

## 1. Eclipse Setup and Configuration

The easiest way to work with Spring Boot in Eclipse is by using the **Spring Tools 4 (STS)** extension.

### Step 1: Install Spring Tools 4 (STS)

1. Open **Eclipse IDE for Enterprise Java and Web Developers**.
2. Go to the main menu: Help -> Eclipse Marketplace....
3. Search for "**Spring Tools 4 (STS)**".
4. Click **Install** and follow the prompts. Restart Eclipse when prompted.

### Step 2: Create a Spring Boot Project

1. Go to File -> New -> **Spring Starter Project**.
2. **Project Details:**
  - **Name:** E.g., TaskManagementMicroservice
  - **Type:** Maven
  - **Java Version:** 17 or higher
3. **Dependencies (The crucial part):** Click **Next**. In the search bar, select the following:
  - **Spring Web:** Essential for creating RESTful APIs.
  - **Lombok:** *Highly recommended* for reducing boilerplate code (getters/setters/constructors).
4. Click **Finish**. Eclipse will generate the project structure and download all required libraries.

### Step 3: Running the Application (Commands)

1. **Run via Eclipse:** Locate the main application file (e.g., TaskManagementMicroserviceApplication.java) in the src/main/java folder. Right-click and choose **Run As -> Spring Boot App**.
2. **Run via Command Line (Optional):**
  - Open a terminal in the root directory of your project.
  - **Command 1 (Build):** mvn clean install
  - **Command 2 (Run):** java -jar target/TaskManagementMicroservice-0.0.1-SNAPSHOT.jar (Replace the file name with your generated JAR file).

The application will start on the default port, usually http://localhost:8080.

---

## 2. Program Task Management API (In-Memory)

- **Questions:** (Spring Task Management API) and (Student Management Microservice).

- **Concept:** Implements a standard **Model-View-Controller (MVC)** pattern. The persistence layer (Repository) uses a simple Java **List** (in-memory database).

#### Java Code Structure

File	Type	Purpose
Task.java	<b>Model (Data)</b>	Defines the structure of a Task object.
TaskRepository.java	<b>Repository (Data Access)</b>	Simulates database operations using a static List.
TaskController.java	<b>Controller (API)</b>	Handles HTTP requests and returns JSON responses.

#### Task.java (Model)

Java

```
// Package: com.example.task.model
```

```
import lombok.Data; // Requires Lombok dependency
```

```
@Data // Generates getters, setters, toString, equals, and hashCode
```

```
public class Task {
```

```
    private Long id;
```

```
    private String title;
```

```
    private String description;
```

```
    private boolean completed;
```

```
    // Default constructor for Spring/Jackson
```

```
    public Task() {}
```

```
    public Task(Long id, String title, String description) {
```

```
        this.id = id;
```

```
        this.title = title;
```

```
        this.description = description;
```

```
        this.completed = false;
```

```
    }
```

```
}
```

## TaskRepository.java (In-Memory Persistence)

Java

```
// Package: com.example.task.repository
```

```
import com.example.task.model.Task;
```

```
import org.springframework.stereotype.Repository;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.Optional;
```

```
import java.util.concurrent.atomic.AtomicLong;
```

```
@Repository // Marks this class as a Spring component for data access
```

```
public class TaskRepository {
```

```
    // Simulating a database table using a static List
```

```
    private final List<Task> tasks = new ArrayList<>();
```

```
    private final AtomicLong counter = new AtomicLong(0);
```

```
    // Initialization block to add some dummy data on startup
```

```
    {
```

```
        save(new Task(null, "Setup Eclipse", "Install STS and JDK 17+"));
```

```
        save(new Task(null, "Implement Slip A1", "Write Controller, Service, and Repository classes"));
```

```
    }
```

```
    public List<Task> findAll() {
```

```
        return tasks;
```

```
    }
```

```
    public Optional<Task> findById(Long id) {
```

```
        return tasks.stream()
```

```

        .filter(t -> t.getId().equals(id))
        .findFirst();
    }

```

```

public Task save(Task task) {
    if (task.getId() == null) {
        // Assign new ID for a new task
        task.setId(counter.incrementAndGet());
        tasks.add(task);
    } else {
        // Update existing task (requires complex logic, simplified here)
        // For a 2-hour exam, focus on create/read/delete
        deleteById(task.getId()); // Remove old version
        tasks.add(task); // Add new version (simple update)
    }
    return task;
}

```

```

public void deleteById(Long id) {
    tasks.removeIf(t -> t.getId().equals(id));
}
}

```

### **TaskController.java (REST API Endpoints)**

Java

// Package: com.example.task.controller

```

import com.example.task.model.Task;
import com.example.task.repository.TaskRepository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

```

```
import java.util.List;

@RestController // Makes this a RESTful controller
@RequestMapping("/api/tasks") // Base URL for all endpoints in this controller
public class TaskController {

    private final TaskRepository taskRepository;

    // Constructor Injection (Spring automatically injects TaskRepository)
    public TaskController(TaskRepository taskRepository) {
        this.taskRepository = taskRepository;
    }

    // 1. GET: List all tasks
    @GetMapping
    public List<Task> getAllTasks() {
        return taskRepository.findAll();
    }

    // 2. POST: Create a new task
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED) // Returns 201 status code
    public Task createTask(@RequestBody Task task) {
        return taskRepository.save(task);
    }

    // 3. DELETE: Remove a task by ID
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteTask(@PathVariable Long id) {
        if (taskRepository.findById(id).isPresent()) {
```

```

        taskRepository.deleteById(id);

        return ResponseEntity.noContent().build(); // 204 No Content
    } else {

        return ResponseEntity.notFound().build(); // 404 Not Found
    }
}
}
}

```

### API Commands (cURL/Postman Equivalent)

Assuming the app is running on <http://localhost:8080>.

Action	HTTP Method	URL	Command/Body
Get All	GET	/api/tasks	curl -X GET <a href="http://localhost:8080/api/tasks">http://localhost:8080/api/tasks</a>
Create Task	POST	/api/tasks	curl -X POST -H "Content-Type: application/json" -d '{"title": "Read Book", "description": "Finish SADP chapter 3"}' <a href="http://localhost:8080/api/tasks">http://localhost:8080/api/tasks</a>
Delete Task	DELETE	/api/tasks/1	curl -X DELETE <a href="http://localhost:8080/api/tasks/1">http://localhost:8080/api/tasks/1</a>

### 3. Program : Employee Details (File-based)

- **Question:** (Employee Details Microservice - File-based).
- **Concept:** The Repository layer will use **Java's java.io package** to write and read data from a local file (employees.txt) instead of an in-memory list.

#### Employee.java (Model)

Use the same structure as Task.java, but for an Employee.

Java

// Package: com.example.employee.model

import lombok.Data;

@Data

public class Employee {

private Long id;

private String name;

```
private String department;
```

```
private double salary;
```

```
// Default constructor/Getters/Setters generated by @Data
```

```
}
```

### **FileEmployeeRepository.java (File Persistence)**

This is the core change. It implements persistence using file I/O.

Java

```
// Package: com.example.employee.repository
```

```
import com.example.employee.model.Employee;
```

```
import org.springframework.stereotype.Repository;
```

```
import java.io.*;
```

```
import java.nio.file.Files;
```

```
import java.nio.file.Paths;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.concurrent.atomic.AtomicLong;
```

```
@Repository
```

```
public class FileEmployeeRepository {
```

```
    private static final String FILE_PATH = "employees.txt";
```

```
    private final AtomicLong counter = new AtomicLong(0);
```

```
// Reads all employees from the file (each line is one employee record)
```

```
public List<Employee> findAll() {
```

```
    List<Employee> employees = new ArrayList<>();
```

```
    try (BufferedReader reader = Files.newBufferedReader(Paths.get(FILE_PATH))) {
```

```
        String line;
```

```

while ((line = reader.readLine()) != null) {
    // Format: ID,NAME,DEPARTMENT,SALARY
    String[] parts = line.split(",");
    if (parts.length == 4) {
        Employee emp = new Employee();
        emp.setId(Long.parseLong(parts[0]));
        emp.setName(parts[1]);
        emp.setDepartment(parts[2]);
        emp.setSalary(Double.parseDouble(parts[3]));
        employees.add(emp);
        // Update counter to ensure unique IDs
        counter.set(Math.max(counter.get(), emp.getId()));
    }
}
} catch (IOException e) {
    // File not found, or error reading (First run scenario)
    System.err.println("Employee file not found or error reading: " + e.getMessage());
}
return employees;
}

```

// Saves a new employee to the file (appends to end)

```

public Employee save(Employee employee) {
    if (employee.getId() == null) {
        employee.setId(counter.incrementAndGet());
    }
}

```

// Data format: ID,NAME,DEPARTMENT,SALARY

```

String record = String.format("%d,%s,%s,%.2f%n",
    employee.getId(),
    employee.getName(),

```

```
        employee.getDepartment(),  
        employee.getSalary());
```

```
        try (FileWriter writer = new FileWriter(FILE_PATH, true)) { // 'true' for append mode  
            writer.write(record);  
        } catch (IOException e) {  
            throw new RuntimeException("Error saving employee to file.", e);  
        }  
        return employee;  
    }  
}
```

### **EmployeeController.java (REST API Endpoints)**

The controller remains simple, but calls the FileEmployeeRepository.

Java

```
// Package: com.example.employee.controller
```

```
import com.example.employee.model.Employee;  
import com.example.employee.repository.FileEmployeeRepository;  
import org.springframework.http.HttpStatus;  
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;
```

```
@RestController
```

```
@RequestMapping("/api/employees")
```

```
public class EmployeeController {
```

```
    private final FileEmployeeRepository repository;
```

```
    public EmployeeController(FileEmployeeRepository repository) {  
        this.repository = repository;
```

```
}
```

```
// 1. GET: List all employees from the file
```

```
@GetMapping
```

```
public List<Employee> getAllEmployees() {
```

```
    return repository.findAll();
```

```
}
```

```
// 2. POST: Create a new employee and save to file
```

```
@PostMapping
```

```
@ResponseStatus(HttpStatus.CREATED)
```

```
public Employee createEmployee(@RequestBody Employee employee) {
```

```
    return repository.save(employee);
```

```
}
```

```
// Note: Delete and Update operations are complex with simple file I/O
```

```
// (requires reading the whole file, modifying, and rewriting the whole file).
```

```
// a simple create/read file-based service is sufficient as point of exam time limit
```

```
}
```

#### API Commands (cURL/Postman Equivalent)

Action	HTTP Method	URL	Command/Body
Get All	GET	/api/employees	curl -X GET http://localhost:8080/api/employees
Create Emp	POST	/api/employees	curl -X POST -H "Content-Type: application/json" -d '{"name": "Alice", "department": "HR", "salary": 65000.00}' http://localhost:8080/api/employees

After running the POST command, a file named employees.txt will appear in your project's root directory containing the saved employee record.