід, але ще підтримується)

У ранніх версіях Spring використовували XML для опису бінів:

<bean id="carService" class="com.example.service.CarService"/>

Цей підхід нині вважається застарілим, поступаючись більш зручним Java-конфігураціям з використанням анотацій.

Життєвий цикл біну

Spring-контейнер управляє життєвим циклом бінів:

1. Створення об'єкта;
2. Впровадження залежностей;
3. Виклик методів ініціалізації;
4. Використання;
5. Виклик методів завершення (destroy).

Можна визначати кастомні методи ініціалізації та знищення за допомогою анотацій @PostConstruct та @PreDestroy, або конфігураційно через initMethod і destroyMethod. Приклад керування життєвим циклом біну за допомогою анотацій наведено у лістингу 7.

Лістинг 7 — Керування життєвим циклом біну за допомогою @PostConstruct та @PreDestroy

```
@Component
public class EngineService {

    @PostConstruct
    public void init() {
        System.out.println("Engine initialized.");
    }

    @PreDestroy
    public void shutdown() {
        System.out.println("Engine shut down.");
    }
}
```

Завдяки використанню контексту застосунку Spring значно спрощує керування залежностями, зменшує кількість шаблонного коду та забезпечує централізовану конфігурацію всіх компонентів системи.

1.4 Аспектно-орієнтоване програмування в Spring (AOP)

У складних веб-застосунках часто виникає потреба у повторюваній технічній логіці, яка розкидана по різних модулях системи: логування, аудит, безпека, обробка транзакцій тощо. Щоб уникнути дублювання коду та підвищити розділення відповідальностей, у Spring реалізовано підтримку **аспектно-орієнтованого програмування (AOP — Aspect-Oriented Programming)**.

AOP дозволяє виділити загальні функції в окремі модулі — **аспекти** — і застосовувати їх до основної бізнес-логіки без її зміни. Це забезпечує чистоту коду, спрощує тестування та підвищує повторне використання.

Основні поняття AOP:

- **Аспект (Aspect)** — модуль, що містить логіку, яка застосовується до різних частин програми (наприклад, логування);
- **Точка перетину (Join Point)** — конкретне місце в програмі, де може бути виконаний аспект (наприклад, виклик методу);
- **Поінткат (Pointcut)** — вираз, що описує, до яких точок перетину застосовується аспект;
- **Адвіс (Advice)** — код, який виконується у визначеній точці: до (@Before), після (@After) або навколо (@Around) виклику методу;
- **Веб-проксі (Proxy)** — механізм, що динамічно перехоплює виклики методів і вставляє аспектну логіку.

Приклад використання AOP у Spring

1. Додаємо залежність до spring-boot-starter-aop (якщо використовується Spring Boot), залежність для підключення AOP наведено у лістингу 8.

Лістинг 8 — Підключення модуля AOP у Spring Boot

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

2. Створюємо аспект, приклад реалізації аспекту для логування викликів методів сервісного рівня наведено у лістингу 9.

Лістинг 9 — Аспект логування викликів методів сервісного шару

```
@Component
public class LoggingAspect {

    @Before("execution(*
com.example.carrental.service.*.*(..))")
    public void logBeforeMethod(JoinPoint joinPoint) {
        System.out.println("Виконання методу: " +
joinPoint.getSignature().getName());
    }
}
```

У цьому прикладі:

- @Aspect позначає клас як аспект;
- @Before означає, що метод logBeforeMethod виконується перед кожним методом у пакеті service;
- execution(* ..) — поінткат, що описує, до яких методів застосовується адвіс.

Типи Advice у Spring:

- @Before — Виконується перед викликом методу
- @After — Виконується після виклику (незалежно від результату)
- @AfterReturning — Виконується після успішного завершення методу
- @AfterThrowing — Виконується, якщо метод завершився з винятком
- @Around — Виконується до і після виклику (можна контролювати виконання повністю)

Застосування AOP у веб-застосунках

У контексті системи оренди автомобілів AOP можна застосувати для:

- логування запитів до сервісів;
- вимірювання продуктивності методів;
- перевірки прав доступу (безпека);
- централізованої обробки помилок;
- аудиту дій користувачів (історія бронювання, вхід у систему тощо).

Аспектно-орієнтоване програмування у Spring дозволяє досягти високого рівня чистоти коду і централізованого керування нефункціональними вимогами, не порушуючи бізнес-логіку застосунку.

1.5 Spring Boot спрощення конфігурації

Spring Boot — це підпроєкт екосистеми Spring, основною метою якого є **максимальне спрощення створення та конфігурації Java-застосунків**. Завдяки Spring Boot розробники можуть зосередитись на бізнес-логіці, не витрачаючи час на налаштування серверів, залежностей та інших інфраструктурних компонентів [2].

Spring Boot надає:

- автоматичну конфігурацію;
- вбудовані веб-сервери (Tomcat, Jetty, Undertow);
- готові стартові шаблони (стартери);
- інтеграцію з базами даних, системами безпеки, логуванням тощо;
- зручне тестування та моніторинг.

Основні переваги Spring Boot

1. Мінімум конфігурацій — завдяки автоконфігурації (@SpringBootApplication) застосунок запускається без явного опису XML-конфігурацій.
2. Вбудований сервер — застосунок можна запустити як самостійний .jar файл, не встановлюючи окремий сервлет-контейнер.
3. Швидкий старт проєкту — за допомогою Spring Initializr можна створити базовий шаблон проєкту буквально за хвилини.
4. Структуроване управління залежностями — через стартери (spring-boot-starter-web, spring-boot-starter-data-jpa, spring-boot-starter-security тощо).
5. Гнучке налаштування через application.properties або application.yml.
6. Вбудована система моніторингу (Spring Boot Actuator).

Конфігурація через application.yml

```
server:
  port: 8080
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/rental_db
username: root
    password: password
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

Запуск застосунку

Після компіляції проєкту (наприклад, через Maven), можна запустити застосунок за допомогою команди:

java -jar target/car-rental-app.jar

Сервер стартує автоматично на порту 8080, і застосунок стає доступним через браузер або API-клієнт.

Практичне значення Spring Boot

У проєкті з оренди автомобілів Spring Boot дозволяє:

- швидко створити RESTful API для клієнтів і адміністраторів;
- зручно конфігурувати з'єднання з базою даних;
- реалізувати безпечну авторизацію через spring-boot-starter-security;
- легко масштабувати застосунок, розгортаючи його у хмарі, контейнеризуючи за допомогою Docker.

Spring Boot надає потужний інструментарій для швидкої та ефективної розробки веб-застосунків, роблячи його ідеальним вибором для сучасних бізнес-сервісів.

1.6 Висновки до розділу 1

У першому розділі було здійснено комплексний огляд екосистеми Spring та ключових концепцій, що лежать в основі розробки сучасних веб-застосунків на платформі Java.

Розглянуто фундаментальні принципи інверсії управління (IoC) та ін'єкції залежностей (DI), на яких базується робота Spring Container. Показано роль ApplicationContext як центрального механізму створення та управління бінів.

Окрему увагу приділено аспектно-орієнтованому програмуванню (AOP), що забезпечує відокремлення технічної логіки — логування, безпеки, обробки транзакцій — від бізнес-функціоналу.

Важливою частиною огляду став Spring Boot — інструмент, що значно спрощує конфігурацію, запуск і структурування Spring-застосунків завдяки автоконфігурації, вбудованим серверам та системі стартових залежностей.

Описані технології є фундаментом для створення веб-застосунку системи оренди автомобілів, що розробляється в межах дипломної роботи. У наступному розділі буде проведено літературний огляд засобів Spring для реалізації веб-функціоналу, проаналізовано можливості Spring MVC, Spring Data JPA, Spring Security, а також сформовано вимоги та модель даних майбутнього застосунку.

РОЗДІЛ 2 ЛІТЕРАТУРНИЙ ОГЛЯД. ЗАСОБИ SPRING ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

2.1 Аналіз сучасних підходів до веб-розробки на Java

Сучасна веб-розробка на платформі Java базується на використанні потужних фреймворків і бібліотек, які забезпечують високу продуктивність, безпеку та масштабованість застосунків. Традиційні підходи, що передбачали написання великої кількості конфігураційного коду, поступово замінюються автоматизованими рішеннями, орієнтованими на зручність розробника та швидке розгортання систем.

На сьогодні існує декілька основних технологічних підходів до створення веб-застосунків на Java:

- Jakarta EE — набір специфікацій для розробки корпоративних застосунків, що включає сервлети, JSP, EJB, JPA та інші технології. Хоча Jakarta EE залишається потужною платформою, її основним недоліком є складність конфігурації та громіздкість архітектури;
- Spring Framework — сучасна альтернатива Jakarta EE, що забезпечує легкість конфігурації, інверсію управління (IoC), а також підтримку модульності й тестованості. Spring активно інтегрується з іншими технологіями, такими як Hibernate, JPA, REST і SOAP;
- Мікросервісна архітектура — сучасний підхід до побудови веб-застосунків, який передбачає розподіл системи на невеликі незалежні сервіси. В екосистемі Java реалізація цього підходу здійснюється через Spring Boot і Spring Cloud;

- Reactive Web (Spring WebFlux) — реактивний підхід до обробки запитів, що дозволяє створювати високопродуктивні системи з асинхронною обробкою даних.

Завдяки поєднанню гнучкості та продуктивності, Spring Framework став де-факто стандартом у сучасній Java-розробці. Його універсальність, підтримка стандартів та висока сумісність з іншими бібліотеками зробили його основою для більшості корпоративних і комерційних веб-застосунків.

2.2 Spring Framework як основа створення веб-застосунків

Spring Framework — це комплексне середовище для розробки Java-застосунків, яке спрощує створення, тестування та розгортання програмного забезпечення. Основна концепція фреймворку полягає в реалізації принципів інверсії управління (Inversion of Control) та ін’єкції залежностей (Dependency Injection), що зменшує зв’язність між компонентами і забезпечує високу модульність системи.

Для розробки веб-застосунків Spring пропонує модуль Spring Web MVC, який реалізує шаблон проектування *Model–View–Controller*. Він розділяє логіку обробки запитів, бізнес-логіку та представлення, що дозволяє створювати чисту архітектуру й легко підтримувати код.

Основні компоненти Spring MVC:

- DispatcherServlet — центральний компонент, який приймає HTTP-запити та координує їх обробку;
- Controller — клас, що містить бізнес-логіку та керує обробкою запитів;
- Model — об’єктна модель, що містить дані для представлення;
- ViewResolver — компонент, який відповідає за вибір потрібного шаблону представлення (HTML, JSP, Thymeleaf тощо).

Крім MVC, екосистема Spring пропонує додаткові засоби:

- Spring Data JPA — для роботи з базами даних за допомогою ORM;
- Spring Security — для реалізації автентифікації та авторизації;
- Spring AOP — для аспектно-орієнтованого програмування;
- Spring Test — для автоматизованого тестування компонентів;

Spring дозволяє створювати як класичні веб-застосунки, так і RESTful-сервіси, що забезпечують взаємодію з клієнтськими інтерфейсами (наприклад, Angular, React, Vue) ;

Таким чином, Spring Framework є потужним інструментом для побудови сучасних веб-рішень із високими вимогами до гнучкості та надійності.

2.3 Spring Boot та його можливості в контексті веб-розробки

Spring Boot — це надбудова над Spring Framework, яка істотно спрощує процес створення та налаштування Java-застосунків. Його ключова мета — **усунути необхідність ручного конфігурування** та забезпечити швидкий запуск готового продукту.

Основні можливості Spring Boot:

1. **Автоконфігурація** — автоматичне визначення та налаштування компонентів залежно від підключених бібліотек.
2. **Вбудовані веб-сервери** (Tomcat, Jetty, Undertow) — застосунок запускається як самостійний .jar-файл, без потреби встановлення зовнішнього сервера.
3. **Стартові залежності (Starters)** — готові набори бібліотек для конкретних задач, наприклад:
    - spring-boot-starter-web — для створення REST API;
    - spring-boot-starter-data-jpa — для роботи з базами даних;

- spring-boot-starter-security — для впровадження безпеки;
- spring-boot-starter-test — для модульного тестування.

4. **Гнучке конфігурування** через файли application.properties або application.yml.
5. **Spring Boot Actuator** — модуль моніторингу стану застосунку.

Переваги використання Spring Boot:

- зменшення часу на розробку і налаштування;
- мінімізація конфігураційного коду;
- спрощене тестування та розгортання;
- легка інтеграція з хмарними платформами (AWS, Google Cloud, Azure);
- сумісність із контейнеризацією через Docker.

Spring Boot широко застосовується у створенні RESTful API, систем управління контентом, інтернет-магазинів та корпоративних систем. Його можливості дозволяють розробляти стабільні, масштабовані рішення навіть для високонавантажених середовищ.

2.4 Визначення функціоналу застосунку для оренди автомобілів

У межах даного дослідження розробляється веб-застосунок для оренди автомобілів, призначений для забезпечення користувачів зручним сервісом перегляду, бронювання та управління транспортними засобами. Основною метою застосунку є автоматизація процесів оренди автомобілів та підвищення зручності взаємодії між клієнтами й адміністраторами системи.

Функціональні можливості веб-застосунку поділяються відповідно до ролей користувачів системи: клієнт та адміністратор.

Функціонал для клієнтів передбачає:

- реєстрацію та авторизацію користувача в системі;

- перегляд каталогу доступних автомобілів;
- перегляд детальної інформації про обраний автомобіль;
- створення бронювання із зазначенням дати початку та завершення оренди;
- перегляд власних активних і завершених бронювань;
- скасування активного бронювання;
- можливість залишення відгуку після завершення оренди.

Функціонал для адміністратора включає:

- управління автопарком (додавання, редагування та видалення інформації про автомобілі);
- перегляд усіх бронювань у системі;
- контроль доступності транспортних засобів;
- перегляд статистичної інформації щодо кількості бронювань і завантаженості автопарку.

Для реалізації зазначеного функціоналу доцільно використати сучасні засоби та фреймворки платформи Java, зокрема Spring Boot для побудови серверної частини застосунку, Spring Data JPA для доступу до бази даних, Spring Security для забезпечення захисту та розмежування доступу, а також Thymeleaf для формування користувацького інтерфейсу.

2.5 Проєктування бази даних для веб-застосунку

Проєктування бази даних є ключовим етапом створення веб-застосунку для оренди автомобілів, оскільки визначає структуру зберігання інформації, логіку побудови зв’язків між сутностями та якість роботи всієї системи. У межах дослідження використовується **реляційна база даних MySQL**, яка

забезпечує надійність, високу продуктивність та зручну інтеграцію зі Spring Data JPA..

2.5.1 Основні сутності та їх призначення

У структурі бази даних передбачено шість основних таблиць:

- **Users** Зберігає інформацію про користувачів системи: клієнтів і адміністраторів.
- **roles** Довідник ролей користувачів (USER, ADMIN).
- **user_roles** Проміжна таблиця для зв'язку "багато-до-багатьох" між users і roles.
- **cars** Містить дані про автомобілі, доступні для оренди.
- **bookings** Зберігає інформацію про бронювання автомобілів користувачами.
- **reviews** Містить відгуки користувачів про орендовані автомобілі.

2.5.2 Детальний опис таблиць та їхніх полів

У таблиці 1 наведено перелік полів сутності users, що використовується для зберігання персональних та облікових даних користувачів системи.

Таблиця 1 — Структура таблиці **users**

Поле	**Тип даних**	**Призначення**
id	BIGINT	Первинний ключ
first_name	VARCHAR(50)	Ім'я користувача
middle_name	VARCHAR(50)	По батькові
last_name	VARCHAR(50)	Прізвище
email	VARCHAR(100)	Унікальна електронна адреса
password	VARCHAR(255)	Зашифрований пароль
phone	VARCHAR(20)	Контактний номер

created_at	DATETIME	Дата створення

У таблиці 2 наведено структуру таблиці roles, яка використовується для зберігання ролей користувачів системи.

Таблиця 2 — Структура таблиці **roles**

Поле	**Тип даних**	**Призначення**
id	INT	Первинний ключ
name	VARCHAR(20)	Назва ролі

У таблиці 3 наведено структуру проміжної таблиці user_roles, яка реалізує зв’язок типу «багато-до-багатьох» між користувачами та ролями.

Таблиця 3 — Структура таблиці **user_roles**

Поле	**Тип даних**	**Призначення**
user_id	BIGINT	Зовнішній ключ → users(id)
role_id	INT	Зовнішній ключ → roles(id)

У таблиці 4 наведено структуру таблиці cars, яка використовується для зберігання інформації про автомобілі, доступні для оренди.

Таблиця 4 — Структура таблиці **cars**

Поле	**Тип даних**	**Призначення**
id	BIGINT	Первинний ключ
make	VARCHAR(50)	Виробник автомобіля

model	VARCHAR(50)	Модель автомобіля
year	INT	Рік випуску
price_per_day	DECIMAL(10,2)	Вартість оренди за добу
is_available	BOOLEAN	Ознака доступності
description	TEXT	Опис автомобіля
image_url	VARCHAR(255)	Посилання на зображення

У таблиці 5 наведено структуру таблиці **bookings**, яка призначена для зберігання інформації про бронювання автомобілів користувачами.

Таблиця 5 — Структура таблиці **bookings**

Поле	Тип даних	Призначення
id	BIGINT	Первинний ключ
user_id	BIGINT	Зовнішній ключ → users(id)
car_id	BIGINT	Зовнішній ключ → cars(id)
start_date	DATE	Дата початку бронювання
end_date	DATE	Дата завершення бронювання
is_active	BOOLEAN	Ознака активності
cancelled_at	DATETIME	Дата дострокового скасування
created_at	DATETIME	Дата створення

У таблиці 6 наведено структуру таблиці **reviews**, яка використовується для зберігання відгуків користувачів після завершення бронювання.

Таблиця 6 — Структура таблиці **reviews**

Поле	Тип даних	Призначення
id	BIGINT	Первинний ключ
booking_id	BIGINT	Зовнішній ключ → bookings(id)
rating	INT	Оцінка
comment	TEXT	Текст відгуку
created_at	DATETIME	Дата створення

2.5.6 Опис зв’язків між таблицями

У результаті проєктування сформовано такі зв’язки між таблицями бази даних:

- між таблицями users та bookings реалізовано зв’язок типу «один-до-багатьох»;
- між таблицями cars та bookings реалізовано зв’язок типу «один-до-багатьох»;
- між таблицями users та roles реалізовано зв’язок типу «багато-до-багатьох» через таблицю user_roles;
- між таблицями bookings та reviews реалізовано зв’язок типу «один-до-нуль або один».

Прямий зв’язок між таблицями users та cars відсутній, оскільки взаємодія між ними здійснюється виключно через таблицю bookings.

На рисунку 1 показано ER-діаграму, яка відображає основні сутності предметної області та зв’язки між ними.



Рисунок 1 — ER-діаграма бази даних веб-застосунку

2.6 Створення Maven-проєкту засобами Spring Initializr

Для створення веб-застосунку з використанням Spring Framework було застосовано інструмент Spring Initializr — офіційний генератор стартових Spring Boot-проєктів [5].

Цей сервіс дозволяє швидко сформувати структуру застосунку, визначити необхідні залежності та автоматично згенерувати файл pom.xml.

2.6.1 Вибір параметрів проєкту

Під час створення проєкту були обрані такі параметри:

- Project: Maven Project
- Language: Java
- Spring Boot: 3.3.x
- Packaging: Jar
- Java Version: 17
- Group: com.carrental
- Artifact: car-rental
- Name: Car Rental Application

Spring Boot 3.3.x забезпечує найкращу сумісність із Java 17, сучасними залежностями, покращеними механізмами безпеки та оптимізованим автоконфігуруванням.

2.6.2 Додавання необхідних залежностей

Під час генерації проєкту через Spring Initializr до нього були додані такі стартові залежності:

1. spring-boot-starter-web забезпечує роботу Spring MVC, обробку HTTP-запитів, маршрутизацію та REST-функціональність.

2. spring-boot-starter-thymeleaf використовується для побудови HTML-шаблонів, форм у веб-інтерфейсі та відображення даних користувачеві.

3. spring-boot-starter-data-jpa забезпечує інтеграцію Spring Data JPA з MySQL та ORM-фреймворк Hibernate.

4. MySQL Driver Драйвер для взаємодії з реляційною СУБД MySQL.

5. spring-boot-starter-security використовується для впровадження автентифікації та авторизації користувачів.

6. spring-boot-starter-validation використовується у веб-застосунку для забезпечення валідації даних, що вводяться користувачем у формах реєстрації,

авторизації та бронювання автомобілів. Перевірка коректності введених даних реалізується на рівні контролерів і доменних об’єктів із використанням стандартних анотацій Bean Validation, таких як @NotNull, @NotBlank, @Email та @Size. Застосування валідації дозволяє запобігти збереженню некоректних або неповних даних у базі даних.

2.6.3 Формування файлу pom.xml

На основі вибраних параметрів Spring Initializr автоматично згенерує файл pom.xml.

У лістингу 10 наведено приклад ключових залежностей, які формуються в результаті.

Лістинг 10 — Ключові залежності Spring Boot-проєкту

```
<dependencies>
    <!-- Web and MVC -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Thymeleaf -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- JPA / Hibernate -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
```

```
        <!-- MySQL Driver -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>

        <!-- Security -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <!-- Validation -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
validation</artifactId>
        </dependency>
```

Ці залежності забезпечують роботу всіх основних компонентів майбутнього веб-застосунку.

2.6.4 Структура згенерованого Spring Boot-проєкту

Після натискання кнопки *Generate* Spring Initializr формує архів із початковою структурою проєкту:

Основні каталоги показано на рисунку 2.

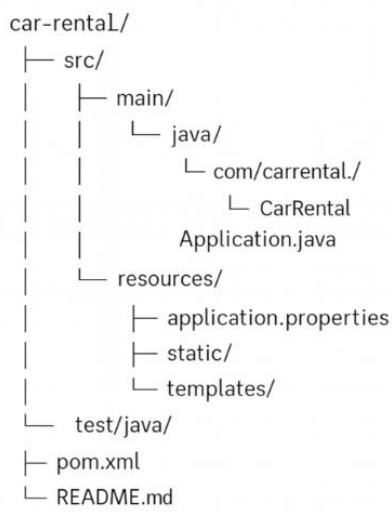


Рисунок 2 – структура проєкту

- /java/ — містить вихідний Java-код, контролери, сервісні класи, сутності, репозиторії;
- /templates/ — HTML-шаблони Thymeleaf;
- /static/ — CSS, зображення та інші статичні файли;
- application.properties — конфігурація доступу до MySQL і налаштування Spring.

Переваги використання Spring Initializr

- автоматичне формування структури проєкту;
- коректно налаштована сумісність версій Spring Boot та залежностей;
- мінімізація помилок початкової конфігурації;
- можливість швидкого розгортання застосунку;
- стандартизований pom.xml, який легко розширювати.

2.7 Висновки до розділу 2

Здійснено літературний огляд сучасних підходів до розробки веб-застосунків на платформі Java із використанням фреймворку Spring. Проаналізовано еволюцію Java-вебтехнологій та обґрунтовано доцільність

застосування Spring Boot як основи для створення масштабованих і безпечних веб-систем.

Розглянуто архітектурний шаблон Model–View–Controller (MVC), який забезпечує чітке розмежування відповідальності між рівнями застосунку та сприяє підвищенню підтримуваності й тестованості програмного коду. Визначено роль Spring MVC у побудові контролерів та маршрутизації HTTP-запитів.

Проаналізовано можливості Spring Data JPA та ORM-фреймворку Hibernate для роботи з реляційними базами даних, що дозволяє зменшити обсяг низькорівневого коду та спростити реалізацію доступу до даних. Розглянуто підходи до проєктування сутностей і взаємозв’язків між ними.

Досліджено механізми автентифікації та авторизації, які надає Spring Security, а також принципи розмежування доступу на основі ролей користувачів. Показано доцільність використання цього фреймворку для захисту веб-застосунків.

Окрему увагу приділено шаблонізатору Thymeleaf як інструменту формування серверного веб-інтерфейсу, який забезпечує тісну інтеграцію з Spring MVC та підтримує динамічне відображення даних.

Результати проведеного літературного огляду стали теоретичною основою для проєктування та реалізації веб-застосунку оренди автомобілів, архітектура і функціонал якого розглядаються у наступному розділі роботи.

РОЗДІЛ 3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ІЗ ВИКОРИСТАННЯМ SPRING BOOT, SPRING MVC, SPRING DATA JPA ТА THYMELEAF

3.1 Загальні принципи архітектури веб-застосунку

Для реалізації веб-застосунку було обрано платформу **Java 17**, оскільки вона є версією з довгостроковою підтримкою (LTS), що забезпечує стабільність, сумісність та регулярні оновлення безпеки. Використання LTS-версії є рекомендованою практикою для корпоративних та академічних проєктів, на відміну від проміжних релізів, які мають обмежений термін підтримки.

Фреймворк **Spring Boot 3.x** використано як стабільне та поширене рішення, повністю сумісне з Java 17 і засноване на специфікаціях **Jakarta EE**. На момент розроблення роботи версія Spring Boot 4.0 перебувала на етапі активного розвитку, що могло призвести до змін API та нестабільної поведінки, тому її застосування у навчальному проєкті є недоцільним.

Таким чином, поєднання Java 17 та Spring Boot 3.x забезпечує оптимальний баланс між актуальністю технологій, стабільністю платформи та відповідністю сучасним вимогам до розробки веб-застосунків.

Архітектура веб-застосунку побудована за шаблоном **Model–View–Controller (MVC)** та доповнена механізмами **інверсії управління (IoC)** і **ін’єкції залежностей (Dependency Injection, DI)**, що підвищує модульність, керованість компонентів і спрощує підтримку системи.

Архітектура застосунку включає такі логічні рівні:

• Model (шар сутностей) — описує доменні об’єкти та структуру бази даних з використанням JPA-анотацій;

• Repository (шар доступу до даних) — забезпечує взаємодію з базою даних за допомогою Spring Data JPA;

• Service (сервісний шар) — реалізує бізнес-логіку застосунку та правила обробки даних;

• Controller (шар контролерів) — обробляє HTTP-запити та координує взаємодію між сервісами й представленням;

• View (шар представлення) — реалізований у вигляді HTML-шаблонів з використанням Thymeleaf;

• Security (шар безпеки) — відповідає за автентифікацію, авторизацію та контроль доступу користувачів;

• Database (база даних) — реляційна база даних MySQL, у якій зберігається вся інформація застосунку.

Обрана архітектура веб-застосунку є **монолітною**, що є доцільним для навчального та демонстраційного проєкту. Такий підхід спрощує розгортання, налагодження та аналіз взаємодії між компонентами системи. Водночас структура коду організована з урахуванням принципів модульності, що створює передумови для подальшого масштабування або рефакторингу системи.

На рисунку 3 представлено загальну архітектуру веб-застосунку, яка відображає взаємодію між основними програмними шарами: представленням, контролерами, сервісами, репозиторіями, шаром безпеки та базою даних.

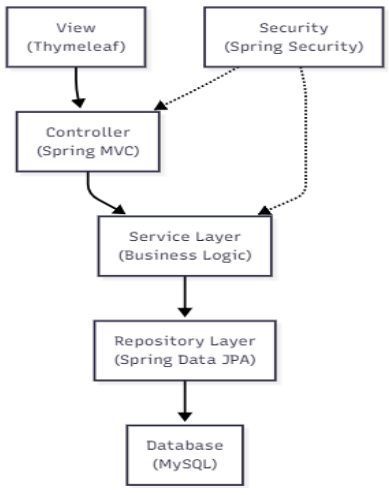


Рисунок 3 — Загальна схема архітектури веб-застосунку

3.2 Функціональна структура системи оренди автомобілів

Функціональна структура розробленого веб-застосунку сформована на основі вимог до функціоналу, визначених у підрозділі 2.4, та відображає логіку організації взаємодії між користувачами і системою.

Система підтримує розмежування доступу відповідно до ролей користувачів, що забезпечує різний набір можливостей для клієнтів і адміністратора. Контроль доступу до функціональних можливостей реалізується на рівні серверної частини застосунку із використанням засобів Spring Security.

Функціональні сценарії клієнта реалізуються через веб-інтерфейс і включають перегляд інформації про автомобілі, створення та управління власними бронюваннями. Обробка запитів користувача здійснюється контролерами Spring MVC, які взаємодіють із сервісним рівнем для виконання бізнес-логіки.

Адміністративна частина системи реалізує функції керування даними автопарку та бронюваннями і доступна виключно користувачам з відповідними правами. Адміністратор має можливість контролювати стан автомобілів та аналізувати інформацію про оренду.

Інтерфейс користувача формується за допомогою шаблонізатора Thymeleaf, що забезпечує динамічне відображення даних, отриманих із серверної частини застосунку. Така функціональна структура забезпечує цілісність системи, відповідність вимогам предметної області та створює основу для подальшого розширення функціональних можливостей веб-застосунку.

3.3 Архітектурне проєктування застосунку

Архітектура веб-застосунку побудована відповідно до багаторівневого підходу, який забезпечує чітке розділення відповідальностей між компонентами системи, спрощує її супровід та створює передумови для масштабування.

Веб-застосунок реалізований на основі фреймворку **Spring Boot 3.x**, який використовує архітектурний шаблон **Model–View–Controller (MVC)**, а також механізми **інверсії управління (IoC)** та **ін'єкції залежностей (DI)** для керування компонентами системи. Застосування цих підходів забезпечує слабке зв'язування між модулями, підвищує модульність коду та спрощує його підтримку.

Архітектуру веб-застосунку сформовано за класичною багаторівневою схемою, яка включає такі основні рівні:

- **Рівень даних (Data / Model Layer)** — відповідає за представлення доменних сутностей та їх відображення у базі даних.
- **Репозиторний рівень (Repository Layer)** — забезпечує доступ до даних та взаємодію з базою даних за допомогою технології Spring Data JPA.
- **Сервісний рівень (Service Layer)** — містить бізнес-логіку застосунку та координує роботу між репозиторіями і контролерами.
- **Контролерний рівень (Web / Controller Layer)** — обробляє HTTP-запити, формує модель даних та передає її до рівня представлення.
- **Рівень представлення (View Layer / Thymeleaf)** — відповідає за відображення даних користувачу за допомогою HTML-шаблонів.

На рисунку 4 наведено узагальнену схему взаємодії основних рівнів архітектури веб-застосунку.

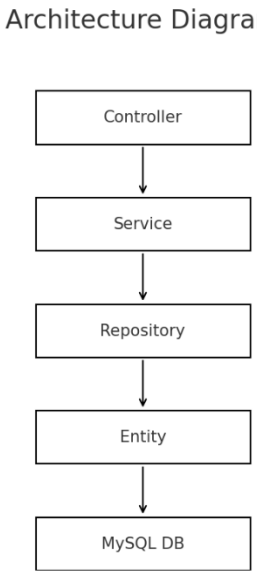


Рисунок 4 — Схема багаторівневої архітектури веб-застосунку

3.3.1 Рівень даних (Model Layer)

Рівень даних містить сутності, які відображають структуру таблиць MySQL у вигляді Java-класів.

Сутності позначаються анотацією **@Entity**, а взаємозв'язки між таблицями описуються через **@OneToMany**, **@ManyToOne**,**@ManyToMany**, **@OneToOne.**

Основні сутності:

- **User** – зберігає дані користувачів;
- **Role** – визначає ролі користувачів;
- **Car** – моделює транспортний засіб, доступний для оренди;
- **Booking** – описує процес бронювання;
- **Review** – описує відгук користувача.

Для забезпечення цілісності даних використовуються:

- зовнішні ключі (FK);
- анотації валідаторів (наприклад, @NotNull, @Email, @Size);

- обмеження JPA та Hibernate (nullable, unique, length)

У лістингу 11 наведена сутність **User**, яка використовується для зберігання даних користувачів системи та реалізує механізм розмежування доступу на основі ролей.

Лістинг 11 – Клас сутності User

```
@Entity
@Table(name = "users")
@Getter
@Setter
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 100)
    private String fullName;

    @Column(nullable = false, unique = true, length = 100)
    private String email;

    @Column(nullable = false)
    private String password;

    @Column(length = 20)
    private String phone;

    private LocalDateTime createdAt = LocalDateTime.now();

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "user_roles",
```

```
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
        )
        private Set<Role> roles;
    }
```

У лістингу 12 наведена сутність **Role** забезпечує реалізацію ролей доступу, зокрема ROLE_USER та ROLE_ADMIN.

Лістинг 12 – Клас сутності Car

```
@Entity
@Table(name = "cars")
@Getter
@Setter
public class Car {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Виробник обов'язковий")
    private String make;

    @NotBlank(message = "Модель обов'язкова")
    private String model;

    @Min(value = 1970)
    private int year;

    @DecimalMin(value = "0.0")
    private double pricePerDay;

    private boolean available = true;
```

```
    private String description;
    private String imageUrl;
    private String category;
}
```

У лістингу 13 наведено реалізацію сутності автомобіля, яка використовується для зберігання інформації про транспортні засоби, доступні для оренди.

Лістинг 13 – Клас сутності Car

```
@Entity
@Table(name = "cars")
@Getter
@Setter
public class Car {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Виробник обов’язковий")
    @Column(nullable = false, length = 50)
    private String make;

    @NotBlank(message = "Модель обов’язкова")
    @Column(nullable = false, length = 50)
    private String model;

    @Min(value = 1970, message = "Рік некоректний")
    private int year;
```

```
    @DecimalMin(value = "0.0", message = "Ціна некоректна")
    @Column(name = "price_per_day", nullable = false)
    private double pricePerDay;

    @Column(name = "is_available")
    private boolean available = true;

    @Column(length = 1000)
    private String description;

    @Column(length = 255)
    private String imageUrl;

    @Column(length = 30)
    private String category;
}
```

Дана сутність описує автомобіль, доступний для оренди, та містить інформацію про марку, модель, рік випуску, вартість оренди за добу і статус доступності. Для забезпечення коректності введених даних використано механізм серверної валідації на основі анотацій **Jakarta Validation.**

У лістингу 14 наведено сутність **Bookin,** яка реалізує зв'язок між користувачем і автомобілем та зберігає інформацію про період оренди**.**

Лістинг 14 – Клас сутності Booking

```
@Entity
@Table(name = "bookings")
@Getter
@Setter
public class Booking {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

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

    @ManyToOne
    @JoinColumn(name = "car_id", nullable = false)
    private Car car;

    private LocalDate startDate;
    private LocalDate endDate;

    private boolean active = true;

    private LocalDateTime createdAt = LocalDateTime.now();
    private LocalDateTime cancelledAt;
}
```

У лістингу 15 наведено сутність **Review**, відгук пов’язаний із бронюванням, що унеможливлює його створення без фактичного користування послугою.

Лістинг 15 – Клас сутності Review

```
@Entity
@Table(name = "reviews")
@Getter
@Setter
public class Review {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    @OneToOne
    @JoinColumn(name = "booking_id", nullable = false)
    private Booking booking;

    @Min(1)
    @Max(5)
    private int rating;

    @Column(length = 1000)
    private String comment;

    private LocalDateTime createdAt = LocalDateTime.now();
}
```

3.3.2 Репозиторний рівень (Repository Layer)

Репозиторії забезпечують доступ до бази даних і реалізуються за допомогою **Spring Data JPA**, що дозволяє працювати з MySQL через об’єктно-реляційне відображення (ORM) без написання SQL-запитів.

Кожен репозиторій є інтерфейсом, що розширює JpaRepository:

- UserRepository;
- RoleRepository;
- CarRepository;
- BookingRepository;
- ReviewRepository;

Типові методи:

- findAll();
- findById();
- save();

- deleteById();

Spring Data JPA автоматично генерує потрібну реалізацію та SQL під час виконання програми.

Переваги такого підходу:

- мінімальна кількість коду;
- автоматична генерація запитів;
- підтримка складних пошуків через *query methods*;
- інтеграція з транзакціями Spring (@Transactional).

У лістингу 16 наведено реалізацію репозиторію для роботи з користувачами системи.

Лістинг 16 – Інтерфейс UserRepository

```
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByEmail(String email);
}
```

Даний репозиторій використовується для отримання інформації про користувачів, зокрема під час процесу автентифікації та авторизації у системі.

У лістингу 17 наведено репозиторій для роботи з ролями доступу.

Лістинг 17 – Інтерфейс RoleRepository

```
public interface RoleRepository extends JpaRepository<Role, Long> {

    Optional<Role> findByName(String name);
}
```

Репозиторій дозволяє отримувати ролі користувачів за назвою та використовується під час створення облікових записів і призначення прав доступу.

У лістингу 18 наведено репозиторій для доступу до даних про автомобілі.

Лістинг 18 – Інтерфейс CarRepository

```
public interface CarRepository extends JpaRepository<Car, Long> {

    List<Car> findByAvailableTrue();
```

Даний репозиторій забезпечує отримання списку доступних для оренди автомобілів і використовується при формуванні каталогу транспортних засобів.

У лістингу 19 наведено репозиторій для роботи з бронюваннями автомобілів.

Лістинг 19 – Інтерфейс BookingRepository

```
public interface BookingRepository extends JpaRepository<Booking, Long> {

    List<Booking> findByUserEmailOrderByCreatedAtDesc(String email);

    List<Booking> findAllByOrderByCreatedAtDesc();

    @Query("""
        select count(b) from Booking b
        where b.car.id = :carId
```

```
            and b.active = true
            and (:start < b.endDate and :end > b.startDate)
        """)
        long countOverlaps(Long carId, LocalDate start, LocalDate
end);
    }
```

Даний репозиторій містить користувацький запит для перевірки перетину періодів бронювання, що дозволяє запобігти одночасній оренді одного автомобіля декількома користувачами.

У лістингу 20 наведено репозиторій для роботи з відгуками користувачів.

Лістинг 20 – Інтерфейс ReviewRepository

```
    public interface ReviewRepository extends
JpaRepository<Review, Long> {

        Optional<Review> findByBookingId(Long bookingId);
    }
```

Репозиторій використовується для отримання відгуків, пов’язаних із конкретними бронюваннями, що забезпечує коректну логіку залишення відгуків лише після завершення оренди.

3.3.3 Сервісний рівень (Service Layer)

Сервісний рівень містить бізнес-логіку застосунку та координує взаємодію між контролерами і репозиторіями. Кожен сервіс — це Spring-компонент, позначений анотацією **@Service**.

Основні сервіси:

- **UserService** – реєстрація, пошук користувачів, керування ролями;

- **CarService** – отримання переліку автомобілів, створення та оновлення записів;
- **BookingService** – створення, скасування і завершення бронювання;
- **ReviewService** – додавання й відображення відгуків.

Основні обов’язки сервісного рівня:

- перевірка коректності даних;
- логіка бронювання (перевірка доступності авто, дат);
- оновлення статусів бронювань;
- виклик репозиторіїв;
- формування готових даних для контролерів.

Сервіси взаємодіють лише з репозиторіями, не звертаючись безпосередньо до БД, що відповідає концепції чистої архітектури.

У лістингу 21 наведено реалізацію сервісу для роботи з автомобілями.

Лістинг 21 – Клас CarService

```
@Service
@RequiredArgsConstructor
public class CarService {

    private final CarRepository carRepository;

    public List<Car> all() {
        return carRepository.findAll();
    }

    public List<Car> availableCars() {
        return carRepository.findByAvailableTrue();
    }
```

```
    public Car get(Long id) {
        return carRepository.findById(id).orElseThrow();
    }

    public Car save(Car car) {
        return carRepository.save(car);
    }

    public void delete(Long id) {
        carRepository.deleteById(id);
```

Даний сервіс забезпечує виконання операцій додавання, видалення та отримання інформації про автомобілі, а також використовується як у клієнтській, так і в адміністративній частинах застосунку.

У лістингу 22 наведено сервіс, який реалізує бізнес-логіку створення та перевірки бронювань.

Лістинг 22 – Клас BookingService

```
@Service
@RequiredArgsConstructor
public class BookingService {

    private final BookingRepository bookingRepository;

    public List<Booking> userBookings(String email) {
        return
bookingRepository.findByUserEmailOrderByCreatedAtDesc(email);
    }

    public List<Booking> allBookings() {
        return
bookingRepository.findAllByOrderByCreatedAtDesc();
    }
```

```
        public boolean isAvailable(Long carId, LocalDate start,
LocalDate end) {
            return bookingRepository.countOverlaps(carId, start,
end) == 0;
        }

        public Booking save(Booking booking) {
            return bookingRepository.save(booking);
        }

        public void cancel(Booking booking) {
            booking.setActive(false);
            booking.setCancelledAt(LocalDateTime.now());
            bookingRepository.save(booking);
        }
    }
```

Сервіс бронювання реалізує перевірку доступності автомобіля на заданий період, створення бронювань та їх скасування. Перевірка перетину дат дозволяє запобігти одночасній оренді одного автомобіля кількома користувачами.

У лістингу 23 наведено сервіс для роботи з користувачами системи.

Лістинг 23 – Клас UserService

```
@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final RoleRepository roleRepository;
    private final PasswordEncoder passwordEncoder;
```

```
        public User register(User user) {

user.setPassword(passwordEncoder.encode(user.getPassword()));
            user.setRoles(Set.of(

roleRepository.findByName("ROLE_USER").orElseThrow()
            ));
            return userRepository.save(user);
        }

        public User getByEmail(String email) {
            return
userRepository.findByEmail(email).orElseThrow();
        }

        public List<User> allUsers() {
            return userRepository.findAll();
        }
```

3.3.4 Контролерний рівень (Controller Layer)

Контролери відповідають за обробку HTTP-запитів користувачів та передачу даних на рівень представлення.

Spring MVC дозволяє описувати контролери через анотації **@Controller** та **@RequestMapping**.

Структура контролерів:

- **HomeController** – відображення головної сторінки та каталогу автомобілів;
- **CarController** – детальна інформація про авто та форма створення бронювання;
- **BookingController** – перегляд і скасування бронювань;
- **ReviewController** – обробка відгуків;

- **AdminController / AdminCarController** – додавання, редагування, видалення автомобілів.

Контролери виконують такі завдання:

- приймають параметри запиту;
- передають дані в модель (Model);
- викликають методи сервісів;
- повертають HTML-шаблони Thymeleaf.

Саме контролери визначають маршрутизацію системи, наприклад:

GET /cars

GET /cars/{id}

POST /cars/{id}/book

GET /admin/cars

POST /admin/cars/add

У лістингу 24 наведено контролер, який відповідає за відображення головної сторінки веб-застосунку.

Лістинг 24 – Клас HomeController

```
@Controller
@RequiredArgsConstructor
public class HomeController {

    private final CarService carService;

    @GetMapping("/")
    public String home(Model model) {
        model.addAttribute("cars",
carService.availableCars());
        return "index";
    }
}
```

Даний контролер формує початкову сторінку застосунку та відображає перелік доступних для оренди автомобілів.

У лістингу 25 наведено контролер, який реалізує перегляд каталогу автомобілів.

Лістинг 25 – Клас CarController

```
@Controller
@RequestMapping("/catalog")
@RequiredArgsConstructor
public class CarController {

    private final CarService carService;

    @GetMapping
    public String catalog(Model model) {
        model.addAttribute("cars",
carService.availableCars());
        return "catalog";
    }

    @GetMapping("/{id}")
    public String details(@PathVariable Long id, Model model)
{
        model.addAttribute("car", carService.get(id));
        return "car-details";
    }
}
```

Контролер забезпечує перегляд каталогу транспортних засобів та детальної інформації про обраний автомобіль.

У лістингу 26 наведено контролер, який відповідає за створення бронювань користувачами.

Лістинг 26 – Клас BookingController

```
@Controller
@RequestMapping("/booking")
@RequiredArgsConstructor
public class BookingController {

    private final BookingService bookingService;
    private final CarService carService;
    private final UserService userService;

    @PostMapping("/create")
    public String create(@RequestParam Long carId,
                         @RequestParam LocalDate startDate,
                         @RequestParam LocalDate endDate,
                         Principal principal) {

        if (!bookingService.isAvailable(carId, startDate,
endDate)) {
            return "redirect:/catalog?error";
        }

        Booking booking = new Booking();
        booking.setCar(carService.get(carId));

booking.setUser(userService.getByEmail(principal.getName()));
        booking.setStartDate(startDate);
        booking.setEndDate(endDate);

        bookingService.save(booking);
        return "redirect:/profile";
```

```
        }
    }
```

Контролер реалізує сценарій створення бронювання з попередньою перевіркою доступності автомобіля.

У лістингу 27 наведено контролер, який відповідає за відображення особистого кабінету користувача.

Лістинг 27 – Клас ProfileController

```
@Controller
@RequestMapping("/profile")
@RequiredArgsConstructor
public class ProfileController {

    private final BookingService bookingService;

    @GetMapping
    public String profile(Model model, Principal principal)
{
        model.addAttribute(
            "bookings",

bookingService.userBookings(principal.getName())
        );
        return "client/profile";
    }
}
```

Даний контролер дозволяє користувачу переглядати власні активні та завершені бронювання.

У лістингу 28 наведено контролер, який реалізує адміністративний функціонал керування автопарком.

Лістинг 28 – Клас AdminCarController

```
@Controller
@RequestMapping("/admin/cars")
@RequiredArgsConstructor
public class AdminCarController {

    private final CarService carService;

    @GetMapping
    public String cars(Model model) {
        model.addAttribute("cars", carService.all());
        model.addAttribute("car", new Car());
        return "admin/cars";
    }

    @PostMapping("/add")
    public String add(@Valid @ModelAttribute("car") Car car,
                      BindingResult result,
                      Model model) {
        if (result.hasErrors()) {
            model.addAttribute("cars", carService.all());
            return "admin/cars";
        }
        carService.save(car);
        return "redirect:/admin/cars";
    }
}
```

Контролер доступний лише користувачам з роллю адміністратора та забезпечує додавання і редагування інформації про автомобілі.

3.3.5 Рівень представлення (View Layer) Thymeleaf

Рівень представлення реалізовано за допомогою **Thymeleaf**, який інтегрується зі Spring MVC і дозволяє будувати динамічні HTML-сторінки.

Основні особливості Thymeleaf:

Підтримка шаблонів HTML із динамічними атрибутами

- th:text — вивід тексту
- th:each — цикли
- th:if, th:unless — умовне відображення
- th:href, th:src — динамічні посилання
- th:object, th:field — робота з формами

Модель даних

Контролер передає в шаблон дані через об’єкт Model, наприклад:

model.addAttribute("cars", carService.findAll());

Структура HTML-шаблонів

У каталозі /resources/templates/ розміщуються сторінки:

- index.html;
- car.html;
- bookings.html;
- reviews.html;
- admin/cars.html;
- fragments/header.html ;

Підтримка валідації

У веб-застосунку реалізовано механізм валідації даних користувача з використанням модуля **spring-boot-starter-validation** у поєднанні з шаблонізатором **Thymeleaf**. Валідація застосовується з метою перевірки коректності даних, що вводяться користувачем у формах реєстрації, авторизації та бронювання автомобілів, ще до їх збереження у базі даних.

Перевірка коректності даних здійснюється на основі стандарту **Bean Validation** та реалізується за допомогою анотацій, які задають обмеження для полів доменних об’єктів. Основними критеріями валідації є:

- обов’язковість заповнення полів;
- відповідність формату введених даних (наприклад, електронної пошти);
- обмеження довжини текстових значень;
- коректність дат бронювання (дата початку та завершення).

Зокрема, для полів форм можуть використовуватися анотації @NotNull, @NotBlank, @Size, @Email, які дозволяють автоматично перевіряти введені значення на відповідність заданим вимогам.

У випадку виявлення помилок валідації відповідні повідомлення автоматично передаються до рівня представлення. Шаблонізатор **Thymeleaf** забезпечує відображення цих повідомлень безпосередньо у формах користувача, що підвищує зручність взаємодії із системою та дозволяє оперативно виправити помилки введення.

Приклад фрагмента HTML-шаблону демонструє відображення повідомлення про помилку для поля дати початку бронювання у разі порушення правил валідації.

Таким чином, механізм валідації дозволяє забезпечити цілісність та коректність даних, мінімізувати кількість помилок користувача та підвищити надійність веб-застосунку в цілому.

Переваги такої багаторівневої архітектури

- чітке розділення відповідальностей;
- легке тестування сервісів і контролерів;
- підтримуваність і розширюваність;
- повна сумісність зі Spring Boot;
- коректна робота зі Spring Security;

- можливість винесення фронтенду на інший фреймворк у майбутньому/

3.4 UML-діаграма доменної моделі веб-застосунку

Побудова UML-діаграми доменної моделі дає можливість узагальнити структуру основних сутностей веб-застосунку оренди автомобілів та визначити зв’язки між ними на рівні програмної логіки. UML-діаграма класів використовується для відображення об’єктної моделі системи та слугує основою для подальшої реалізації сутностей у програмному коді.

На основі спроєктованої схеми бази даних було сформовано UML-діаграму доменної моделі, яка відображає ключові класи, їх атрибути та асоціації між ними. Кожен клас доменної моделі є проєкцією відповідної сутності бази даних та містить набір полів, що відповідають атрибутам відповідних таблиць.

До доменної моделі веб-застосунку входять такі основні класи: **User**, **Role**, **Car**, **Booking** та **Review**.

Клас **User** зберігає інформацію про користувачів системи та пов’язаний із класом **Booking** зв’язком типу «один-до-багатьох», що відображає можливість створення користувачем декількох бронювань. Крім того, клас **User** асоційований із класом **Role** зв’язком типу «багато-до-багатьох», що дозволяє реалізувати механізм авторизації та розмежування доступу на основі ролей користувачів.

Клас **Car** описує транспортний засіб, доступний для оренди, та пов’язаний із класом **Booking** зв’язком типу «один-до-багатьох», оскільки один автомобіль може бути заброньований багаторазово різними користувачами у різні періоди часу. Прямий зв’язок між класами **User** та **Car** у доменній моделі відсутній, оскільки їх взаємодія здійснюється виключно через клас **Booking**.

Клас **Booking** є центральною сутністю доменної моделі та реалізує основну бізнес-логіку застосунку. Він поєднує в собі інформацію про користувача, автомобіль та часові межі оренди. Активність бронювання визначається на основі дат оренди та відповідного логічного атрибута, що дозволяє коректно обробляти як завершені, так і достроково скасовані бронювання.

Клас **Review** призначений для зберігання відгуків користувачів після завершення бронювання та пов’язаний із класом **Booking** зв’язком типу «один-до-нуль або один». Такий підхід забезпечує однозначну відповідність відгуку конкретному бронюванню та опосередковано пов’язує відгук із користувачем і автомобілем.

На Рисунку 5 наведено UML-діаграму доменної моделі застосунку, що демонструє структуру класів, їх атрибути та всі необхідні зв’язки.

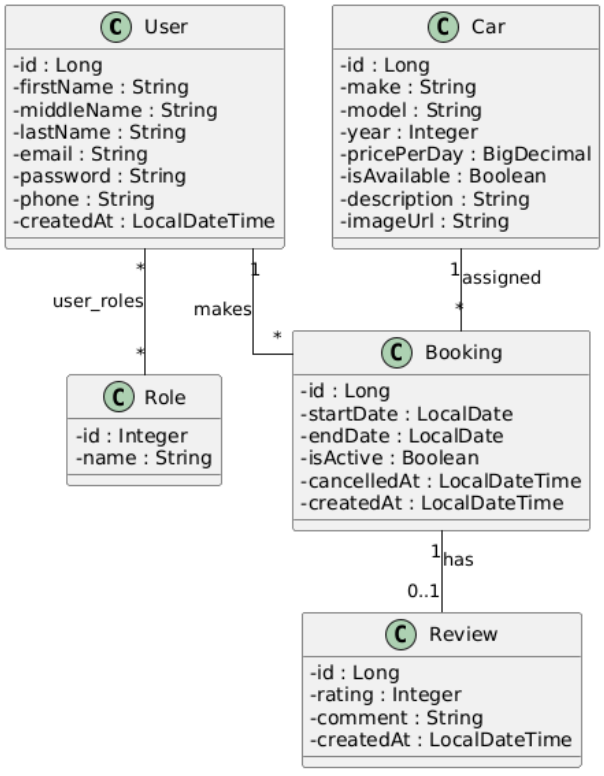


Рисунок 5 – UML-діаграма доменної моделі веб-застосунку

3.5 Реалізація взаємодії між компонентами

На рівні **контролерів** реалізовано логіку оброблення запитів користувача. Контролер *HomeController* відповідає за відображення каталогу автомобілів, *CarController* — за перегляд деталей і створення бронювання, *BookingController* — за перегляд та скасування бронювань, *AdminCarController* — за роботу адміністратора з автопарком.

На **сервісному рівні** класи *CarService* і *BookingService* виконують перевірки введених даних і викликають методи відповідних репозиторіїв.

Рівень даних реалізовано за допомогою Spring Data JPA, що дозволяє уникнути написання SQL-запитів — усі стандартні операції (пошук, збереження, видалення) виконуються автоматично.

Для валідації застосовано бібліотеку **Jakarta Validation**, а обмеження (наприклад, перевірка дат або коректності електронної пошти) задаються анотаціями безпосередньо в класах сутностей.

3.6 Представлення даних користувачу

Інтерфейс користувача реалізовано засобами шаблонізатора **Thymeleaf**, який інтегрований зі Spring MVC.

Створено чотири основні сторінки:

- **index.html** — головна сторінка з переліком доступних автомобілів;
- **car.html** — детальна сторінка автомобіля з формою бронювання;
- **bookings.html** — сторінка перегляду власних бронювань і кнопкою *Скасувати*;
- **admin/cars.html** — адміністративна панель для додавання та видалення автомобілів.

Оформлення виконано за допомогою простих CSS-стилів, що забезпечують зручне відображення на екрані ноутбука або проєктора під час захисту. Всі сторінки підтримують двосторонній обмін даними між сервером і шаблоном через атрибути моделі, які заповнюються контролерами.

3.7 Безпека та авторизація користувачів

Доступ до відкритих сторінок (*/*, */cars/*, */bookings*) дозволено всім користувачам без авторизації.

Адміністративні маршрути (//admin/***) захищено системою **Spring Security**.

У демонстраційній версії використано вбудоване сховище користувачів *InMemoryUserDetailsManager*, у якому створено обліковий запис адміністратора: **логін — admin**, **пароль — password**.

Spring Security автоматично формує сторінки входу й виходу та захищає доступ до адмін-розділу. У майбутньому цей механізм може бути розширений для інтеграції з базою даних користувачів або зовнішніми сервісами аутентифікації.

3.8. Використання шаблонізатора Thymeleaf у веб-застосунку

У веб-застосунку для оренди автомобілів рівень представлення реалізовано за допомогою шаблонізатора Thymeleaf, який є рекомендованим рішенням для серверної генерації HTML у рамках фреймворку Spring MVC. Застосування Thymeleaf дозволяє поєднати логіку формування інтерфейсу користувача з модельними даними, що передаються з контролерів, без використання додаткових клієнтських фреймворків.

Шаблони представлення розміщені у каталозі **resources/templates** та формують основні сторінки веб-застосунку. Для забезпечення повноцінного рівня представлення реалізовано такі основні макети інтерфейсу:

- головна сторінка з переліком доступних автомобілів;
- сторінка детального перегляду інформації про автомобіль;
- форма бронювання автомобіля;
- сторінки реєстрації та авторизації користувачів;
- сторінка керування автомобілями для адміністратора.

Кожен HTML-шаблон побудований з використанням стандартних атрибутів Thymeleaf (th:text, th:href, th:each, th:if), що дозволяє динамічно відображати дані, отримані з серверної частини застосунку. Такий підхід забезпечує коректне відображення списків автомобілів, даних бронювання та інформації про користувачів.

На рисунках 6–9 наведено приклади основних сторінок веб-застосунку, які демонструють реалізацію рівня представлення за допомогою Thymeleaf. Наведені макети підтверджують, що інтерфейс користувача є структурованим, логічно організованим та забезпечує зручну взаємодію з функціоналом системи.

Таким чином, використання шаблонізатора Thymeleaf у поєднанні з Spring MVC дозволяє реалізувати повноцінний рівень представлення веб-застосунку, забезпечуючи динамічне формування сторінок, інтеграцію з серверною логікою та підтримку валідації введених користувачем даних.

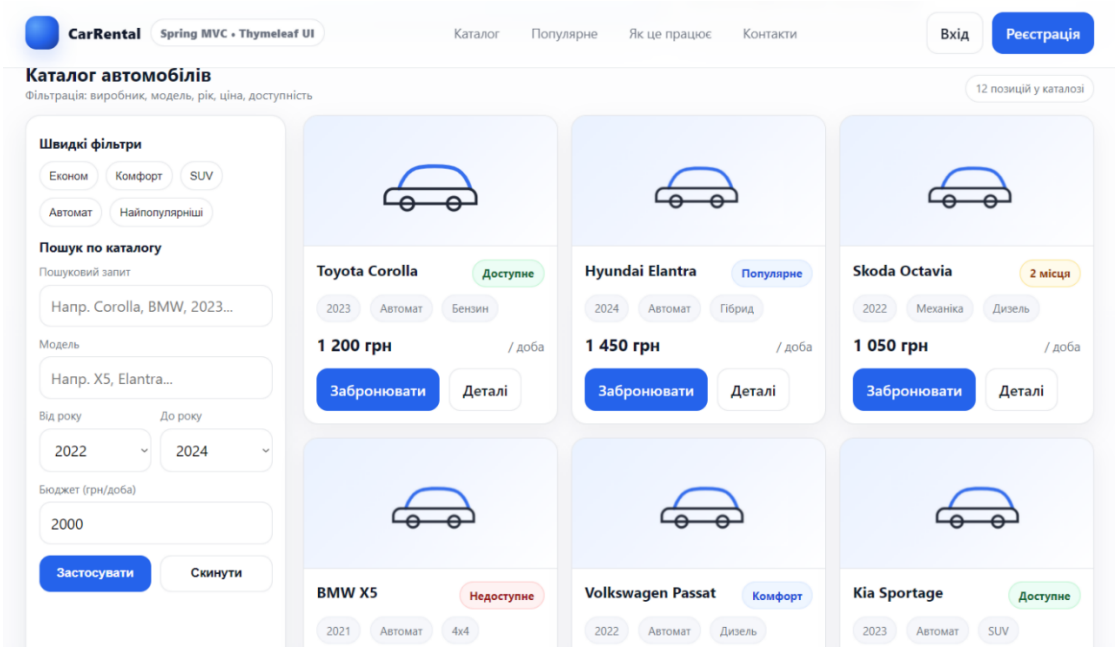


Рисунок 6 — Головна сторінка веб-застосунку

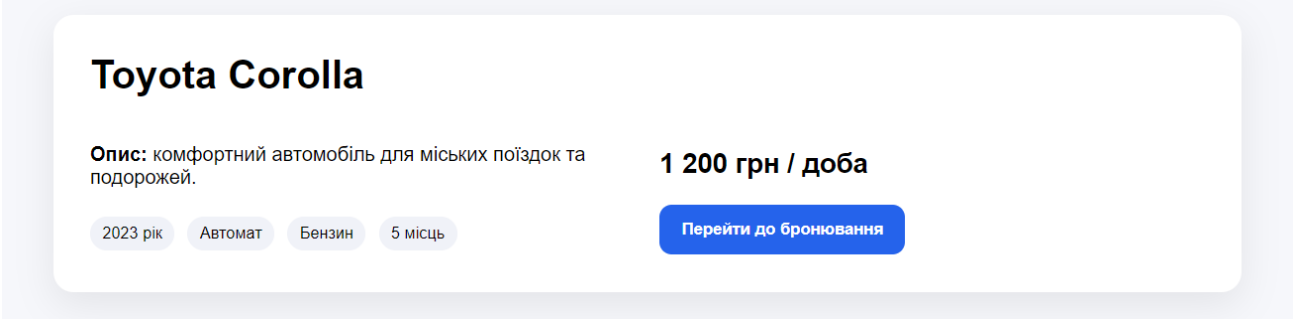


Рисунок 7 — Сторінка перегляду інформації про автомобіль

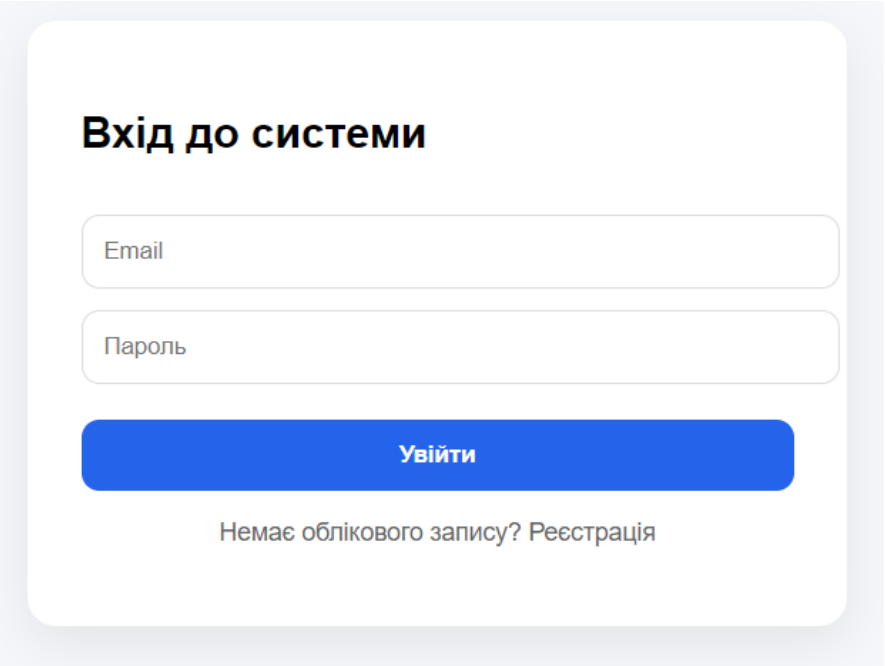


Рисунок 8 — Сторінка авторизації користувача

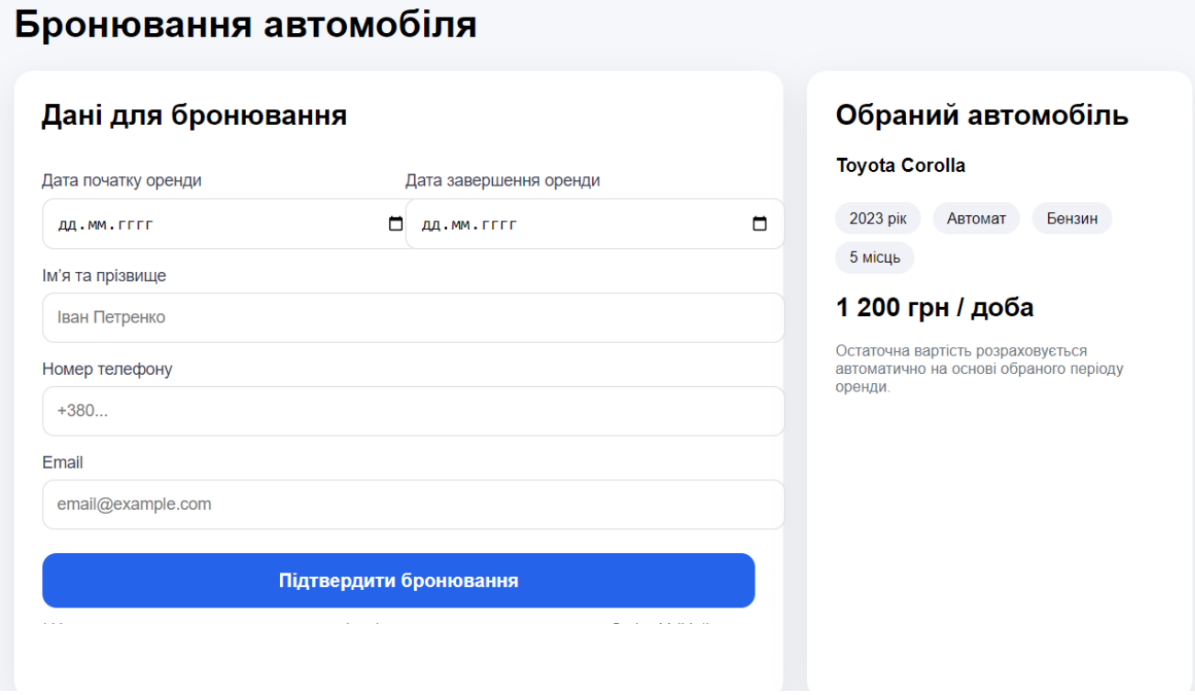


Рисунок 9 — Форма бронювання автомобіля

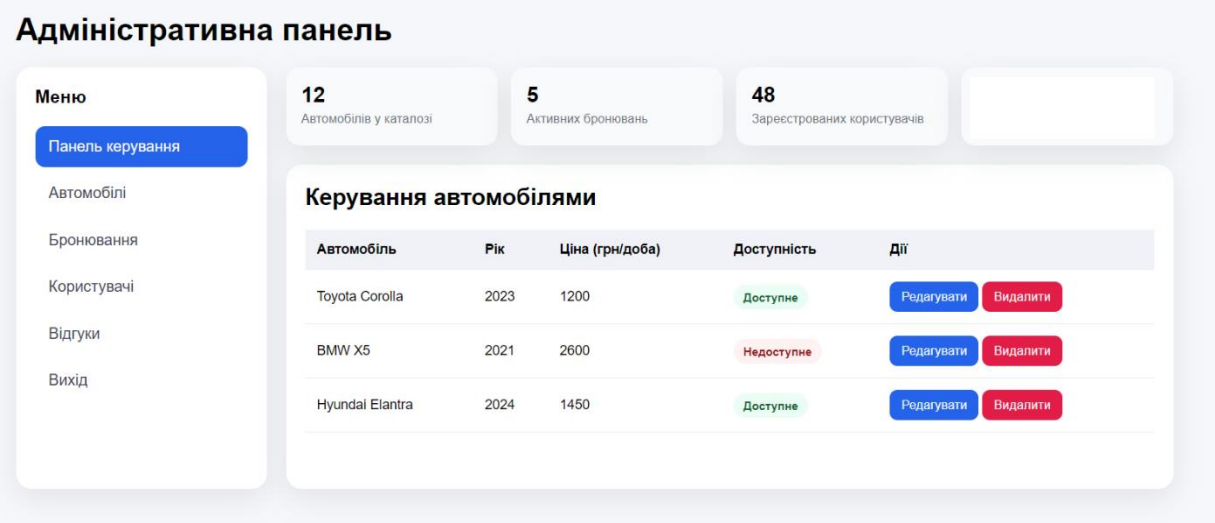


Рисунок 10 — Інтерфейс адміністратора

3.8.1 Загальна характеристика Thymeleaf

Thymeleaf — це сучасний серверний шаблонізатор Java, який перетворює HTML-шаблони у готові веб-сторінки, використовуючи дані, передані контролером. Він підтримує:

- HTML5-стандарти
- двостороннє зв’язування форми
- умовне відображення елементів
- повторення блоків
- фрагменти та компоненти
- міжнародні повідомлення (i18n)
- валідацію полів вводу
- автоматичну інтеграцію зі Spring MVC

На відміну від JSP, Thymeleaf дозволяє відкривати шаблони як звичайні HTML-файли у браузері, що полегшує розробку.

3.8.2 Інтеграція Thymeleaf зі Spring Boot

Spring Boot забезпечує повну автоконфігурацію Thymeleaf. Основні налаштування розміщені у файлі application.properties.

Папка із шаблонами за замовчуванням:

src/main/resources/templates/

Усі ресурси (CSS, зображення) розміщуються в:

src/main/resources/static/

Spring автоматично підключає ViewResolver, який відповідає за пошук HTML-файлів та їх рендеринг.

3.8.3 Передача даних з контролера до шаблону

У контролерах Spring MVC дані передаються за допомогою об’єкта Model або ModelAndView. Приклад передачі списку автомобілів із контролера до шаблону наведено у лістингу 11.

Лістинг 29 — Передача даних із контролера Spring MVC до шаблону

```
@GetMapping("/cars")
public String listCars(Model model) {
    model.addAttribute("cars", carService.findAll());
    return "cars";
}
```

У шаблоні Thymeleaf до цих даних можна звернутися через ${...}, приклад відображення даних у шаблоні наведено у лістингу 12.

Лістинг 30 — Відображення даних у шаблоні Thymeleaf

```
<tr th:each="car : ${cars}">
    <td th:text="${car.make}"></td>
    <td th:text="${car.model}"></td>
    <td th:text="${car.pricePerDay} + ' грн/доба'"></td>
```

```
</tr>
```

Таким чином забезпечується динамічне формування HTML вмісту.

3.8.4 Основні атрибути Thymeleaf

Thymeleaf використовує просту та інтуїтивну систему атрибутів:

Атрибут	**Призначення**
th:text	Вивід тексту в HTML-елемент
th:each	Цикли (перебирання колекцій)
th:if, th:unless	Умовне відображення
th:href, th:src	Динамічні посилання
th:object	Прив’язка форми до об’єкта
th:field	Прив’язка поля форми до змінної
th:value	Значення інпутів
th:errors, th:errorclass	Відображення помилок валідації

3.8.5 Робота з формами

Thymeleaf повністю інтегрується з валідацією Java через анотації (Jakarta Validation). Приклад форми бронювання автомобіля з використанням валідації наведено у лістингу 13.

Лістинг 31 — Форма бронювання автомобіля з валідацією у Thymeleaf

```
    <form        th:action="@{/book}"        th:object="${booking}"
method="post">
        <input type="date" th:field="*{startDate}" />
        <span th:errors="*{startDate}"></span>

        <input type="date" th:field="*{endDate}" />
        <span th:errors="*{endDate}"></span>
```

```
        <button type="submit">Забронювати</button>
    </form>
    У контролері:
    @PostMapping("/book")
    public String bookCar(@Valid @ModelAttribute("booking") Booking
booking,
                          BindingResult result) {
        if (result.hasErrors()) {
            return "car";
        }
        bookingService.createBooking(booking);
        return "redirect:/bookings";
    }
```

3.8.6 Використання фрагментів і шаблонів (layout)

Thymeleaf дозволяє створювати спільні частини інтерфейсу — header, footer, меню — і підключати їх до сторінок. Приклад фрагмента заголовка наведено у лістингу 32

Лістинг 32 — Фрагмент заголовка (header.html)

```
<header th:fragment="header">
    <h1>Car Rental Service</h1>
</header>
Підключення фрагмента у шаблоні:
<div th:insert="fragments/header :: header"></div>
```

Це значно скорочує дублювання коду і спрощує підтримку.

3.8.7 Валідація даних у шаблонах

Завдяки інтеграції зі Spring Validation Thymeleaf відображає помилки введення користувачем.

Валідація відповідного поля описується у класі сутності за допомогою стандартних анотацій Jakarta Validation, що наведено у лістингу 33.

Лістинг 33 — Валідація поля електронної пошти у сутності

```
<span th:if="${#fields.hasErrors('email')}"
      th:errors="*{email}">Помилка</span>
У сутності:
@NotBlank
@Email
private String email;
```

3.8.8 Переваги використання Thymeleaf у Spring Boot

- природна інтеграція зі Spring MVC
- підтримка HTML5 і адаптивної верстки
- можливість використання шаблонів без запуску сервера
- простий синтаксис
- можливість створення компонентної структури (фрагменти)
- підтримка i18n
- ідеальний для навчальних і виробничих Spring-застосунків

Thymeleaf дозволяє створювати зрозумілий, структурований і масштабований веб-інтерфейс без необхідності використання додаткових JavaScript-фреймворків, що робить його оптимальним вибором для дипломного застосунку.

3.9 Реалізація безпеки та авторизації користувачів у веб-застосунку засобами Spring Security

Spring Security є стандартним фреймворком для забезпечення автентифікації, авторизації та захисту веб-застосунків на платформі Spring. У межах проєкту система безпеки відповідає за:

- перевірку особи користувача;
- розмежування доступу між ролями (USER, ADMIN);
- захист адміністративних сторінок;
- шифрування паролів;
- автоматичну переадресацію після входу та виходу;
- обробку помилок доступу.

Spring Security інтегрується у Spring Boot 3.3 автоматично через залежність:

3.9.1 Модель безпеки та ролей

У застосунку реалізовано дві ролі користувачів:

- **USER** – може переглядати автомобілі, створювати бронювання, писати відгуки;
- **ADMIN** – має доступ до адміністративної панелі, може додавати, змінювати й видаляти автомобілі.

Ролі зберігаються у таблиці **roles**, а зв’язок користувачів з ролями — у таблиці **user_roles** (зв’язок M:N).

Сутність User імплементує інтерфейс UserDetails, що дозволяє Spring Security отримувати дані користувача, як показано у лістингу 19.

Лістинг 34 — Реалізація інтерфейсу UserDetails у сутності User

```
@Entity
public class User implements UserDetails {
    @ManyToMany(fetch = FetchType.EAGER)
    private Set<Role> roles;
```

```
        @Override
        public     Collection<?     extends     GrantedAuthority>
getAuthorities() {
            return roles.stream()
                .map(role                  ->                  new
SimpleGrantedAuthority(role.getName()))
                .toList();
        }
        @Override
        public String getUsername() {
            return email;
      }
    }
```

3.9.2 Реалізація UserDetailsService

Для завантаження користувачів із бази даних використовується власний сервіс, наведена у лістингу 35.

Лістинг 35 — Реалізація сервісу UserDetailsService

```
    @Service
    public     class     CustomUserDetailsService     implements
UserDetailsService {
        private final UserRepository userRepository;
        @Override
        public UserDetails loadUserByUsername(String email)
                throws UsernameNotFoundException {
            return userRepository.findByEmail(email)
                .orElseThrow(() ->
                    new  UsernameNotFoundException("Користувача  не
знайдено"));
```

Цей сервіс використовується Spring Security під час автентифікації.

3.9.3 Шифрування паролів

Усі паролі користувачів зберігаються у зашифрованому вигляді за допомогою алгоритму **BCrypt**, що забезпечує стійкість до атак методом перебору, конфігурація якого наведена у лістингу 36.

Лістинг 36 — Налаштування PasswordEncoder

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Під час реєстрації:

```
user.setPassword(passwordEncoder.encode(user.getPassword()));
userRepository.save(user);
```

3.9.4 Конфігурація Spring Security

Spring Boot 3.x використовує новий підхід до налаштування безпеки через SecurityFilterChain, приклад якого наведено у лістингу 37.

Лістинг 37 — Конфігурація Spring Security

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/", "/cars/**", "/login",
"/register", "/css/**").permitAll()
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
```

```
                )
                .formLogin(login -> login
                    .loginPage("/login")
                    .defaultSuccessUrl("/", true)
                    .permitAll()
                )
                .logout(logout -> logout
                    .logoutUrl("/logout")
                    .logoutSuccessUrl("/")
                    .permitAll()
                );
            return http.build();
        }
    }
```

Пояснення:

- **permitAll()** — сторінки, доступні без авторизації;
- **hasRole("ADMIN")** — доступ лише адміністраторам;
- **authenticated()** — будь-який авторизований користувач;
- **formLogin()** — використовує кастомну сторінку логіну;
- **logout()** — очищає сесію та cookie;
- **csrf.disable()** — дозволяє тестувати застосунок локально.

3.9.5 Захист адміністративних сторінок

Усі маршрути /admin/** доступні тільки користувачам із роллю ADMIN:

.requestMatchers("/admin/**").hasRole("ADMIN")

Spring автоматично поверне сторінку 403, якщо користувач не має достатніх прав.

3.9.6 Сторінка входу та виходу

Spring Boot дозволяє визначити власну сторінку логіну, наведено у лістингу 38.

Лістинг 38 — HTML-форма входу користувача

```
.formLogin(login -> login
    .loginPage("/login")
    .defaultSuccessUrl("/", true)
```

HTML-шаблон login.html містить форму:

```
<form th:action="@{/login}" method="post">
    <input type="text" name="username" placeholder="Email">
    <input type="password" name="password"
placeholder="Пароль">
    <button type="submit">Увійти</button>
</form>
Для виходу використовується URL /logout:
<a th:href="@{/logout}">Вийти</a>
```

3.9.7 Обробка помилок авторизації

Spring автоматично перенаправляє неавторизованих користувачів на сторінку логіну.

Помилки типу 403 Access Denied можна відловити:

```
.exceptionHandling((exceptions) ->
    exceptions.accessDeniedPage("/access-denied"));
```

3.9.8 Переваги використання Spring Security

- захист від несанкціонованого доступу
- надійний механізм шифрування паролів
- автоматична інтеграція із Spring MVC
- гнучка система ролей і прав
- можливість розширення OAuth2, JWT, SSO
- високий рівень безпеки без додаткового коду

Spring Security дозволяє створити надійну систему доступу та забезпечує повну відповідність веб-застосунку сучасним вимогам захисту інформації.

3.10 Висновки до розділу 3

Спроєктовано та реалізовано веб-застосунок системи оренди автомобілів із використанням фреймворків Spring Boot, Spring MVC, Spring Data JPA та шаблонізатора Thymeleaf. У процесі розроблення сформовано багаторівневу архітектуру застосунку, побудовану за шаблоном Model–View–Controller з чітким розмежуванням відповідальності між окремими програмними шарами.

Реалізовано основні функціональні можливості системи, зокрема перегляд каталогу автомобілів, створення та скасування бронювань, а також адміністративне керування автопарком і користувачами. Забезпечено автентифікацію та авторизацію користувачів із використанням механізмів Spring Security, що дозволяє обмежувати доступ до функціоналу відповідно до ролей.

Застосування сучасного стеку технологій та принципів модульної побудови забезпечує коректну роботу веб-застосунку, спрощує його супровід і створює передумови для подальшого розширення функціональних можливостей системи.

Розділ 4 Конструювання застосунку із використанням Spring Boot, Spring MVC, Spring Data JPA та Thymeleaf

4.1 Загальні підходи до конструювання веб-застосунку

На етапі конструювання здійснюється безпосередня реалізація програмних компонентів, описаних у попередньому розділі: сутностей доменної моделі, репозиторіїв, сервісів, контролерів, конфігурації безпеки та шаблонів представлення.

Для розробки обрано стек технологій:

- Spring Boot 3.x – основа застосунку, автоконфігурація, запуск вбудованого сервера;
- Spring MVC – обробка HTTP-запитів та організація шарів контролерів і представлення;
- Spring Data JPA – доступ до бази даних MySQL через ORM-шар;
- Thymeleaf – формування динамічних HTML-сторінок;
- Spring Security – реалізація автентифікації та авторизації на основі ролей USER/ADMIN.

Усі компоненти організовано у вигляді багаторівневої архітектури (Model–Repository–Service–Controller–View), що забезпечує низьку зв’язаність між класами, можливість модульного тестування та подальшого масштабування системи.

Структура пакунків проєкту має вигляд:

- com.carrental.car_rental – кореневий пакет;
- config – конфігураційні класи (безпека, ініціалізація даних);
- entity – сутності JPA;
- repository – інтерфейси репозиторіїв;
- service та service.impl – інтерфейси сервісів та їх реалізації;

- controller – веб-контролери;
- security – розширення для Spring Security.

4.2 Створення та конфігурація Spring Boot-проєкту

При створенні застосунку було використано Spring Initializr — стандартний інструмент для генерації структури проєкту. На етапі конфігурації обираються необхідні модулі:

- **Spring Web** — для роботи з HTTP-протоколом, контролерами та маршрутизацією;
- **Spring Data JPA** — для взаємодії із базою даних через ORM;
- **Spring Security** — для автентифікації та авторизації;
- **Thymeleaf** — для шаблонізації HTML-сторінок;
- **MySQL Driver** — для можливості підключення до реляційної БД.

Після генерації проєкту Spring Boot автоматично створює мінімальну структуру та основний клас запуску, що дозволяє уникнути складної конфігурації та зменшує кількість boilerplate-коду.

Важливим елементом під час налаштування є файл application.yml, у якому визначаються параметри підключення до бази даних, режими роботи JPA та налаштування веб-сервера. Саме завдяки Spring Boot більшість параметрів конфігуруються автоматично, що значно прискорює процес розробки, приклад якого наведено у лістингу 39.

Лістинг 39 — Файл конфігурації application.yml

```
spring:
 datasource:
   url: jdbc:mysql://localhost:3306/car_rental_db
   username: root
```

```
    password: root

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
ddl-auto: update — Spring сам створює та оновлює таблиці;

show-sql: true — показ SQL у консолі.
```

4.3 Реалізація доменної моделі

Доменна модель описує сутності предметної області, їхні властивості та зв’язки. У застосунку такі сутності як користувач, роль, автомобіль, бронювання, відгук були визначені ще на етапі аналізу, що дозволило перейти до їхнього програмного представлення.

4.3.1 Принципи створення JPA-сутностей

При конструюванні сутностей використовувалися такі підходи:

- **Анотація @Entity** визначає клас як об’єкт, що зберігатиметься в БД;
- **Анотація @Table** задає назву таблиці;
- **Анотація @Id** та стратегії генерації ключів забезпечують унікальність записів;
- **Анотації зв’язків (@OneToMany, @ManyToOne, @ManyToMany, @OneToOne)** забезпечують зв’язність між сутностями відповідно до ER-моделі;
- Усі сутності містять мінімальні набори полів, необхідних для роботи системи.

4.3.2 Пояснення ключових сутностей

- **User** — зберігає особисті дані, логін та пароль користувача. Користувач може мати декілька ролей, що дозволяє реалізувати різні рівні доступу.
- **Car** — представляє автомобіль, доступний для оренди. Містить опис, рік випуску, ціну оренди та інші метадані.
- **Booking** — фіксує операцію бронювання, включаючи дати оренди, статус, користувача та автомобіль.
- **Review** — надає можливість користувачам залишати відгуки про автомобілі.

4.4 Реалізація рівня доступу до даних (Repository Layer)

Шар доступу до даних базується на Spring Data JPA, що дозволяє:

- автоматично генерувати SQL-запити;
- створювати методи пошуку на основі імен (наприклад, findByAvailableTrue());
- мінімізувати кількість коду, необхідного для роботи з БД.

Репозиторії є інтерфейсами, які успадковують JpaRepository, тому Spring створює для них реалізацію автоматично. Такий підхід дозволяє зосередитися на логіці програми, а не на деталях збереження даних.

Spring Data JPA дозволяє не писати SQL вручну — репозиторій це робить автоматично. Приклад основного репозиторію для роботи з сутністю автомобіля наведено у лістингу 40.

Лістинг 40 — Інтерфейс CarRepository

```
public interface CarRepository extends JpaRepository<Car, Long>
{
    List<Car> findByAvailableTrue();
```

```
}
Spring сам генерує запит:
SELECT * FROM cars WHERE available = true;
```

4.5 Реалізація бізнес-логіки (Service Layer)

Сервісний шар є ключовим у структурі застосунку, адже він містить усю бізнес-логіку. Саме тут виконується:

- перевірка доступності автомобіля у вибраний період;
- обробка помилок користувача;
- зміна статусів бронювання;
- призначення ролей користувачам;
- взаємодія між декількома репозиторіями;
- реалізація транзакцій.

Приклади бізнес-процесів:

1. Створення бронювання показано у лістингу

Під час створення бронювання система:

1. Перевіряє існування користувача;
2. Перевіряє існування автомобіля;
3. Перевіряє коректність дат;
4. Перевіряє перетин бронювань;
5. Створює об’єкт Booking;
6. Записує його в базу даних.

2. Скасування бронювання

Скасувати бронювання може лише той користувач, який його створив. Це забезпечується перевіркою ідентифікатора в сервісі.

Лістинг 41 - Створення бронювання:

```
@Service
public class BookingServiceImpl implements BookingService {

    public Booking create(Long userId, Long carId,
                          LocalDate start, LocalDate end) {

        if (end.isBefore(start))
            throw new IllegalArgumentException("Дата завершення
раніше початку");

        boolean busy = bookingRepository
            .findByCarIdAndStatus(carId, BookingStatus.ACTIVE)
            .stream()
            .anyMatch(b -> !(end.isBefore(b.getStartDate()) ||
                             start.isAfter(b.getEndDate())));

        if (busy)
            throw new IllegalStateException("Авто недоступне у
вибраний період");

        Booking booking = new Booking();

booking.setUser(userRepository.getReferenceById(userId));
        booking.setCar(carRepository.getReferenceById(carId));
        booking.setStartDate(start);
        booking.setEndDate(end);

        return bookingRepository.save(booking);
    }
}
```

4.6 Реалізація контролерів Spring MVC

Контролери відповідають за обробку веб-запитів та передачу даних до шаблонів.

Застосунок містить такі основні контролери:

- **HomeController** — відображає головну сторінку та список автомобілів;
- **CarController** — відображає детальну інформацію про автомобіль та приймає форму бронювання;
- **BookingController** — відображає сторінку особистих бронювань користувача;
- **AdminCarController** — забезпечує інтерфейс адміністратора для керування автопарком.

У лістингу 42 показано контролери, які використовують сервісний шар і не містять бізнес-логіки, що відповідає принципу “тонких контролерів”.

Лістинг 42 - Контролер головної сторінки

```
@Controller
public class HomeController {

    @GetMapping("/")
    public String index(Model model) {
        model.addAttribute("cars",
carService.findAllAvailable());
        return "index";
```

4.7 Конструювання інтерфейсу користувача на основі Thymeleaf

Thymeleaf забезпечує можливість вставляти динамічні дані у HTML-сторінки.

При розробці інтерфейсу застосовано такі принципи:

- використання циклів (th:each) для відображення списків автомобілів;
- використання атрибутів th:object та th:field для форм;
- підключення спільних елементів (хедер, меню) через фрагменти;
- дата-поля для бронювання контрольовані на рівні браузера.

Основні шаблони:

- *index.html* — список автомобілів;
- *car.html* — опис автомобіля та форма бронювання;
- *bookings.html* — таблиця бронювань користувача;
- *admin/cars.html* — CRUD для автомобілів.

4.8 Реалізація авторизації та автентифікації Spring Security

Spring Security забезпечує контроль доступу до частин застосунку:

- сторінки адміністратора доступні лише ролі **ADMIN**;
- приватні сторінки (бронювання) доступні лише авторизованим користувачам;
- публічні сторінки (головна, сторінка автомобіля) доступні всім.

Алгоритм автентифікації:

1. користувач вводить email та пароль;
2. система завантажує користувача за email;
3. Spring автоматично перевіряє пароль;
4. користувач отримує відповідні ролі;
5. на основі ролей формується доступ до необхідних URL.

Основна конфігурація доступів показана у лістингу 43.

Лістинг 43 – Конфігурація доступів

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {

    http.authorizeHttpRequests(auth -> auth
        .requestMatchers("/", "/cars/**").permitAll()
        .requestMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated()
    );

    http.formLogin(login -> login.loginPage("/login"));
    return http.build();
}
```

4.9 Ініціалізація тестових даних

Для демонстрації роботи застосунку під час запуску автоматично створюються:

- ролі ADMIN та USER;
- обліковий запис адміністратора;
- базовий набір тестових автомобілів;
- декілька прикладів бронювань.

Для автоматичної ініціалізації тестових даних під час запуску застосунку використовується компонент DataInitializer, приклад якого наведено у лістингу 44.

Лістинг 44 — Ініціалізація тестових даних при старті застосунку’

```
@Component
public class DataInitializer implements CommandLineRunner {
    @Override
    public void run(String... args) {
        // створення ролей, адміністратора та тестових авто
```

```
        }
    }
```

Це спрощує тестування та дозволяє перевірити всі функції без ручного наповнення БД.

4.10 Висновки до 4 розділу

Здійснено конструювання веб-застосунку для оренди автомобілів із використанням фреймворку Spring Boot та пов’язаних з ним технологій Spring MVC, Spring Data JPA, Spring Security і шаблонізатора Thymeleaf. Реалізація застосунку виконана відповідно до багаторівневої архітектури, що забезпечує чітке розмежування відповідальності між шарами моделі даних, доступу до даних, бізнес-логіки, контролерів і представлення.

Реалізовано доменну модель предметної області у вигляді JPA-сутностей, які відображають основні об’єкти системи оренди автомобілів та їх взаємозв’язки. Шар доступу до даних побудовано з використанням Spring Data JPA, що дозволило мінімізувати обсяг низькорівневого коду та забезпечити зручну взаємодію з реляційною базою даних MySQL.

Сервісний рівень забезпечує реалізацію основних бізнес-процесів системи, зокрема перевірку доступності автомобілів, створення та скасування бронювань, а також контроль коректності введених даних. Контролери Spring MVC реалізують обробку HTTP-запитів і взаємодію з користувацьким інтерфейсом, дотримуючись принципу «тонких контролерів».

Інтерфейс користувача сформовано з використанням Thymeleaf, що забезпечує динамічне відображення даних та повторне використання елементів сторінок. Захист веб-застосунку реалізовано за допомогою Spring Security, що дозволяє здійснювати автентифікацію та авторизацію користувачів відповідно до ролей і обмежувати доступ до функціональних можливостей системи.

Для демонстрації роботи застосунку реалізовано механізм ініціалізації тестових даних, який спрощує перевірку функціоналу без додаткового налаштування бази даних. Отримані результати підтверджують коректність обраних підходів до конструювання веб-застосунку та доцільність використання технологій Spring для реалізації систем оренди автомобілів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено комплексне дослідження засобів фреймворку Spring для розробки веб-застосунків на платформі Java та реалізовано повнофункціональний веб-застосунок для оренди автомобілів.

У роботі проаналізовано сучасні підходи до веб-розробки на платформі Java, розглянуто еволюцію Java-вебтехнологій і обґрунтовано доцільність використання екосистеми Spring як основи для створення масштабованих, надійних і безпечних веб-систем. Детально досліджено ключові компоненти Spring Framework, зокрема Spring Boot, Spring MVC, Spring Data JPA, Spring Security та Thymeleaf, а також їх взаємодію у межах багаторівневої архітектури застосунку.

У процесі проєктування було розроблено архітектуру веб-застосунку на основі шаблону Model–View–Controller (MVC) з чітким розмежуванням відповідальності між рівнями представлення, бізнес-логіки та доступу до даних. Спроєктовано реляційну базу даних MySQL, визначено основні сутності предметної області, їх атрибути та зв’язки, що забезпечує коректне зберігання і обробку інформації.

У практичній частині роботи реалізовано веб-застосунок для оренди автомобілів із підтримкою реєстрації та автентифікації користувачів, ролей доступу, перегляду каталогу автомобілів, створення та керування бронюваннями, а також адміністративного керування автопарком. Для забезпечення безпеки застосунку впроваджено механізми автентифікації й авторизації засобами Spring Security, з використанням ролей користувачів та шифрування паролів.

Розроблений веб-застосунок відповідає сучасним вимогам до функціональності, безпеки та масштабованості й може бути використаний як

основа для подальшого розширення, зокрема впровадження REST API, інтеграції з платіжними сервісами або переходу до мікросервісної архітектури.

Отримані результати підтверджують ефективність використання фреймворку Spring для створення сучасних веб-застосунків на платформі Java та демонструють практичну цінність обраних архітектурних і технологічних рішень. Розроблений проєкт може бути використаний як навчальний приклад, а також як прототип реальної інформаційної системи для сервісів оренди автомобілів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Walls C. Spring in Action. 6th ed. Manning Publications, 2022. 520 p.

2. Смирнов Є., Коломоєць Г.П. Дослідження засобів фреймворку Spring для розробки веб-застосунків на платформі Java. Збірник наукових праць студентів, здобувачів PhD, док-торантів і молодих вчених «Молода наука-2025» / Запорізький національний університе-тет. – Запоріжжя : ЗНУ, 2025. – Т.5. – С. 415-416.

3. Johnson R., Hoeller J., Arendsen A., Thomas R. Expert One-on-One J2EE Design and Development. Wrox Press, 2002. 1184 p.

4. Long J. Spring Microservices in Action. Manning Publications, 2021. 400 p.

5. Webb P., Syer D., Long J., Nicoll S. Spring Boot Reference Documentation. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 15.04.2025)

6. Spring Framework Documentation. URL: https://docs.spring.io/spring-framework/reference/ (дата звернення: 12.04.2025)

7. Spring Data JPA Reference Documentation. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (дата звернення: 05.05.2025)

8. Spring Security Reference Documentation. URL: https://docs.spring.io/spring-security/reference/ (дата звернення: 18.06.2025)

9. Hibernate ORM Documentation. URL: https://hibernate.org/orm/documentation/ (дата звернення: 10.05.2025)

10. Jakarta Persistence API Specification. URL: https://jakarta.ee/specifications/persistence/ (дата звернення: 07.05.2025)

11. MySQL 8.0 Reference Manual. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 02.05.2025)

12. Thymeleaf Documentation.

U\RL: https://www.thymeleaf.org/documentation.html (дата звернення: 22.06.2025)

13. Baeldung. Spring Boot Tutorials.

URL: https://www.baeldung.com/spring-boot (дата звернення: 01.07.2025)

14. Baeldung. Spring Security Guides.

URL: https://www.baeldung.com/security-spring (дата звернення: 05.07.2025)

15. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.

16. Oracle. Java Platform, Standard Edition 17 Documentation.

URL: https://docs.oracle.com/en/java/javase/17/ (дата звернення: 01.04.2025)

17. Maven Documentation.

URL: https://maven.apache.org/guides/ (дата звернення: 20.05.2025)

18. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 533 p.

19. Bloch J. Effective Java. 3rd ed. Addison-Wesley, 2018. 416 p.

20. OWASP. Web Application Security Top 10.

URL: https://owasp.org/www-project-top-ten/ (дата звернення: 30.06.2025)

21. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation. University of California, 2000

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Смирнов Єгор Теімуразович, студент 2 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальності 121 «Інженерія програмного забезпечення», адреса електронної пошти:ipz20bd-112@stu.zsea.edu.ua,

підтверджую, що написана мною кваліфікаційна робота на тему «Дослідження засобів фреймворку Spring для розробки Веб-застосунків на платформі Java» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений.

Заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії.

Також згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата: Підпис: Смирнов Єгор Теімуразович
(студент)

Дата: Підпис: Коломоєць Геннадій Павлович
(науковий керівник)