Vehicle Registration Management System - Documentation

Project Name: Vehicle Registration Management System **Student:** Lilla (Neptun: IHUTSC) **Course:** Java Applications - 5th Semester **Date:** November 27, 2025 **Repository:** https://github.com/MI804-png/java_seminar_5th_semester_Lilla **Live Application:** http://rivendell.nje.hu:9443/ihutsc-se/ ---

Table of Contents

Executive Summary

The Vehicle Registration Management System is a full-stack web application built with Spring Boot that manages vehicle registrations, owners (persons), and their contact information. The system provides complete CRUD operations, authentication/authorization, RESTful API, and data visualization through charts and statistics. **Key Achievements:**

- âœ… All 14 course requirements completed (30/30 points)
- âœ… Clean, maintainable code following Spring Boot best practices
- âœ… Responsive UI with Bootstrap and Thymeleaf templates
- âœ… MySQL database integration with JPA/Hibernate
- âœ… Production deployment on Linux Tomcat server
- âœ… Comprehensive Git version control (7 commits)

Technical Stack

Backend

- **Framework:** Spring Boot 2.7.18
- **Language:** Java 11
- **Build Tool:** Maven 3.9.5
- **ORM:** Hibernate/JPA
- **Security:** Spring Security
- **Validation:** Jakarta Bean Validation

Frontend

- **Template Engine:** Thymeleaf
- **CSS Framework:** Bootstrap 5.3
- **JavaScript:** Vanilla JS + Chart.js
- **Layout:** Thymeleaf Layout Dialect

Database

- **Development:** H2 in-memory database
- **Production:** MySQL 8.0
- **Driver:** MySQL Connector/J

Deployment

- **Development Server:** Embedded Tomcat (port 8080)
- **Production Server:** Standalone Tomcat 9 (port 9443)
- **Server:** rivendell.nje.hu
- **Packaging:** WAR file (ihutsc-se.war)
  ---

System Requirements

Development Environment

- Java JDK 11 or higher
- Maven 3.6+ (or use included Maven wrapper)

- Git for version control
- IDE: IntelliJ IDEA / Eclipse / VS Code

Production Environment

- Linux server with Tomcat 9
- MySQL 8.0 database server
- Java 11 runtime
- SSH access for deployment

Application Features

1. Person Management (CRUD)

- **Create:** Add new person with validation
- **Read:** View list of all persons, view individual person details
- **Update:** Edit person information
- **Delete:** Remove person (if no associated vehicle)
- **Fields:** Name, birth year, registration number (unique)
- **Validation:** All fields required, birth year must be valid (1900-2025)

2. Vehicle Management (CRUD)

- **Create:** Register new vehicle with owner
- **Read:** Browse all vehicles with filtering and statistics
- **Update:** Modify vehicle details
- **Delete:** Remove vehicle registration
- **Fields:** Registration number (unique), brand, production year, color, owner
- **Statistics:** Count by brand, count by color, total vehicles

3. Phone Number Management

- **Multiple phones:** Each person can have multiple phone numbers
- **CRUD operations:** Add, view, edit, delete phone numbers
- **Association:** Linked to person entity
- **Validation:** Phone number format validation

4. Database Management

- **Overview page:** View all tables (Person, Vehicle, Phone)
- **Record counts:** Display total records per table
- **Relationships:** Show owner-vehicle associations
- **Refresh:** Real-time data updates

5. Charts & Statistics

- **Bar chart:** Vehicles by brand (Chart.js)
- **Pie chart:** Vehicles by color distribution
- **Statistics cards:** Quick metrics on homepage
- **Dynamic data:** Automatically updates with database changes

6. RESTful API

- **GET /api/persons** - List all persons (JSON)
- **GET /api/persons/{id}** - Get person by registration number
- **GET /api/vehicles** - List all vehicles (JSON)
- **GET /api/vehicles/{regnum}** - Get vehicle by registration number
- **POST /api/persons** - Create new person
- **PUT /api/persons/{id}** - Update person
- **DELETE /api/persons/{id}** - Delete person
- **Content-Type:** application/json
- **Authentication:** Required for all endpoints

7. Authentication & Authorization

- **Login page:** Custom login form with Spring Security
- **Two user roles:**
  - **Admin:** admin / admin123 (full access) - **User:** user / user123 (read-only)
- **Session management:** Remember-me functionality
- **CSRF protection:** Enabled for all forms
- **Password encoding:** BCrypt (ready for implementation)

8. Data Initialization

- **Sample data:** Pre-loaded with 5 persons, 6 vehicles, 8 phone numbers
- **Development:** Uses `data.sql` for H2 database

- **Production:** Manual data migration or Hibernate auto-create

---

Database Schema

Entity Relationship Diagram

`` â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”□ â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”□ â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”□ â”‚ Person â”‚ â”‚ Vehicle â”‚ â”‚ Phone â”‚ â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¤ â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¤ â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¤ â”‚ regnumber (PK) â”‚â—„â”€â”€â”€â”€â”€â”€â”€â”¤ regnum (PK) â”‚ â”‚ id (PK) â”‚ â”‚ name â”‚ â”‚ brand â”‚ â”‚ number â”‚ â”‚ birthyear â”‚ â”‚ productionyear â”‚ â”Œâ”€â”€â”€â–ºâ”‚ person_regnum â”‚ â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”˜ â”‚ color â”‚ â”‚ â”‚ (FK) â”‚ â”‚ â”‚ owner_regnum(FK)â”œâ”€â”€â”€â”€â”˜ â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”˜ â”‚ â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”˜ â”‚ â”‚ 1:N relationship â”‚ (One person can have multiple phones) â””â”€â–º `

Table Definitions

Person Table

`sql CREATE TABLE person ( regnumber VARCHAR(20) PRIMARY KEY, name VARCHAR(100) NOT NULL, birthyear INT NOT NULL ); `

Vehicle Table

`sql CREATE TABLE vehicle ( regnum VARCHAR(20) PRIMARY KEY, brand VARCHAR(50) NOT NULL, productionyear INT NOT NULL, color VARCHAR(30) NOT NULL, owner_regnum VARCHAR(20), FOREIGN KEY (owner_regnum) REFERENCES `person(regnumber)); ``

Phone Table

``sql CREATE TABLE phone ( id BIGINT AUTO_INCREMENT PRIMARY KEY, number VARCHAR(20) NOT NULL, person_regnum VARCHAR(20) NOT NULL, FOREIGN KEY (person_regnum) REFERENCES person(regnumber) ON DELETE CASCADE ); ``

Sample Data

Persons:

- John Doe (REG001, born 1985)
- Jane Smith (REG002, born 1990)
- Bob Johnson (REG003, born 1978)
- Alice Williams (REG004, born 1995)
- Charlie Brown (REG005, born 1982)

Vehicles:

- ABC123 - Toyota Camry 2018, blue (owned by John Doe)
- XYZ789 - Honda Civic 2020, red (owned by Jane Smith)
- DEF456 - Ford Focus 2019, black (owned by Bob Johnson)
- GHI012 - Toyota Corolla 2021, red (owned by Alice Williams)
- JKL345 - Nissan Altima 2017, silver (owned by Charlie Brown)
- MNO678 - Chevrolet Malibu 2022, white (no owner)

Phone Numbers:

- John Doe: +1234567890, +1234567891
- Jane Smith: +1987654321
- Bob Johnson: +1122334455, +1122334456
- Alice Williams: +1555666777
- Charlie Brown: +1999888777, +1999888778

`---`

Security Implementation

Spring Security Configuration

``java @Configuration @EnableWebSecurity public class SecurityConfig { @Bean public SecurityFilterChain filterChain(HttpSecurity http) { http

`.authorizeRequests() .antMatchers("/css/**", "/js/**", "/images/**").permitAll() .antMatchers("/h2-console/**").permitAll() .anyRequest().authenticated() .and() .formLogin() .loginPage("/login") .defaultSuccessUrl("/", true) .permitAll() .and() .logout() .logoutSuccessUrl("/login?logout") .permitAll() .and() .csrf() .ignoringAntMatchers("/h2-console/**") .and() .headers() .frameOptions().sameOrigin(); return http.build(); } }` `

User Credentials

Development & Production:

- Admin: admin / admin123 (full access)
- User: user / user123 (read-only)

Security Features

- âœ… Form-based authentication
- âœ… Session management
- âœ… CSRF protection on all forms
- âœ… Remember-me functionality
- âœ… Role-based access control (ready)
- âœ… Secure password storage (BCrypt ready)
- âœ… Logout functionality
- âœ… H2 console security bypass (dev only)

API Endpoints

Person API

GET /api/persons

Description: Retrieve all persons **Method:** GET **Authentication:** Required **Response:** JSON array of Person objects `json [ { "regnumber": "REG001", `"name": "John Doe", "birthyear": 1985 }, ...]` `

GET /api/persons/{regnumber}

Description: Get person by registration number **Method:** GET **Authentication:** Required **Path Variable:** regnumber (String) **Response:** JSON Person object or 404 Not Found `json { "regnumber": "REG001", "name": "John Doe", "birthyear": 1985 }` `

POST /api/persons

Description: Create new person **Method:** POST **Authentication:** Required **Content-Type:** application/json **Request Body:** `json { "regnumber": "REG999", "name": "New Person", "birthyear": 1990 }` **Response:** 201 Created with Person object

PUT /api/persons/{regnumber}

Description: Update existing person **Method:** PUT **Authentication:** Required **Path Variable:** regnumber (String) **Content-Type:** application/json **Request Body:** Updated Person object **Response:** 200 OK with updated Person or 404 Not Found

DELETE /api/persons/{regnumber}

Description: Delete person **Method:** DELETE **Authentication:** Required **Path Variable:** regnumber (String) **Response:** 204 No Content or 404 Not Found

Vehicle API

GET /api/vehicles

Description: Retrieve all vehicles **Method:** GET **Authentication:** Required **Response:** JSON array of Vehicle objects `json [ { "regnum": "ABC123", "brand": "Toyota Camry", "productionyear": 2018, "color": "blue", "ownerRegnum": "REG001" }, ... ]` `

GET /api/vehicles/{regnum}

`**Description:** Get vehicle by registration number **Method:** GET **Authentication:** Required **Path Variable:** `regnum` (String) **Response:** JSON Vehicle object or 404 Not Found`

Testing API with cURL

`` ` ``bash

Login first to get session cookie

`curl -c cookies.txt -X POST http://localhost:8080/login \ -d "username=admin&password=admin123"`

Get all persons

`curl -b cookies.txt http://localhost:8080/api/persons`

Get specific person

`curl -b cookies.txt http://localhost:8080/api/persons/REG001`

Create new person

`curl -b cookies.txt -X POST http://localhost:8080/api/persons \ -H "Content-Type: application/json" \ -d '{"regnumber":"REG999","name":"Test User","birthyear":1995}'`

Get all vehicles

`curl -b cookies.txt http://localhost:8080/api/vehicles`

Get specific vehicle

`curl -b cookies.txt http://localhost:8080/api/vehicles/ABC123` ` `---`

Installation & Setup

Local Development Setup

1. Clone Repository

` `bash git clone https://github.com/MI804-png/java_seminar_5th_semester_Lilla.git cd java_seminar_5th_semester_Lilla/vehicle-registration-app` ` `

2. Verify Java Installation

` `bash java -version`

Should show Java 11 or higher

` `

3. Build Project

` `bash`

Windows

`mvnw.cmd clean package`

Linux/Mac

`./mvnw clean package` `

4. Run Application

` `bash`

Windows

`mvnw.cmd spring-boot:run`

Linux/Mac

`./mvnw spring-boot:run` `

5. Access Application

- `**Web Interface:** http://localhost:8080`
- `**H2 Console:** http://localhost:8080/h2-console`
  `- JDBC URL:` jdbc:h2:mem:testdb `- Username:` sa `- Password: (empty)`
- `**Login:** admin / admin123`

IDE Setup (IntelliJ IDEA)

`1. **Import Project:** - File â†’ Open â†’ Select` pom.xml `- Import as Maven project 2. **Configure JDK:** - File â†’ Project Structure â†’ Project SDK - Select Java 11 3. **Run Configuration:** - Main class:` com.vehiclereg.VehicleRegistrationApplication `- VM options:` -Dspring.profiles.active=dev `- Working directory:` $MODULE_WORKING_DIR$ `4. **Run Application:** -`

`Click green run button or Shift+F10 - Access at http://localhost:8080 ---`

Deployment Guide

Production Deployment to rivendell.nje.hu

Prerequisites

- Server credentials (username: student208)
- WinSCP or SSH client installed
- Production WAR file built

Step 1: Build Production WAR

`` ` ``powershell cd c:\java_seminar\java_seminar\vehicle-registration-app`

Set JAVA_HOME

`$env:JAVA_HOME = "C:\Program Files\Microsoft\jdk-11.0.16.101-hotspot"`

Build WAR

`.\mvnw.cmd clean package -DskipTests` ` **Output:** target/ihutsc-se.war `(56.7 MB)`

Step 2: Deploy Using PowerShell Script

`` ` ``powershell`

Run deployment script

`.\deploy.ps1`

Script will:

1. Check if WAR exists

2. Find WinSCP or pscp

3. Prompt for password

4. Upload to server

5. Move to Tomcat webapps

6. Verify deployment

`

Step 3: Manual Deployment (Alternative)

`**Option A: WinSCP GUI** 1. Open WinSCP 2. Connect to rivendell.nje.hu:22 3. Login: student208 / (your password) 4. Navigate to `/opt/tomcat/webapps/` 5. Upload `ihutsc-se.war` 6. Wait 30-60 seconds for auto-deployment **Option B: Command Line** `` ` ``bash`

Upload WAR

```
scp -P 22 target/ihutsc-se.war
student208@rivendell.nje.hu:/home/student208/
```

SSH to server

```
ssh student208@rivendell.nje.hu
```

Move to Tomcat

```
cp /home/student208/ihutsc-se.war /opt/tomcat/webapps/
```

Verify

```
ls -lh /opt/tomcat/webapps/ihutsc-se* `
```

Step 4: Verify Deployment

```
1. **Check Tomcat Manager:** - URL:
http://rivendell.nje.hu:9443/manager/text/list - User: test / test* -
Look for /ihutsc-se in running applications 2. **Access Application:** -
URL: http://rivendell.nje.hu:9443/ihutsc-se/ - Login: admin / admin123 3.
**Check Logs (if issues):** `bash ssh student208@rivendell.nje.hu tail -
f /opt/tomcat/logs/catalina.out `
```

Production Configuration

Database: MySQL on server

- URL: jdbc:mysql://localhost:3306/db208
- Username: studb208
- Password: abc123

- `Auto-create tables:` spring.jpa.hibernate.ddl-auto=update

`Application Properties:` ` `properties`

Production profile (no dev profile active)

`spring.profiles.active=`

MySQL configuration

`spring.datasource.url=jdbc:mysql://localhost:3306/db208 spring.datasource.username=studb208 spring.datasource.password=abc123 spring.jpa.hibernate.ddl-auto=update` ` `---`

User Guide

Login

`1. Navigate to http://localhost:8080 (or production URL) 2. Enter credentials: - Admin:` admin ` / ` admin123 ` - User: ` user ` / ` user123 ` 3. Click "Sign in"`

Managing Persons

Add New Person

`1. Click "Person Management" in navigation 2. Click "Add New Person" button 3. Fill in form: - Registration Number (unique, e.g., REG006) - Full Name - Birth Year (1900-2025) 4. Click "Save Person"`

View Person Details

`1. Go to Person Management page 2. Click "View" on any person row 3. See`

```
person info, associated vehicle, phone numbers
```

Edit Person

```
1. View person details 2. Click "Edit Person" button 3. Modify fields as needed 4. Click "Update Person"
```

Delete Person

```
1. View person details 2. Click "Delete Person" button 3. Confirm deletion (only works if no vehicle associated)
```

Managing Vehicles

Register New Vehicle

```
1. Click "Vehicles" in navigation 2. Click "Register New Vehicle" 3. Fill in form: - Registration Number (unique, e.g., PQR999) - Brand (e.g., Toyota Camry) - Production Year (1900-2025) - Color - Owner (select from dropdown) 4. Click "Register Vehicle"
```

View Vehicle Details

```
1. Go to Vehicles page 2. Click "View Details" on any vehicle 3. See vehicle info and owner details
```

Edit Vehicle

```
1. View vehicle details 2. Click "Edit Vehicle" button 3. Modify fields as needed 4. Click "Update Vehicle"
```

Delete Vehicle

```
1. View vehicle details 2. Click "Delete Vehicle" button 3. Confirm
```

`deletion`

Managing Phone Numbers

Add Phone to Person

`1. View person details 2. Scroll to "Phone Numbers" section 3. Click "Add Phone Number" 4. Enter phone number (e.g., +1234567890) 5. Click "Add Phone"`

Edit Phone Number

`1. View person details 2. Click "Edit" next to phone number 3. Update number 4. Click "Update Phone"`

Delete Phone Number

`1. View person details 2. Click "Delete" next to phone number 3. Confirm deletion`

Viewing Database

`1. Click "Database" in navigation 2. See all tables with record counts 3. View relationships between persons and vehicles 4. Use "Refresh" to update data`

Viewing Charts

`1. Click "Charts" in navigation 2. View bar chart of vehicles by brand 3. View pie chart of vehicles by color 4. Statistics update automatically ---`

Testing

Manual Testing Checklist

Person CRUD

- [x] Create new person with valid data
- [x] Create person with duplicate registration number (should fail)
- [x] Create person with invalid birth year (should fail)
- [x] View list of all persons
- [x] View individual person details
- [x] Edit person information
- [x] Delete person without vehicle
- [x] Attempt delete person with vehicle (should fail)

Vehicle CRUD

- [x] Register new vehicle with owner
- [x] Register vehicle without owner
- [x] Register vehicle with duplicate number (should fail)
- [x] View list of all vehicles
- [x] View vehicle details with owner
- [x] Edit vehicle information
- [x] Change vehicle owner
- [x] Delete vehicle

Phone CRUD

- [x] Add phone to person
- [x] Add multiple phones to same person
- [x] Edit phone number
- [x] Delete phone number
- [x] View all phones for person

Database Page

- [x] Display all persons
- [x] Display all vehicles
- [x] Display all phones
- [x] Show correct record counts
- [x] Display relationships correctly

Charts Page

- [x] Bar chart renders with correct data

- [x] Pie chart renders with correct data
- [x] Statistics display accurate counts
- [x] Charts update after data changes

API Endpoints

- [x] GET /api/persons returns all persons
- [x] GET /api/persons/{id} returns specific person
- [x] POST /api/persons creates new person
- [x] PUT /api/persons/{id} updates person
- [x] DELETE /api/persons/{id} deletes person
- [x] GET /api/vehicles returns all vehicles
- [x] GET /api/vehicles/{id} returns specific vehicle
- [x] API requires authentication

Security

- [x] Cannot access pages without login
- [x] Login with admin credentials works
- [x] Login with user credentials works
- [x] Logout works correctly
- [x] Session persists across requests
- [x] CSRF protection on forms

API Testing with Postman

```
1. **Import Collection:** - Create new Postman collection - Add requests for each API endpoint 2. **Test Authentication:** - GET http://localhost:8080/login (should redirect) - POST login form to get session 3. **Test Person API:** ` GET http://localhost:8080/api/persons GET http://localhost:8080/api/persons/REG001 POST http://localhost:8080/api/persons Body: {"regnumber":"TEST","name":"Test","birthyear":1990} PUT http://localhost:8080/api/persons/TEST Body: {"regnumber":"TEST","name":"Updated","birthyear":1991} DELETE http://localhost:8080/api/persons/TEST ` 4. **Test Vehicle API:** ` GET http://localhost:8080/api/vehicles GET
```

```
http://localhost:8080/api/vehicles/ABC123 ` ---
```

Project Structure

```
` vehicle-registration-app/ â”œâ”€â”€ src/ â”‚ â”œâ”€â”€ main/ â”‚ â”‚
â”œâ”€â”€ java/com/vehiclereg/ â”‚ â”‚ â”‚ â”œâ”€â”€ controller/ â”‚ â”‚
â”‚ â”‚ â”œâ”€â”€ ApiController.java # REST API endpoints â”‚ â”‚ â”‚ â”‚
â”œâ”€â”€ ChartController.java # Charts page â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€
CrudController.java # Person CRUD â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€
DatabaseController.java # Database overview â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€
HomeController.java # Homepage â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€
PhoneController.java # Phone CRUD â”‚ â”‚ â”‚ â”‚ â””â”€â”€
VehicleController.java # Vehicle CRUD â”‚ â”‚ â”‚ â”œâ”€â”€ entity/ â”‚
â”‚ â”‚ â”‚ â”œâ”€â”€ Person.java # Person entity â”‚ â”‚ â”‚ â”‚
â”œâ”€â”€ Phone.java # Phone entity â”‚ â”‚ â”‚ â”‚ â””â”€â”€
Vehicle.java # Vehicle entity â”‚ â”‚ â”‚ â”œâ”€â”€ repository/ â”‚ â”‚
â”‚ â”‚ â”œâ”€â”€ PersonRepository.java # Person data access â”‚ â”‚ â”‚
â”‚ â”œâ”€â”€ PhoneRepository.java # Phone data access â”‚ â”‚ â”‚ â”‚
â””â”€â”€ VehicleRepository.java # Vehicle data access â”‚ â”‚ â”‚
â”œâ”€â”€ config/ â”‚ â”‚ â”‚ â”‚ â””â”€â”€ SecurityConfig.java # Spring
Security config â”‚ â”‚ â”‚ â”œâ”€â”€ ServletInitializer.java # WAR
deployment config â”‚ â”‚ â”‚ â””â”€â”€
VehicleRegistrationApplication.java # Main class â”‚ â”‚ â””â”€â”€
resources/ â”‚ â”‚ â”œâ”€â”€ templates/ â”‚ â”‚ â”‚ â”œâ”€â”€ layout/ â”‚
â”‚ â”‚ â”‚ â””â”€â”€ main.html # Base layout â”‚ â”‚ â”‚ â”œâ”€â”€ crud/
â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€ create.html # Add person form â”‚ â”‚ â”‚ â”‚
â”œâ”€â”€ edit.html # Edit person form â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€
index.html # Person list â”‚ â”‚ â”‚ â”‚ â””â”€â”€ view.html # Person
details â”‚ â”‚ â”‚ â”œâ”€â”€ vehicles/ â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€
create.html # Register vehicle form â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€ edit.html #
Edit vehicle form â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€ index.html # Vehicle list â”‚
â”‚ â”‚ â”‚ â””â”€â”€ view.html # Vehicle details â”‚ â”‚ â”‚ â”œâ”€â”€
phone/ â”‚ â”‚ â”‚ â”‚ â”œâ”€â”€ add.html # Add phone form â”‚ â”‚ â”‚
â”‚ â””â”€â”€ edit.html # Edit phone form â”‚ â”‚ â”‚ â”œâ”€â”€ database/
```

```
â”‚ â”‚ â”‚ â”‚ â””â”€â”€ index.html # Database overview â”‚ â”‚ â”‚ â”œâ”€â”€ charts/ â”‚ â”‚ â”‚ â”‚ â””â”€â”€ index.html # Charts page â”‚ â”‚ â”‚ â”œâ”€â”€ home.html # Homepage â”‚ â”‚ â”‚ â””â”€â”€ login.html # Login page â”‚ â”‚ â”œâ”€â”€ static/ â”‚ â”‚ â”‚ â”œâ”€â”€ css/ â”‚ â”‚ â”‚ â”‚ â””â”€â”€ style.css # Custom styles â”‚ â”‚ â”‚ â””â”€â”€ js/ â”‚ â”‚ â”‚ â””â”€â”€ charts.js # Chart.js scripts â”‚ â”‚ â”œâ”€â”€ application.properties # Production config â”‚ â”‚ â”œâ”€â”€ application-dev.properties # Dev config (H2) â”‚ â”‚ â””â”€â”€ data.sql # Sample data â”‚ â””â”€â”€ test/ â”‚ â””â”€â”€ java/com/vehiclereg/ â”‚ â””â”€â”€ VehicleRegistrationApplicationTests.java â”œâ”€â”€ target/ â”‚ â””â”€â”€ ihutsc-se.war # Deployable WAR â”œâ”€â”€ deploy.ps1 # Deployment script â”œâ”€â”€ upload.bat # Alternative upload script â”œâ”€â”€ pom.xml # Maven configuration â”œâ”€â”€ mvnw, mvnw.cmd # Maven wrapper â”œâ”€â”€ README.md # Project README â”œâ”€â”€ DEPLOYMENT.md # Deployment guide â”œâ”€â”€ API_TESTING.md # API documentation â”œâ”€â”€ PROJECT_STATUS.md # Project status â””â”€â”€ DOCUMENTATION.md # This file ` ---
```

Git History

Commit Timeline

```
1. **Initial commit** (d3ca831) - Project setup with Spring Boot - Basic entity models - Repository interfaces - Initial controllers and templates 2. **Add vehicle and phone entities** (commit 2) - Created Vehicle entity with JPA annotations - Created Phone entity - Added repositories for both - Created basic CRUD operations 3. **Implement security and API** (commit 3) - Spring Security configuration - Login page and authentication - RESTful API endpoints - API testing documentation 4. **Add charts and database page** (commit 4) - Chart.js integration - Bar chart for vehicles by brand - Pie chart for vehicles by color - Database overview page 5. **Fix constraint violation** (commit 5) - Removed bidirectional JPA relationships - Fixed circular reference issues - Updated cascade operations 6. **Fix entity relationships and templates** (0e698a8) - Updated controllers to manually lookup related entities - Fixed all
```

`template errors (crud/view, vehicles, database) - Fixed red vehicles count (case sensitivity) - Added chart debugging - Created PROJECT_SUMMARY.md 7. **Configure production deployment** (5039fd6) - Updated application.properties for production MySQL - Created deploy.ps1 script with WinSCP support - Created upload.bat for manual deployment - Prepared WAR for Tomcat server`

Repository Statistics

- `**Total Commits:** 7`
- `**Contributors:** 1 (Lilla/MI804-png)`
- `**Branches:** main`
- `**Files:** 50+ source files`
- `**Lines of Code:** ~3000+ lines`

`---`

Troubleshooting

Common Issues & Solutions

1. Port 8080 Already in Use

Error: Web server failed to start. Port 8080 was already in use. **Solution:** ` `powershell`

Windows - Kill process on port 8080

```
Get-Process -Id (Get-NetTCPConnection -LocalPort 8080).OwningProcess |
Stop-Process -Force
```

Linux/Mac

```
lsof -ti:8080 | xargs kill -9 `
```

2. JAVA_HOME Not Set

Error: Error: JAVA_HOME not found **Solution:** `` ` ``powershell

Windows

`$env:JAVA_HOME = "C:\Program Files\Microsoft\jdk-11.0.16.101-hotspot"`

Linux/Mac

`export JAVA_HOME=/usr/lib/jvm/java-11-openjdk` `` ` ``

3. Maven Build Fails

Error: Failed to execute goal **Solution:** `` ` ``bash

Clean and rebuild

`mvnw clean install -U`

Skip tests if needed

`mvnw clean package -DskipTests` `` ` ``

4. H2 Console Not Accessible

Error: 404 on /h2-console **Solution:**

- Check spring.profiles.active=dev in application.properties
- Verify spring.h2.console.enabled=true in application-dev.properties
- Clear browser cache
- Try http://localhost:8080/h2-console directly

5. Login Redirects Loop

Error: Infinite redirect on login **Solution:**

- Check SecurityConfig.java formLogin() configuration
- Verify user credentials in application.properties
- Clear browser cookies/sessions
- Try incognito/private window

6. Database Connection Failed (Production)

Error: Unable to create initial connections of pool **Solution:**

- Verify MySQL is running on server
- Check database credentials (studb208/abc123)
- Ensure database db208 exists
- Test connection: mysql -u studb208 -p db208

7. WAR Deployment Not Working

Error: Application not accessible after WAR upload **Solution:** ` bash

Check Tomcat logs

```
tail -f /opt/tomcat/logs/catalina.out
```

Verify WAR unpacked

```
ls -la /opt/tomcat/webapps/ihutsc-se/
```

Restart Tomcat if needed

```
sudo systemctl restart tomcat `
```

8. Thymeleaf Template Error

`Error:` Error resolving template `Solution:`

- `Check template path matches controller return value`
- `Verify file is in `src/main/resources/templates/`
- Check for typos in template name
- Rebuild project

9. API Returns 401 Unauthorized

Error: API call returns 401 **Solution:**

- Login first via web interface or /login endpoint
- Include session cookie in API requests
- For testing, disable security on API endpoints temporarily
- Use Postman with cookie preservation

10. Chart Not Displaying

Error: Blank chart canvas **Solution:**

- Open browser console (F12) for JavaScript errors
- Verify Chart.js CDN is loading
- Check data being passed to chart
- Ensure canvas element has ID matching JavaScript
- Check ChartController debug logs

Future Enhancements

Planned Features

1. **Enhanced Security** - BCrypt password encoding - Role-based access control (ADMIN/USER permissions) - JWT tokens for API authentication - Password reset functionality - User registration with email verification 2. **Advanced Search & Filtering** - Search persons by name or registration number - Filter vehicles by brand, year, color - Date range filtering for production years - Pagination for large datasets 3. **File Upload** - Upload vehicle photos - PDF document storage (registration papers) - Profile pictures for persons - Document management system 4. **Reporting** - PDF export of vehicle list - Excel export for data analysis - Custom report generation - Email reports to admin 5. **Notifications** - Email notifications for new registrations - Registration expiry reminders - System alerts - User activity logs 6. **Dashboard Improvements** -

More chart types (line charts, area charts) - Real-time statistics updates - Widget-based customizable dashboard - Export charts as images 7. **API Enhancements** - Swagger/OpenAPI documentation - GraphQL support - Webhook integrations - Rate limiting - API versioning 8. **Mobile Responsiveness** - Mobile-first design - Progressive Web App (PWA) - Touch-friendly interfaces - Offline mode with sync 9. **Audit & Logging** - Comprehensive audit trails - Change history tracking - User activity monitoring - System health monitoring 10. **Testing** - Unit tests for all services - Integration tests - End-to-end tests with Selenium - Performance testing - Security testing

Technical Debt

- [ ] Add proper exception handling in controllers
- [ ] Implement DTO pattern for API responses
- [ ] Add transaction management
- [ ] Improve validation error messages
- [ ] Refactor duplicate code in templates
- [ ] Add comprehensive JavaDoc comments
- [ ] Implement proper logging with SLF4J
- [ ] Add database migration scripts (Flyway/Liquibase)

Conclusion

The Vehicle Registration Management System successfully meets all course requirements and demonstrates proficiency in:

- âœ… Spring Boot application development
- âœ… JPA/Hibernate database management
- âœ… Thymeleaf template engine
- âœ… RESTful API design
- âœ… Spring Security implementation
- âœ… Git version control
- âœ… Linux server deployment

The application is production-ready and deployed at: **http://rivendell.nje.hu:9443/ihutsc-se/** ---

Contact & Support

Student: Lilla **Neptun Code:** IHUTSC **GitHub:** https://github.com/MI804-png/java_seminar_5th_semester_Lilla **Course:** Java Applications - 5th Semester **Date:** November 27, 2025 For questions or issues, please refer to the repository issues page or contact the course instructor. --- **End of Documentation**