1. Project Overview

The Car Details Retriever Application parses car details from a CSV file and persists the information into a relational database. This application is built using Spring Boot, Spring Data JPA, and HyperSQL (HSQLDB) as the database. The focus is on data handling, persistence, and preventing duplicate records.

2. Technologies Used

Spring Boot: Framework for building standalone, production-ready Spring applications.

Spring Data JPA: Abstraction over common CRUD operations using JPA.

HyperSQL (HSQLDB): A relational database management system written in Java, used for both in-memory and disk-based database operations.

Maven: Manages project dependencies and the build lifecycle.

JDK 17 (or any LTS version of Java compatible with Spring Boot).

3. Application Structure

The project follows a standard layered architecture with the following components:

3.1 Entity Layer

Car Entity: Represents the data structure for storing car details. The fields are id, modelName, year, price, distance, engineType, sellerType, transmission, and owner.

Key Annotations:

@Entity: Marks the Car class as a JPA entity.

@Id and @GeneratedValue(strategy = GenerationType.IDENTITY): Defines the primary key and auto-generation strategy for the id field.

@Column: Maps class fields to database columns.

3.2 Repository Layer

CarRepository: Extends JpaRepository<Car, Long> and provides basic CRUD operations out of the box.

Key Features:

Automatically generates methods such as save(), findById(), deleteById(), and more.

Custom query methods can be added using the @Query annotation or derived query methods by following naming conventions.

3.3 Service Layer (Optional)

CarService: Handles business logic like verifying data integrity, checking for duplicates, or applying additional validation before interacting with the repository layer.

3.4 Data Loader

CsvDataLoader: Implements CommandLineRunner and reads data from a CSV file when the application starts. The data is converted into Car objects and persisted into the database after checking for duplicates.

Key Responsibilities:

Reading the CSV file from the resources directory.

Parsing the CSV data into Car entities.

Preventing duplicate data entry by checking existing records before inserting new ones.

Saving valid entries to the database.

4. CSV Data Loading

The application is designed to automatically load data from a CSV file (CAR_DETAILS_DATA.csv) upon startup.

4.1 CSV Format

The CSV file should follow this structure:

plaintext

Copy code

```
modelName,year,price,distance,engineType,sellerType,transmission,owner
```

Honda Civic,2018,20000,15000,Gasoline,Dealer,Automatic,First

Toyota Corolla,2019,18000,12000,Gasoline,Individual,Manual,First

4.2 File Location

Place the CSV file in the src/main/resources directory. The application automatically loads it from the classpath.

4.3 Duplicate Prevention

The application prevents duplicate entries by checking for existing records before inserting a new one. The check can be based on a unique combination of fields (e.g., modelName, year, price, etc.) or by comparing the entire object.

5. Database Configuration

5.1 HyperSQL (HSQLDB) Configuration

HyperSQL (HSQLDB) is used as the database engine. It can operate in both in-memory and persistent modes. For development purposes, this project uses HSQLDB in-memory mode, but it can be easily configured for disk-based persistence in production.

5.2 Application Properties

The application.properties file contains the configuration needed to set up the HSQLDB database.

FILE STRUCTURE

```
src
│
├── main
│   ├── java
│   │   └── com.example.cardetailsretriever
│   │       ├── entity
│   │       │   └── Car.java
│   │       ├── repository
│   │       │   └── CarRepository.java
│   │       ├── loader
│   │       │   └── CsvDataLoader.java
│   │       ├── security
│   │       │   └── SecurityConfig.java
│   │       ├── controller
│   │       │   └── CarController.java
│   │       └── CarDetailsRetrieverApplication.java
│   ├── resources
│   │   ├── application.yml
│   │   ├── CAR_DETAILS_DATA.csv
│   │   └── schema.sql
│   └── test
│       └── java
│           └── com.example.cardetailsretriever
│               └── CarDetailsRetrieverApplicationTests.java
```

JAVADOC -------------------------------

```
/**
* Car Details Retriever Application Documentation.
*
* This application parses car details from a CSV file and persists them into a HyperSQL
(HSQLDB) database.
* It uses Spring Boot, Spring Data JPA, and HSQLDB as the primary technologies.
*
* <h1>Project Overview</h1>
* The Car Details Retriever application loads and stores car information, ensuring no duplicate
entries
* are inserted. The data is initially loaded from a CSV file upon application startup.
*
* <h2>Technologies Used:</h2>
* <ul>
*   <li>Spring Boot: Framework for building Java applications</li>
*   <li>Spring Data JPA: Abstraction for database interactions using JPA</li>
*   <li>HyperSQL (HSQLDB): An in-memory or disk-based relational database</li>
*   <li>Maven: Build tool and dependency manager</li>
*   <li>JDK 17</li>
* </ul>
*/
public class CarDetailsRetrieverApplication {
    // Main class for running the Spring Boot application
}

/**
```

```
* Car Entity class representing the car details to be persisted in the database.
*
* @Entity: Marks the class as a JPA entity
* @Table(name = "cars"): Maps this entity to the "cars" table in the database
*
* Fields:
* <ul>
*   <li>id: Auto-generated primary key</li>
*   <li>modelName: Name of the car model</li>
*   <li>year: Manufacturing year of the car</li>
*   <li>price: Price of the car</li>
*   <li>distance: Distance the car has traveled</li>
*   <li>engineType: Type of the car's engine (e.g., Gasoline, Diesel)</li>
*   <li>sellerType: Type of seller (Dealer or Individual)</li>
*   <li>transmission: Transmission type (Manual, Automatic)</li>
*   <li>owner: Number of previous owners</li>
* </ul>
*/
@Entity
@Table(name = "cars")
public class Car {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String modelName;
    private String year;
    private Double price;
    private Double distance;
    private String engineType;

    private String sellerType;

    private String transmission;

    private String owner;


    // Getters and Setters

}


/**

* Repository interface for interacting with the car entity in the database.

*

* @Repository: Marks the interface as a Spring Data repository

* @param <Car>: The entity type managed by this repository

* @param <Long>: The type of the entity's primary key

*

* <h2>Methods:</h2>

* <ul>

*   <li>save(): Saves a new or updated car to the database</li>

*   <li>findById(): Finds a car by its ID</li>

*   <li>findAll(): Retrieves all car records</li>

*   <li>deleteById(): Deletes a car by its ID</li>

*   <li>existsBy[Field](): Checks if a car with a specific field value already exists</li>

* </ul>

*/

@Repository

public interface CarRepository extends JpaRepository<Car, Long> {

    // Custom queries for checking if a car record exists can be defined here

}
```

```
/**
* CSV Data Loader class that loads car data from a CSV file into the database.
*
* @Component: Marks this class as a Spring Bean and allows it to be run during application
startup
* @Implements CommandLineRunner: This interface is used to execute the run() method after
the application context is loaded
*
* <h2>Responsibilities:</h2>
* <ul>
*   <li>Reads the CSV file from the resources directory</li>
*   <li>Parses the CSV file into Car entities</li>
*   <li>Prevents the insertion of duplicate car entries</li>
*   <li>Saves valid car data to the database</li>
* </ul>
*/
@Component
public class CsvDataLoader implements CommandLineRunner {
    @Autowired
    private CarRepository carRepository;

    /**
     * This method is executed when the application starts and loads data from the CSV file.
     *
     * @param args: Command-line arguments passed to the application
     * @throws Exception: Throws exception if any error occurs during file reading or database
interaction
     */
    @Override
    public void run(String... args) throws Exception {
        BufferedReader reader = new BufferedReader(new InputStreamReader(new
ClassPathResource("CAR_DETAILS_DATA.csv").getInputStream()));
        String line;
        reader.readLine(); // Skip the header
        while ((line = reader.readLine()) != null) {
            String[] fields = line.split(",");
            Car carDetails = new Car(
                null,                  // ID will be auto-generated
                fields[0],             // modelName
                fields[1],             // year
                Double.valueOf(fields[2]), // price
                Double.valueOf(fields[3]), // distance
                fields[4],             // engineType
                fields[5],             // sellerType
                fields[6],             // transmission
                fields[7]              // owner
            );
            if (!carRepository.existsByModelNameAndYearAndPrice(carDetails.getModelName(),
carDetails.getYear(), carDetails.getPrice())) {
                carRepository.save(carDetails);
            }
        }
    }
}


/**
```

```
* Configuration class for setting up the HSQLDB database.
*
* @Configuration: Marks this class as a configuration class for Spring Boot
*
* <h2>Application Properties:</h2>
* <ul>
*   <li>spring.datasource.url: The URL for connecting to the HSQLDB database</li>
*   <li>spring.datasource.driverClassName: The JDBC driver for HSQLDB</li>
*   <li>spring.jpa.database-platform: Hibernate dialect for HSQLDB</li>
*   <li>spring.jpa.hibernate.ddl-auto: DDL auto configuration for Hibernate</li>
* </ul>
*/
@Configuration
public class HSQLDBConfig {
    // This class can be used for custom database configuration if necessary
}

/**
* Main class for running the application using Spring Boot.
*
* @SpringBootApplication: Marks this as a Spring Boot application
*/
@SpringBootApplication
public class CarDetailsRetrieverApplication {
    public static void main(String[] args) {
        SpringApplication.run(CarDetailsRetrieverApplication.class, args);
    }
}
```