Express.js Backend Project – Architecture & Implementation

Table of Contents

---

1. Aim of the Project

The aim of this project is to design and implement a **progressive, modular Express.js backend** that demonstrates real-world backend concepts such as:

- HTTP request/response handling
- Static file serving
- Middleware execution flow
- Modular routing
- Server-side rendering using EJS
- Form handling and query parameters

This project is intentionally structured to reflect **industry-level backend architecture**, rather than a single tutorial-based application.

---

2. Architecture Overview

This project follows a **learning-to-production progression model**.
Each server file introduces a new backend concept, allowing a clear understanding of how Express applications evolve.

Key architectural principles used:

- Separation of concerns
- Modular routing
- Middleware-based request flow
- Defensive rendering using default values
- Clean folder organization

This structure reflects how **real backend systems** are built and maintained.

---

3. Folder & File Structure

📌 **[Insert Folder Structure Image Here]**

The folder structure is organized into:

- **public/** – Static assets served directly by Express
- **routes/** – Modular route handlers
- **views/** – EJS templates for dynamic rendering
- **server1.js – server6.js** – Progressive server implementations
- **Guide.txt** – Developer learning & reference guide

This separation improves maintainability, scalability, and readability.

---

4. Static Files

index.html

Access URL:
`http://localhost:3000/`

Purpose:
Served directly from the `public` folder using `express.static`.
This file confirms that the server correctly delivers static content without route handling.

What it proves:

- Understanding of static middleware
- Browser-to-server static file flow

---

within_test.html

Location:
`public/test/within_test.html`

Access URL:
`http://localhost:3000/test/within_test.html`

Purpose:
Validates nested static file serving.

What it proves:

- Correct static folder configuration
- Understanding of URL-to-file mapping

Recruiter Insight:
Many beginners skip static serving. This file shows foundational backend knowledge.

---

5. Routing & Middleware – loggersjs.js

This router demonstrates **custom middleware** and **dynamic routing**.

Key Concepts Used:

- Custom logger middleware
- `router.use()` for middleware binding
- `router.param()` for request preprocessing
- Dynamic route parameters

Middleware Flow:

1. Request enters router
2. Logger prints requested URL
3. User data is attached via `router.param()`
4. Data is rendered using EJS

What it proves:

- Deep understanding of Express middleware lifecycle
- Clean request preprocessing

---

6. User Management Flow – server6users.js

This module implements **form-based user handling**.

Features:

- GET routes for form rendering
- POST routes for form submission
- Query parameter filtering (gender-based search)

- In-memory user storage
- Dynamic rendering using EJS

Key Backend Skills Demonstrated:

- Handling `req.body`
- Using `express.urlencoded()`
- Managing server-side state
- Redirecting and rendering based on user input

---

7. Modular Routing – users.js

This router demonstrates **REST-style routing**.

Concepts Used:

- `router.route()` for grouped HTTP methods
- Dynamic URL parameters
- `router.param()` for user preloading

Why this matters:

This structure closely matches how **real APIs** are designed in production systems.

Recruiter Insight:
Shows API design thinking, not just basic routing.

---

8. View Templates (EJS)

new.ejs

- Displays user input form
- Sends POST request to backend
- Demonstrates form-to-server data flow

---

showName.ejs

- Displays submitted user data
- Uses fallback values for safety
- Includes navigation back to form

---

index.ejs

- Renders dynamic user data
- Uses `locals` to prevent runtime errors

---

logsejs.ejs

- Displays logged user and URL information
- Demonstrates backend-to-view data flow

What EJS usage proves:

- Server-side rendering skills
- Defensive templating practices

---

9. Progressive Server Implementations

server1.js

- HTTP status codes
- JSON responses

server2.js

- EJS view engine setup
- Dynamic rendering

server3.js

- Modular routing using `routes/`

server4.js

- Middleware-based logging

server5.js

- Static file serving

server6.js

- Form handling
- User data processing

Recruiter Insight:
This progression clearly demonstrates **learning depth and intent**.

---

10. Developer Guide & Tooling

Tools Used:

- Node.js
- Express.js
- EJS
- Nodemon

Guide.txt Covers:

- Express fundamentals
- Middleware concepts
- Routing rules
- Debugging techniques
- Best practices

This reflects **self-driven learning and documentation discipline**.

---

11. Challenges & Solutions

Challenge: Route complexity

Solution: Modularized routes using Express Router

Challenge: Middleware execution order

Solution: Implemented layered logging middleware

Challenge: Form data handling

Solution: Used `express.urlencoded()` and defensive rendering

---

12. Conclusion

This project demonstrates:

- Strong Express.js fundamentals
- Clean backend architecture
- Middleware mastery
- Server-side rendering
- Real-world backend thinking

It serves as a solid foundation for **backend engineering or SDET roles**.

---

13. Author

Routh Kiran Babu
Backend & SDET Enthusiast
Focused on building scalable, testable backend systems