CSV Reader and REST API Documentation

Overview

This application is designed to read customer information from a CSV file, send the data to a REST API endpoint and save it to a SQL database. The application consists of two main parts:

. CSV Reader (Console App): Reads customer information from a CSV file.

. REST API (Spring Boot): Exposes endpoints to receive and retrieve customer information.

1. CSV Reader (Console App)

Customer Class

The Customer class represents the customer entity with fields such as customerRef, customerName, addressLine1, etc.

```
public class Customer {
    // Fields, getters, and setters
}
```

CustomerMapper Class

The CustomerMapper class converts CSV records to Customer objects.

```
public class CustomerMapper {
    public static List<Customer> mapToCustomers(List<CSVRecord> records
        // Implementation
    }
}
```

CSVReaderApp Class

The CSVReaderApp class is the entry point for the console app. It reads data from a CSV file, maps it to Customer objects, and sends the data to a REST API endpoint.

```
public class CSVReaderApp {
    public static void main(String[] args) {
        // Implementation
    }
}
```

2. REST API (Spring Boot)

Customer Controller

The CustomerController class exposes REST endpoints for adding and retrieving customer information.

```
@RestController
@RequestMapping("/api/customers")
public class CustomerController {
    // Endpoints for adding and retrieving customers
}
```

Customer Service

The CustomerService class contains business logic for adding and retrieving customer information.

```
@Service
public class CustomerService {
    // Methods for adding and retrieving customers
}
```

Customer Repository

The CustomerRepository interface extends JpaRepository for CRUD operations on the Customer entity.

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {
    // Custom query methods
}
```

Application Class

The Application class is the entry point for the Spring Boot application.

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

3. Tests

CSV Reader Tests

JUnit tests for the CSV reader app using Mockito.

```
public class CSVReaderAppTest {
    // Tests for CSVReaderApp using Mockito
}
```

Spring Boot Tests

Spring Boot tests using JUnit for testing the REST API.

```
@SpringBootTest
public class CustomerServiceTest {
    // Tests for CustomerService using Mockito and JUnit
}
```

In these tests, Mockito is used to mock the CustomerRepository dependency injected into the CustomerService. The @Mock annotation is used to create a mock, and the @InjectMocks annotation injects the mock dependencies into the CustomerService.

The testAddCustomer method verifies that the save method of the repository is called once when adding a customer.

The testGetCustomerByRef method tests the getCustomer method by setting up a mock response from the repository and then asserting that the retrieved customer matches the expected customer.

Running the application on your device:

Prerequisites:

. Java Runtime Environment (JRE)

. Database

Steps to Run the Application:

. Navigate to the Project Directory: Open a terminal or command prompt on the target machine and navigate to the directory where the project files are stored.

. Run the CSV Reader (Console App): Navigate to the target directory where the compiled JAR file is located. Then, execute the JAR file.

. Run the Spring Boot Application: If your project is a Spring Boot application, navigate to the project root directory where the compiled JAR file is located. Then, execute the JAR file.

. Configuration :

. Database Configuration: If your application uses an external database, make sure to update the database configuration in the application.properties or application.yml file.

. API Endpoint: Update the API endpoint in the CSVReaderApp class to match the URL of your REST API on the target machine.

Troubleshooting

. Port Conflicts: If your application uses a specific port (e.g., 8080 for Spring Boot), ensure the port is available on the target machine. You can change the port in the application.properties or application.yml file.

. Firewall: Ensure that the firewall on the target machine allows traffic on the port your application is using.