Low-Level Design (LLD) Document

Node.js To-Do List Application

Mahesh Kumar Tippanu

December 9, 2025

Contents

1 Introduction

This Low-Level Design (LLD) document describes the internal technical structure of the Node.js To-Do List project shown in the screenshot. The project is built using:

- Node.js
- Express.js
- EJS Template Engine
- HTML/CSS frontend

The purpose of the application is to allow users to:

- Add tasks
- View tasks
- Remove selected tasks

The system stores tasks **in memory**, without a database.

2 Project Structure

```
node-js-dummy-test/

          public/
                styles.css

          src/
                index.js

          views/
                index.ejs

          package.json
          Dockerfile
          README.md
```

3 System Overview

The project follows a Model-View-Controller style structure:

- **Controller**: Express routes in index.js
- **Model**: In-memory task list (array)
- **View**: index.ejs template rendered by Express

4 Module-Level LLD

4.1 Backend Module (src/index.js)

Responsibilities

- Start Express server
- Maintain task list in memory
- Render the UI with EJS
- Add tasks (POST /addtask)
- Remove tasks (POST /removetask)

Internal Data Structure

`TaskList: string[]`

Key Operations

1. **addTask(task: string)** Appends a new task to the in-memory list.
2. **removeTask(list: string[])** Removes tasks matching the selected checkbox values.
3. **renderPage()** Sends task data to index.ejs for rendering.

Backend Code Snippet

```
// src/index.js (representative structure)

const express = require("express");
const app = express();
app.set("view engine", "ejs");

let task = [];

app.get("/", (req, res) => {
    res.render("index", { task: task });
});

app.post("/addtask", (req, res) => {
    task.push(req.body.newtask);
    res.redirect("/");
});

app.post("/removetask", (req, res) => {
    const remove = req.body.check;
```

```
    task = task.filter(t => !remove.includes(t));
    res.redirect("/");
});

app.listen(3000);
```

5 View Layer LLD (views/index.ejs)

Responsibilities

- Display current tasks
- Provide input field for new tasks
- Provide checkboxes for removing tasks

Template Behavior

```
<% for(var i = 0; i < task.length; i++) { %>
    <li>
        <input type="checkbox" name="check" value="<%= task[i] %>">
        <%= task[i] %>
    </li>
<% } %>
```

6 Data Model

Task Model

A simple in-memory representation.

`TaskList: string[]`

No database is used; tasks are lost when server restarts.

7 Endpoint-Level LLD

7.1 GET /

- Renders the UI
- Sends current task list to index.ejs

7.2 POST /addtask

- Input: newtask (string)
- Adds a new task to the list

7.3 POST /removetask

- Input: check[] (selected tasks)
- Removes selected tasks from TaskList

8 Sequence Diagram

To-Do Workflow

```
User -> Browser: Load "/"
Browser -> Express: GET "/"
Express -> EJS: Render page with task list
EJS -> User: HTML UI

User -> Browser: Submit Add Task
Browser -> Express: POST "/addtask"
Express -> TaskList: Add new task
Express -> Browser: Redirect "/"

User -> Browser: Submit Remove Task
Browser -> Express: POST "/removetask"
Express -> TaskList: Remove selected tasks
Express -> Browser: Redirect "/"
```

9 Design Principles Applied

- **Separation of Concerns**: Logic, data, and UI are clearly separated.
- **Minimal State Management**: All tasks stored in a single array for simplicity.
- **Scalable Architecture**: Easy to extend to a database later (MongoDB, MySQL, etc.)
- **Clean UI Rendering**: EJS template loops render dynamic content.

10 Conclusion

This document describes the low-level structure and components of a Node.js To-Do List application. It outlines backend logic, tasks data model, UI template behavior, and full execution flow.

The system is simple, modular, and easy to extend with features like:

- Database integration
- User authentication
- Completed task history