# ASSIGNMENT 7:

# Spring Boot Leave Management System with CRUD APIs

## Technologies Used

* **Spring Boot:** Simplifies application setup with auto-configuration.
* Spring Data JPA: Manages database interactions for bus data persistence.
* Spring Web: Builds RESTful APIs for bus management.
* **Maven:** Manages project dependencies efficiently.
* **Java 11+:** The programming language for the application.
* H2/MySQL: Databases to store bus information.
* **IntelliJ IDEA/Eclipse:** IDEs for coding and debugging the application.

## Project Structure

The project follows a standard Maven structure. Key directories include:

BusManagementSystem/  
├── src/  
│ ├── main/  
│ │ ├── java/com/example/cms/  
│ │ │ ├── controller/  
│ │ │ ├── model/  
│ │ │ ├── repository/  
│ │ │ └── service/  
│ └── resources/  
│ └── application.properties  
├── pom.xml

## Entity Class (Bus.java)

package com.example.cms.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
public class Bus {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private String firstName;  
 private String lastName;  
 private String email;  
 private String phone;  
  
}

## Repository Interface (BusRepository.java)

package com.example.cms.repository;  
  
import com.example.cms.model.Bus;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface BusRepository extends JpaRepository<Bus, Long> {  
}

This interface enables easy data access and management for bus entities.

## Service Class (BusService.java)

package com.example.cms.service;  
  
import com.example.cms.model.Bus;  
import com.example.cms.repository.BusRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class BusService {  
  
 @Autowired  
 private BusRepository repository;  
  
 public Bus save(Bus bus) {  
 // Implementation  
 }  
  
 public List<Bus> getAll() {  
 // Implementation  
 }  
  
 public Optional<Bus> getById(Long id) {  
 // Implementation  
 }  
  
 public void delete(Long id) {  
 // Implementation  
 }  
}

## Controller Class (BusController.java)

package com.example.cms.controller;  
  
import com.example.cms.model.Bus;  
import com.example.cms.service.BusService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/buss")  
public class BusController {  
  
 @Autowired  
 private BusService service;  
  
 @PostMapping  
 public Bus create(@RequestBody Bus bus) {  
 // Implementation  
 }  
  
 @GetMapping  
 public List<Bus> getAll() {  
 // Implementation  
 }  
  
 @GetMapping("/{id}")  
 public ResponseEntity<Bus> getById(@PathVariable Long id) {  
 // Implementation  
 }  
  
 @PutMapping("/{id}")  
 public ResponseEntity<Bus> update(@PathVariable Long id, @RequestBody Bus bus) {  
 // Implementation  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity<Void> delete(@PathVariable Long id) {  
 // Implementation  
 }  
}  
   
 application.properties Configuration

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true

## pom.xml Dependencies:

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>  
   
**Testing the API**

Use tools like Postman or curl to test the Bus Reservation System API. Below are the endpoints for CRUD operations:

* POST /api/buss: Creates a new bus.
* GET /api/buss: Retrieves all buss.
* GET /api/buss/{id}: Retrieves a bus by ID.
* PUT /api/buss/{id}: Updates a bus.
* DELETE /api/buss/{id}: Deletes a bus.

## Conclusion

The Spring Boot Bus Reservation System showcases a structured backend using Spring technologies. It uses a layered architecture and clean coding practices, forming a base for full-stack development. Next steps include adding user authentication, improving the UI, and incorporating more complex data validations for robust bus management.