■ Project Plan: To-Do Web App (React + Java + PostgreSQL)

■ Features

Core Features

1. Add a new task → type text and save it.
2. View all tasks → show in a list.
3. Mark task complete/incomplete → checkbox or toggle.
4. Delete a task → remove from list and database.
5. Data persistence → tasks remain after refresh (saved in PostgreSQL).

Optional Features

1. Edit task title.
2. Show separate lists: Active Tasks and Completed Tasks.
3. Deploy app online (Netlify/Vercel for frontend, Render/Heroku for backend).

■ Week 1: Setup + Database

Day 1

1. Install Node.js, Java JDK, PostgreSQL.
2. Install IDEs: VS Code (frontend), IntelliJ/Eclipse (backend).
3. Verify installations with commands: node -v, java -version, psql --version.

Day 2

1. Open pgAdmin.
2. Create database todo_app.
3. Create table tasks: CREATE TABLE tasks (id SERIAL PRIMARY KEY, title TEXT NOT NULL, completed BOOLEAN DEFAULT FALSE);

Day 3

1. Insert 2–3 sample tasks manually with INSERT INTO tasks.
2. Run SELECT * FROM tasks; to confirm.
3. Learn basic SQL (INSERT, SELECT, UPDATE, DELETE).

Day 4

1. Practice updating/deleting rows manually in pgAdmin.
2. Example: UPDATE tasks SET completed = TRUE WHERE id=1;

Day 5

1. Recap: You can create a DB, table, insert tasks, and query them.
2. Write a mini SQL cheat sheet.

■ Week 2: Backend (Java + Spring Boot)

Day 6

1. Go to start.spring.io.
2. Generate new project with: Spring Web, Spring Data JPA, PostgreSQL Driver.
3. Open project in IntelliJ.

Day 7

1. Configure application.properties to connect to PostgreSQL.
2. spring.datasource.url=jdbc:postgresql://localhost:5432/todo_app
3. spring.datasource.username=your_username
4. spring.datasource.password=your_password
5. spring.jpa.hibernate.ddl-auto=update
6. Run app → confirm it starts with no errors.

Day 8

1. Create Task.java (entity with fields: id, title, completed).
2. Create TaskRepository.java (extends JpaRepository).

Day 9

1. Create TaskController.java.
2. Add endpoint: GET /tasks → return all tasks.
3. Test with Postman.

Day 10

1. Add endpoints: POST /tasks → add task, PUT /tasks/{id} → update task, DELETE /tasks/{id} → remove task.
2. Test all in Postman.

■ Week 3: Frontend (React Basics)

Day 11

1. In VS Code: npx create-react-app todo-frontend, cd todo-frontend, npm start.
2. Confirm app runs at http://localhost:3000.

Day 12

1. Create TaskInput.js component → textbox + 'Add Task' button.
2. Display input value in console when pressing button.

Day 13

1. Create TaskList.js → show a list of hardcoded tasks (array).
2. Create TaskItem.js → single task with checkbox + delete button.

Day 14

1. Use React useState to manage tasks array.
2. Add new tasks locally (without backend).

Day 15

1. Add toggle complete + delete (still local).
2. By now, you have a working frontend with dummy data.

■ Week 4: Connect Frontend + Backend

Day 16

1. Install axios in React: npm install axios.
2. Call backend GET /tasks → show tasks from DB instead of dummy data.

Day 17

1. Connect POST /tasks → adding a task in frontend should save it to DB.

Day 18

1. Connect PUT /tasks/{id} → toggling checkbox updates DB.

Day 19

1. Connect DELETE /tasks/{id} → deleting removes from DB.

Day 20

1. Full testing:
2. Add → task saved in DB.
3. Reload → still there.
4. Mark complete → updates DB.
5. Delete → task removed from DB.

■ Week 5: Optional Polish

1. Add task editing.
2. Add 'Active' vs 'Completed' filter.
3. Push code to GitHub.
4. Try deployment (Netlify + Render).

■ Final Deliverables

1. ■ React frontend (todo-frontend).
2. ■ Spring Boot backend (todo-backend).
3. ■ PostgreSQL database with real tasks.
4. ■ GitHub repository with both projects.
5. ■ Working app: user can add, view, complete, delete tasks.