Deliverable B – SE D3

System Architecture

ProjectPulse

Group Members:

Tauha Imran	22i-1239,
Minahil Ali	22i-0849,
Nabeed Haider	22i-0871

Project Title: **Project Pulse**
Submission Date: 28th April, 2025

1. Identifying Subsystems

1.1 UML Package Diagram of Subsystems

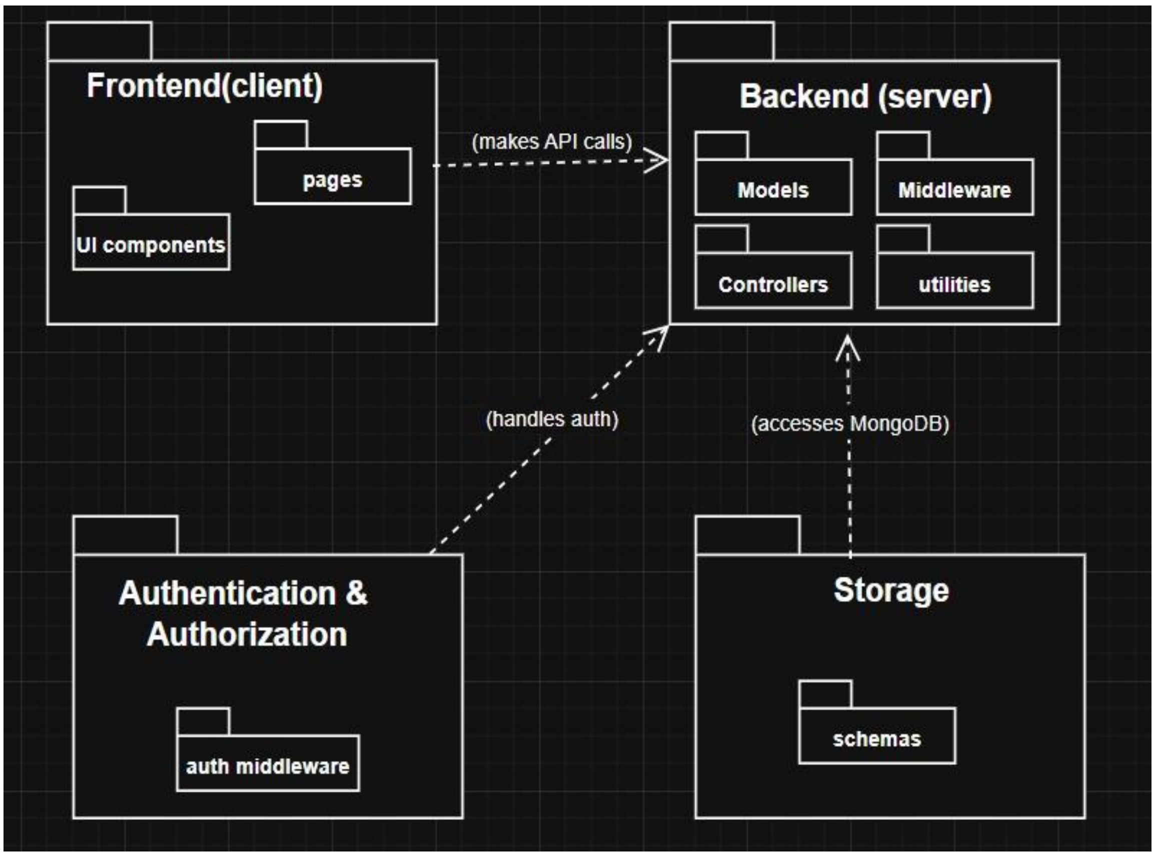

fig: package diagram.

1.2 Description

The system is broken down into the following logical subsystems (represented as UML packages):

- **Frontend Subsystem** (client)
    - Contains UI components, user interaction handlers, and Redux state management.
    - Includes packages: Components, Pages, Redux, and Utilities.

- **Backend Subsystem** (server)
    - Contains routing, middleware, controllers, models, and database connection logic.
    - Includes packages: Routes, Controllers, Middleware, Models, and Utils.

- **Database Subsystem**
    - Hosted on MongoDB.
    - Interacts through Mongoose models in the backend.
    - Includes packages: Routes, Controllers, Middleware, Models, and Utils.

- **Authorization & Authentication Subsystem**
    - authMiddleware.js: Ensures only authorized users can access certain routes.
    - Token-based authentication using JWT.
    - Registration and login logic in userController.js.
    - Ensures secure access and session management

These subsystems interact via API requests and database queries to ensure seamless functionality.

---

2. Architecture Styles

2.1 Client-Server Architecture

- **Description**:
  The application clearly follows the Client-Server model, where the frontend (React app under /client) acts as the client, and the backend (Node.js + Express under /server) acts as the server.

- **Responsibility Separation:**
    - The **client** handles user interactions, UI rendering, and sends HTTP requests.

- The **server** handles business logic, authentication, and communication with the database.

2.2 Model-View-Controller (MVC) (Backend)

- Description:
  The backend follows the MVC pattern:
  - Models: Defined in /models/ to handle MongoDB schemas (e.g., userModel.js, taskModel.js)
  - Views: Not applicable as it's an API server, but could return JSON responses
  - Controllers: Business logic is encapsulated in /controllers/
- Routes: Serve as the entry point for API requests, linking to appropriate controller functions

---

3. Deployment Diagram for Client Deployments

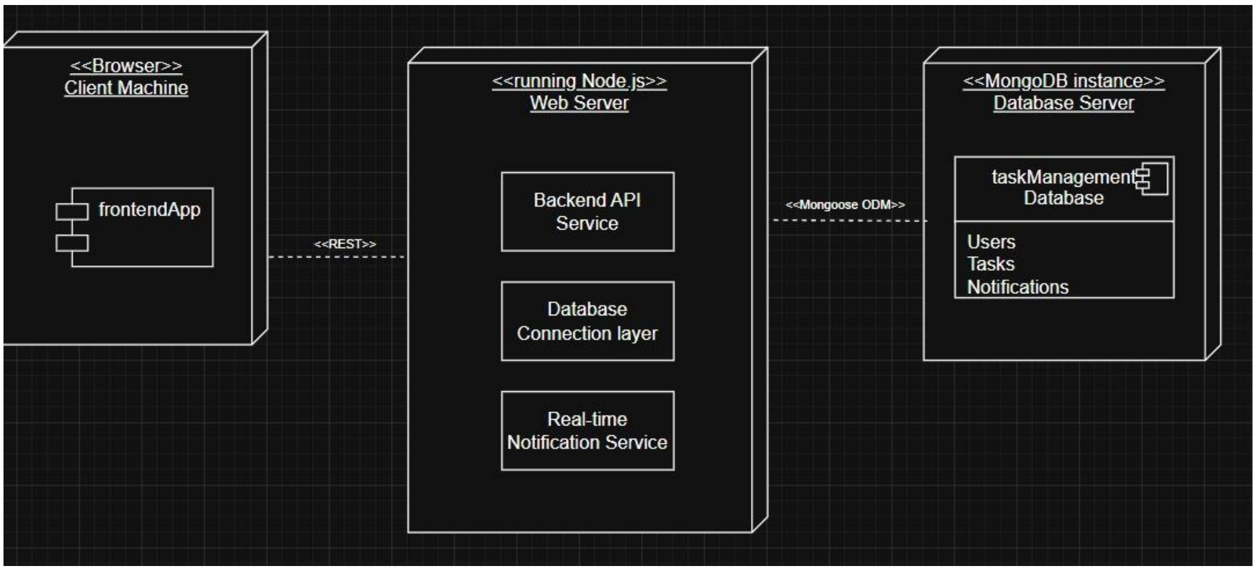

fig: deployment diagram.

3.1 Description:

The deployment diagram shows:

- A **Web Browser Node** running the compiled React app (built using Vite) served via a CDN like Netlify or Vercel.
- A **Node.js Server Node** deployed on a platform like Render or Railway, running the Express backend server.
- A **MongoDB Node** representing the database hosted in the cloud.

- **Communication Links**:
    - The browser connects to the Node.js server via HTTP/HTTPS.
    - The Node.js server communicates with the MongoDB via a secure connection using the Mongoose ODM.

---

4. Component Diagram

A component diagram .

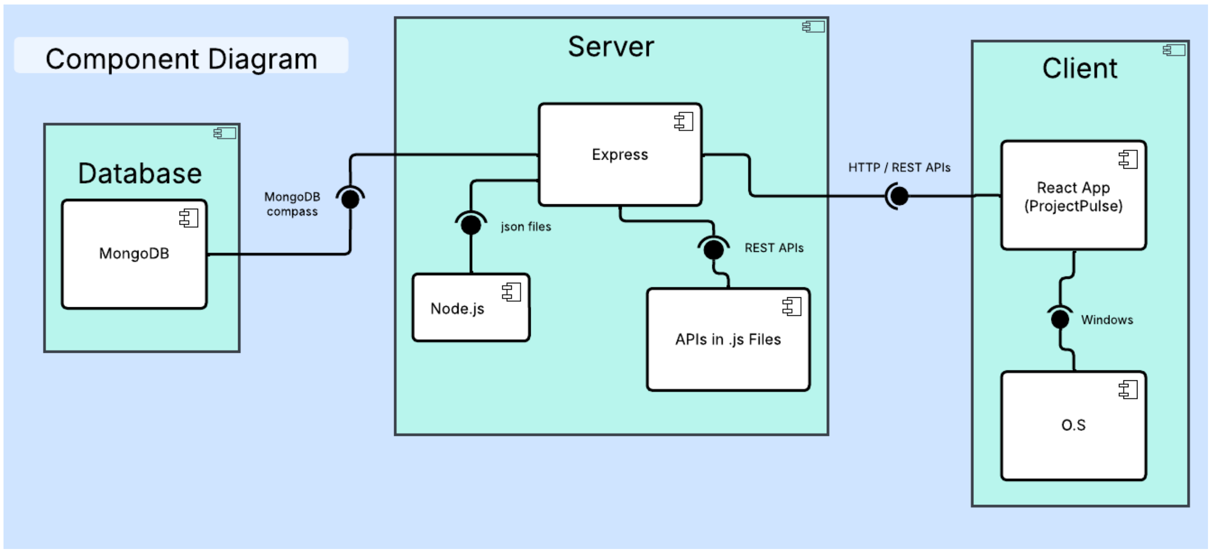

fig: component diagram.

It has three major components:

1. Database Component

- **MongoDB**: The primary database system being used
- **MongoDB Compass**: MongoDB's graphical user interface for interacting with the database

2. Server Layer

- **Express**: The web application framework running on Node.js
- **Node.js**: The JavaScript runtime environment

3. Client Layer

- **React App (ProjectPulse)**: The frontend application built with React
- **Windows OS**: Indicates the client runs on Windows operating systems
- **HTTP/REST APIs**: Shows communication between client and server via RESTful APIs