Todo List Web App – Requirements

# Requirements

- store a list

- add items

- mark items as done

- remove items

- edit items

**Tech discovery**

**Frontend (React)**

1. **React components**:
   * **TodoList**: Displays the list of tasks.
   * **TodoItem**: Represents individual items (editable, markable as done, removable).
   * **AddTodo**: Form to add new tasks.
   * **EditTodo**: Allows editing task details (optional).
2. **State management**: Use React's useState or useReducer to manage the state of tasks (e.g., completed or not, item list).
3. **Event handling**: Handle user actions like adding, editing, marking as done, and deleting tasks through React event listeners.

**Backend (TypeScript)**

1. **API with Express**:
   * **POST** /todos: Add a new todo item.
   * **GET** /todos: Retrieve all tasks.
   * **PUT** /todos/:id: Edit a todo item (mark as done, update details).
   * **DELETE** /todos/:id: Remove a todo item.
2. **Database**:
   * Use a simple database like SQLite, or PostgreSQL if scaling.
   * Define a **Todo** model with fields like id, text, isDone, and createdAt.
3. **Data validation**: Use libraries like zod or Joi to ensure valid data for requests.
4. **API testing**: Tools like Postman for testing backend APIs.
5. **Deployment**: You can deploy on services like Vercel (for frontend) and Fly.io/Heroku (for backend).

# [Done] Steps 1-3

To get started, here are the first three steps:

1. **Set Up the Frontend (React)**
   * Install Node.js if you don’t have it.
   * Create a React app:

bash

Copy code

npx create-react-app todo-app --template typescript

* + This will set up a React project with TypeScript support.

1. **Set Up the Backend (TypeScript)**
   * Initialize a Node.js + TypeScript backend:

bash

Copy code

mkdir todo-backend

cd todo-backend

npm init -y

npm install express typescript @types/express ts-node nodemon

* + Initialize TypeScript configuration:

bash

Copy code

npx tsc --init

* + Create a basic Express server in index.ts and set it up to listen on a port.

1. **Link Frontend and Backend**
   * Set up CORS in your Express backend to allow cross-origin requests:

bash

Copy code

npm install cors

* + In index.ts:

typescript

Copy code

import cors from 'cors';

app.use(cors());

These steps will get your development environment ready. You can then start implementing features (task management, API endpoints) step by step.

# Steps 4-6

1. **Create the Todo Model and Database Setup**:
   * [Done] Choose a database (e.g., SQLite or PostgreSQL).
     + PostgreSQL as it is used at Klarna
   * [Done] Install PostgreSQL
   * [Done] Create todo\_app database
   * [Done] Set up database connection
   * [Done] Create Sequelize file
   * [Done] Create a Todo model/schema with fields: id, text, isDone, and createdAt.
   * [Done] Sync model with database
   * [Done] Test database connection
   * [Done] Verify table in postgresql
2. **Implement API Endpoints**:
   * Set up your Express server with the following routes:
     + POST /todos to add a new todo item.
     + GET /todos to retrieve all tasks.
     + PUT /todos/:id to edit a todo item.
     + DELETE /todos/:id to remove a todo item.
   * Ensure you handle requests and responses correctly.
3. **Build Frontend Components**:
   * Create React components for:
     + TodoList to display the list of todos.
     + TodoItem to show each item with options to mark as done, edit, or delete.
     + AddTodo form to submit new tasks.
   * Use useEffect to fetch todos from the backend when the app loads.

This will set a solid foundation for your todo app!

# Appendix

## Appendix A – Sequelize as an Object-Relational Mapping

Object-Relational Mapping (ORM) = programming technique that allows developers to interact with a relational database using object-oriented programming languages

Sequelize is an ORM, but not an object database. Key difference:

* Sequelize: ORM for relational databases.
* Object Database: Directly stores data as objects, with no conversion to tables.

Nice side-node: Some folks at Klarna use Sequelize to construct PostgreSQL queries ([Klarna Engineering Blog](https://engineering.klarna.com/node-js-integration-testing-with-ease-fab5f8d29163)).