**report**

I have created a Todo List web application using HTML, JavaScript, Spring MVC, MySQL, and RESTful APIs as part of Sprint 2 Day 5.

The frontend is built using index.html, which is placed under src/main/webapp. This file includes JavaScript code that demonstrates key JavaScript concepts such as data types, functions, control flow, hoisting, objects, and arrays. I used let, const, and var to showcase data types and hoisting. Core functionalities like adding, deleting, and rendering tasks are implemented using functions: addTodo(), deleteTodo(), and renderTodos() respectively. An array named todos is used to manage the list of tasks fetched from the backend. The forEach() method is used to iterate and display each task, and conditionals handle task validation. Objects are used to represent individual tasks with properties like id and task.

On the backend, I developed a Java class TodoController.java under src/main/java/com/nisum/controller. It is annotated as a Spring @RestController and exposes REST endpoints:

GET /api/todos – returns the list of tasks from the database

POST /api/todos – adds a new task to the database

DELETE /api/todos/{id} – deletes the task with the specified ID

Tasks are stored in a MySQL database named todo\_db, inside a todos table. JDBC is used for database connectivity, and JdbcTemplate handles SQL execution. The database connection is configured via db.properties, and Spring beans for DataSource and JdbcTemplate are defined in spring-servlet.xml.

The entire application is configured using Spring XML files — web.xml and spring-servlet.xml — under WEB-INF. These files enable DispatcherServlet, component scanning, and annotation support. Maven is used for dependency and build management. The project is packaged as a WAR and deployed using Smart Tomcat with MySQL running locally.

The app is tested by accessing http://localhost:8080/todo. The frontend interacts with the backend via fetch-based API calls. All REST calls (GET, POST, DELETE) are functional, and responses are correctly rendered on the UI. Chrome DevTools' Network tab shows 200 OK for all API calls, and visiting http://localhost:8080/todo/api/todos returns the expected JSON output.

All six required JavaScript concepts — data types, functions, control flow statements, hoisting, object, and array — are demonstrated in the frontend code and verified through the working app.

Output:













