Architectural Pattern: Three-Tier for BUP Alumni System

1. Introduction

The BUP Alumni System is a web-based platform designed to connect the former students of Bangladesh University of Professionals (BUP). It enables users to register, log in, update their profiles, view other members, participate in events, and engage within the alumni network. To ensure the system remains organized, scalable, and secure, an appropriate architectural pattern is essential.
The project uses the following technology stack:

- **Languages:** HTML, CSS, JavaScript
- **Frontend Framework:** React.js with Tailwind CSS
- **Backend Framework:** Node.js with Express.js
- **Database:** MySQL
- **Editor:** Visual Studio Code

Considering the system's design goals and chosen technologies, the **Three-Tier Architecture** is the most suitable for this project.

2. What is an Architectural Pattern?

An architectural pattern is a reusable solution to a recurring software design problem. It defines how software components should be structured and how they interact with each other. Architectural patterns improve system organization, scalability, and maintainability by enforcing a clear separation of responsibilities.
Common patterns include:

- Model-View-Controller (MVC)
- Three-Tier Architecture
- Client–Server Architecture
- Event-Driven Architecture

Each serves different purposes, but the choice depends on the project scale, complexity, and resource constraints.

3. Why an Architectural Pattern is Needed for Our Project

Using an architectural pattern provides structure and long-term stability for the BUP Alumni System. Key reasons include:

1. **Separation of Concerns:** Divides the system into independent layers — Presentation, Application, and Data — making the project modular.

2. **Maintainability:** Developers can modify one layer without affecting others, simplifying updates and debugging.

3. **Scalability:** Each layer can be independently optimized or expanded as the system grows.

4. **Security:** The database is isolated behind the application layer, preventing direct user access.

5. **Reusability:** Backend APIs and database logic can be reused for mobile apps or future extensions.

For these reasons, the Three-Tier Architecture is suitable for building a reliable and secure BUP Alumni System.

4. What is Three-Tier Architecture ?

The Three-Tier Architecture (also called the Layered Architecture) is a well-structured software design pattern that divides an application into three separate layers, each responsible for a specific function.

Presentation Layer (Frontend)

- This is the topmost layer of the application that directly interacts with the users.
- It displays the information to the users and collects their inputs.
- Technologies: *React.js*, *Tailwind CSS*, *HTML*, *CSS*, *JavaScript*.
- Example: Login page, registration form, and alumni profile dashboard.

Application Layer (Backend / Business Logic)

- This is the middle layer that acts as a bridge between the frontend and the database.
- It handles all the business logic, processes user requests, and controls data flow between the user interface and the database.
- In our project, this layer is developed using Node.js and Express.js.

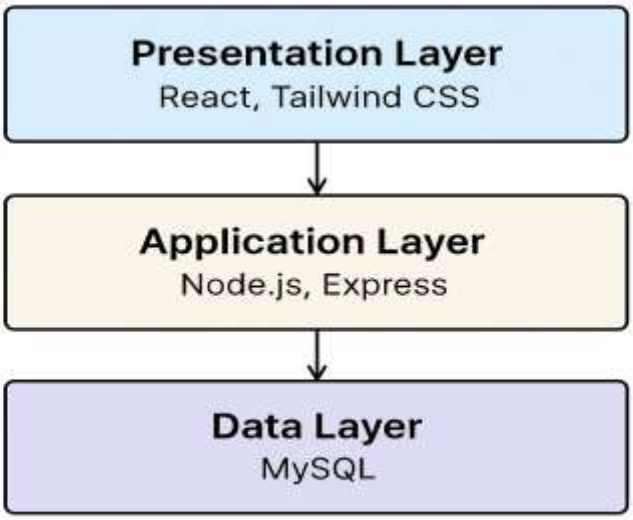
- Example: When a user logs in, this layer checks whether the entered email and password match with the data stored in the database.
- Technologies: Node.js, Express.js.

Data Layer (Database)

- This is the bottom layer of the architecture that stores, retrieves, and manages all the application data.
- It ensures that the data is properly organized and secured.
- In our project, MySQL is used as the database system.
- Example: It stores user information, such as name, email, password, and alumni details. Manages data storage, retrieval, and updates.
- Technology: MySQL.

Each of these Three tiers works independently but communicates with each other in an organized way. The frontend never directly interacts with the database; instead, it sends requests to the backend (application layer), which then communicates with the database. This layered structure makes the system more organized, secure, maintainable, and scalable, as changes in one layer (for example, updating the UI or modifying the database) do not directly affect the others.

5. Three-Tier Architecture Diagram



6. How Three-Tier Architecture Works

The Three-Tier Architecture works through a smooth flow of information between the three layers — Presentation, Application, and Data. Each layer has a specific role, and together they ensure that the system operates efficiently, securely, and in an organized manner.

Below are the five main steps explaining how it works in the BUP Alumni System:

Step 1 — User Interaction (Frontend):
The user interacts with the system through the React.js interface, such as by entering login details and clicking the login button. The frontend sends this information to the backend through an API request.

Step 2 — Request Handling (Application Layer):
The Node.js/Express.js backend receives the request from the frontend and identifies what operation needs to be performed, such as verifying user credentials.

Step 3 — Business Logic Execution:
The backend applies the required business logic, such as checking whether the input fields are valid and preparing a query to fetch matching data from the database..

Step 4 — Data Access (Database Layer):
The backend communicates with the MySQL database to check if the provided email and password exist. The database sends back the result (valid or invalid).

Step 5 — Response Back to Client:
The backend sends the final response to the frontend, which then displays a suitable message — for example, “Login Successful” or “Invalid Credentials.”

7. Example File Structure (Three-Tier Implementation)

```
bup-alumni-system/
│
├── backend/                    # Application Layer
│   ├── server.js
│   │
│   ├── config/
│   │   └── db.js
│   │
│   ├── routes/
│   │   ├── login-route.js
│   │   ├── signup-route.js
│   │   ├── profile-route.js
│   │   └── event-route.js
│   │
```

```
    └── controllers/
        ├── loginController.js
        ├── signupController.js
        ├── profileController.js
        └── eventController.js

├── database/                    # Data Layer
    └── bupalumni.sql

├── frontend/                    # Presentation Layer
    └── html
        ├── login.html
        ├── signup.html
        ├── profile.html
        └── event.html
    └── CSS
        ├── login.html
        ├── signup.html
        ├── profile.html
        └── event.html

└── README.md
```

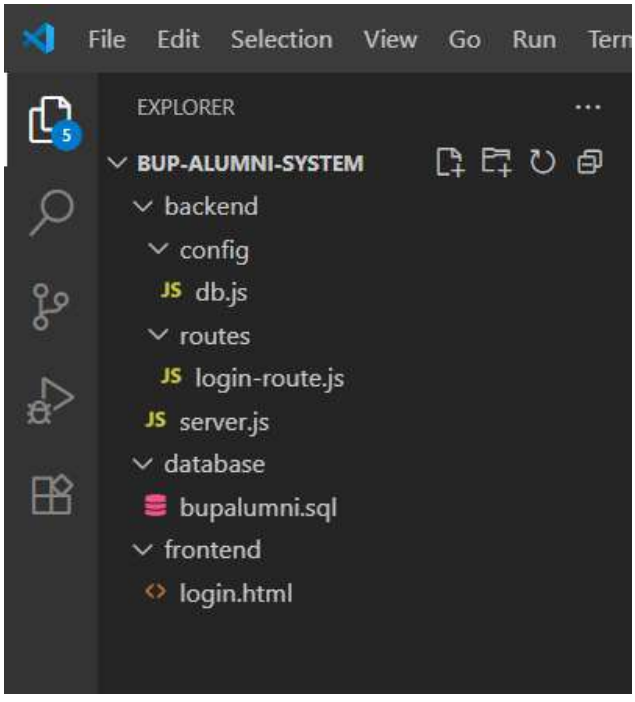
8. Step-by-Step Implementation Example:
Login Implementation

Step 1: Create Project Folder

Action: Create a main folder named bup-alumni-system in VS Code.

Description: This folder will hold all files — frontend, backend, and database setup.

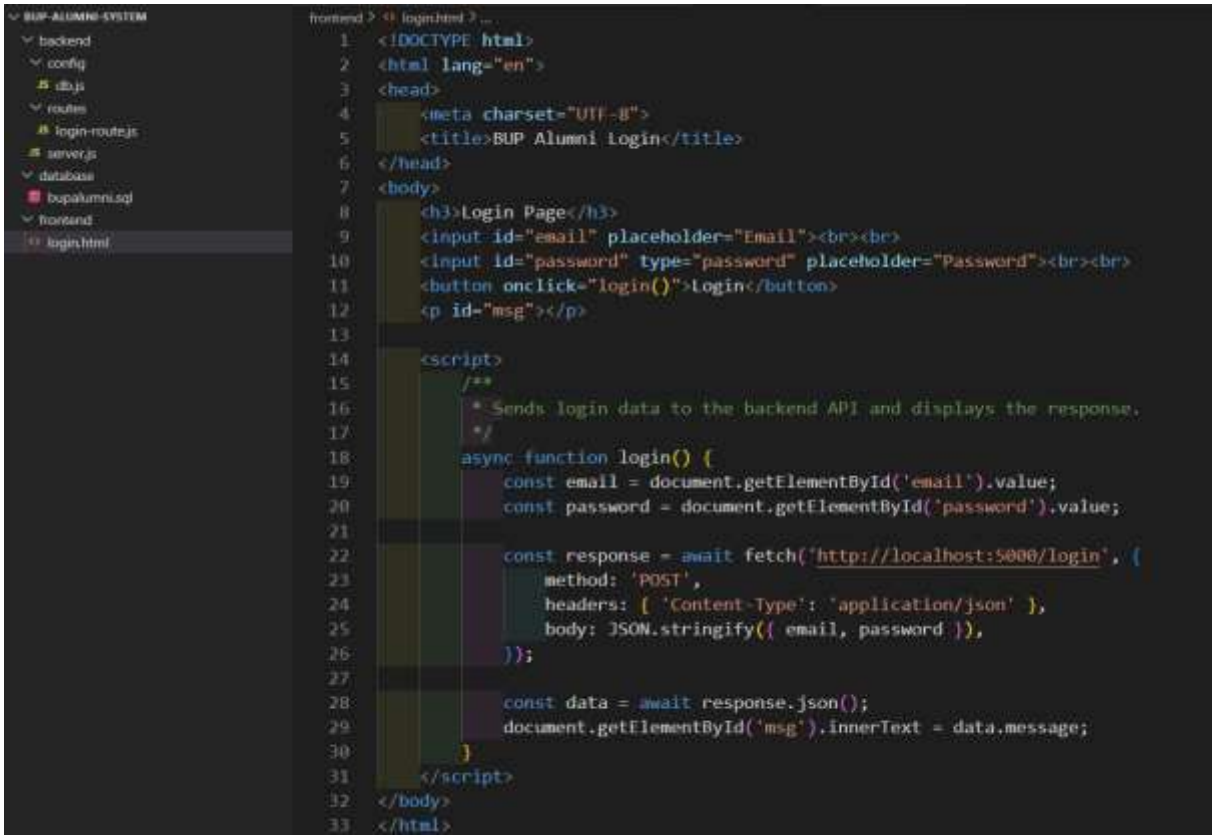
Folder Structure:



Step 2: Create the Frontend (User Interface)

File: frontend/login.html

Description: This page collects user input (email and password) and sends it to the backend.



Step 3: Set Up the Database

File: database/bupalumni.sql

Description: This SQL file creates the database and the users table for login.

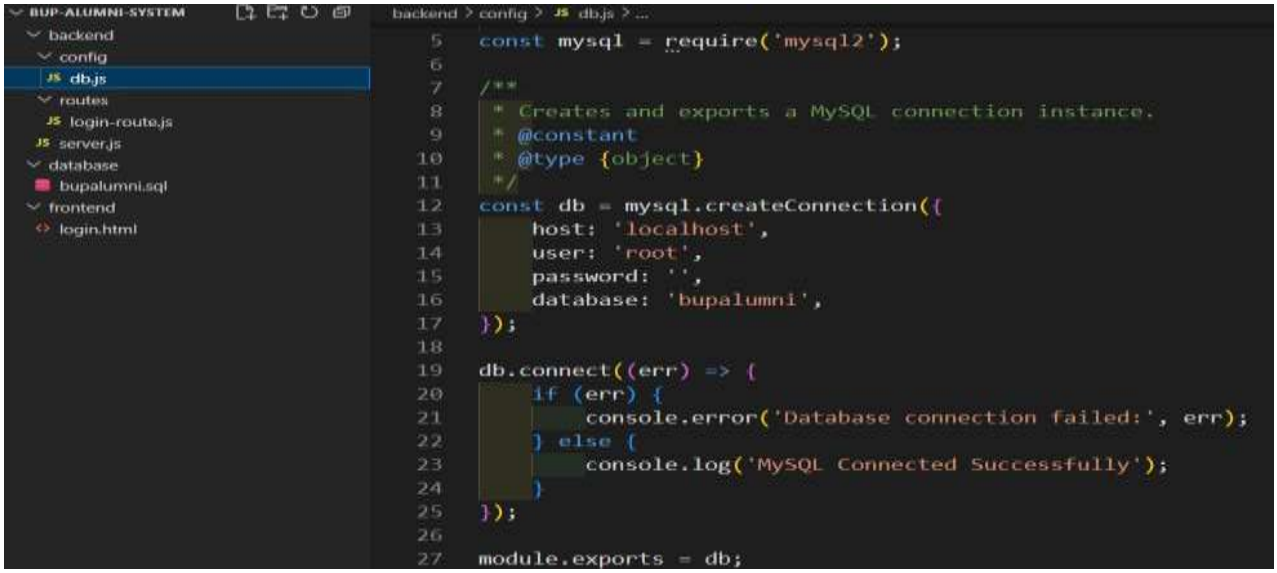
Steps:

- Open XAMPP or MySQL Workbench.
- Create a new database named bupalumni.
- Run this SQL code.

Step 4: Set Up Backend Configuration

File: backend/config/db.js

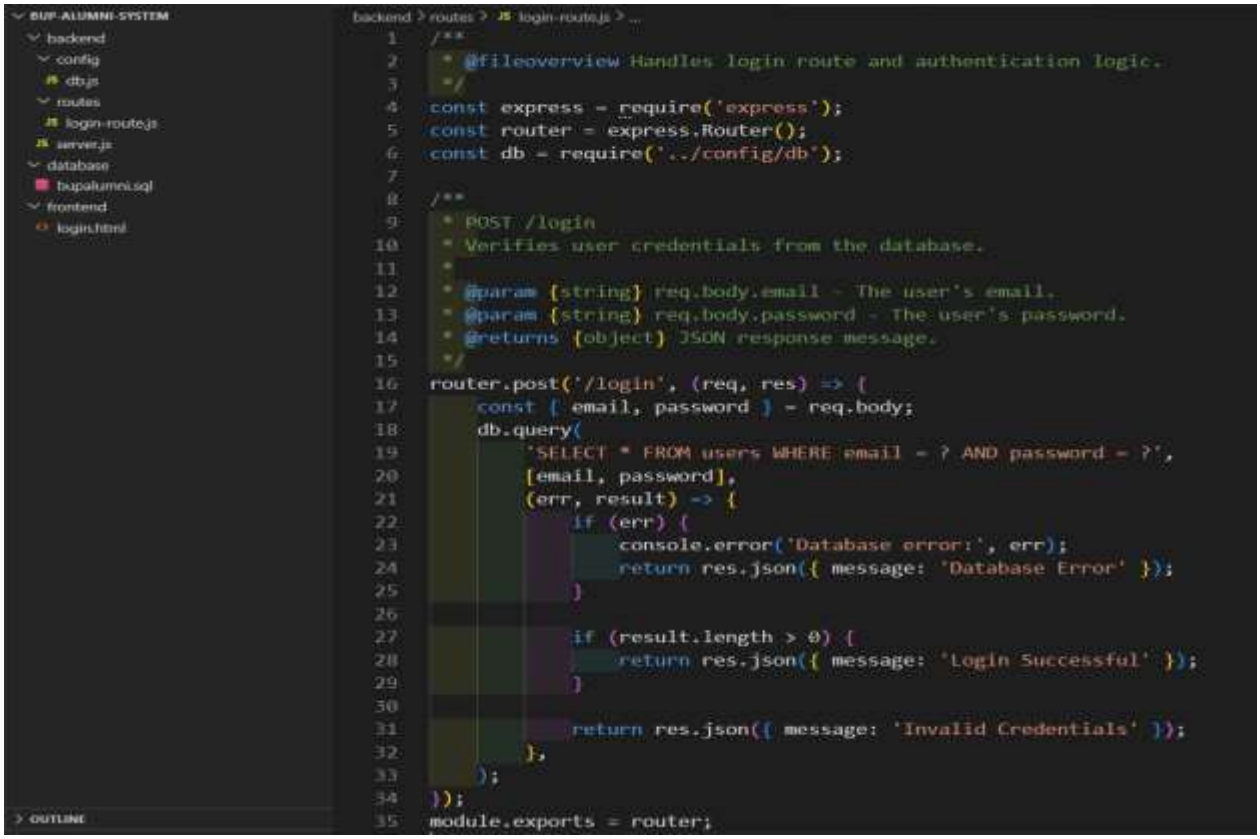
Description: This file connects the Node.js backend to the MySQL database.



Step 5: Create the Login Route

File: backend/routes/login-route.js

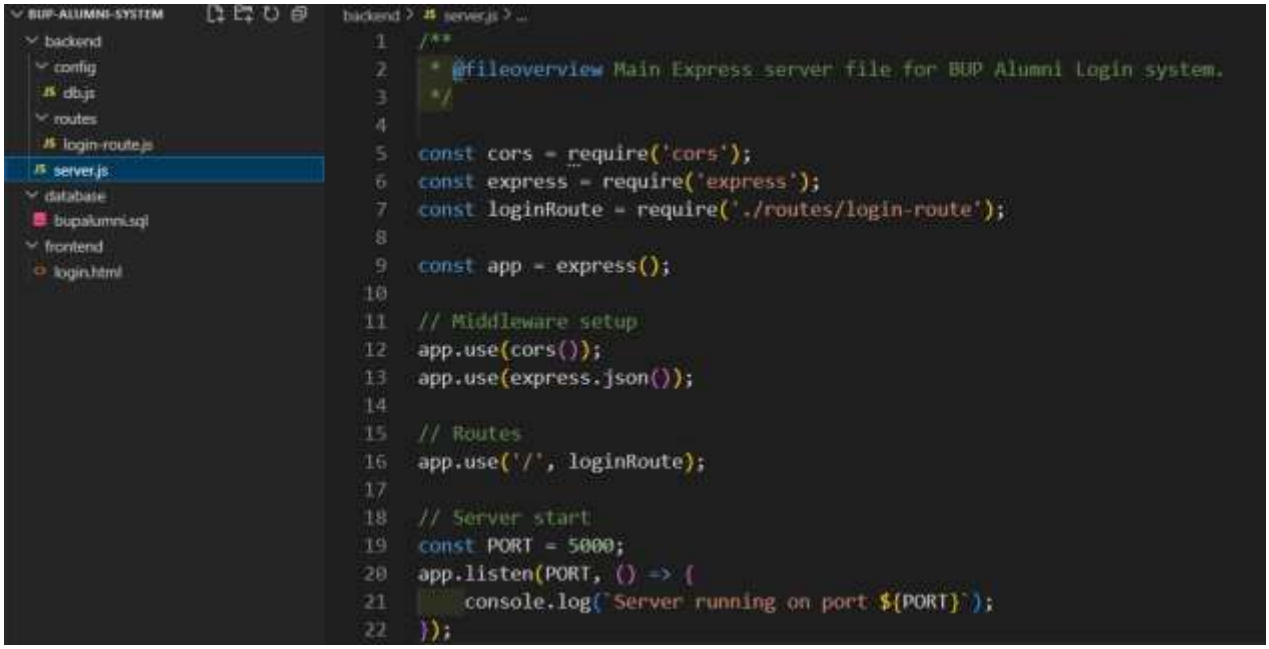
Description: This file handles the login request sent from the frontend.



Step 6: Create the Main Server File

File: backend/server.js

Description: This is the main file that runs the backend server.



Step 7: Install Required Packages

Description: These packages help connect and run the backend server.

Steps:

- Open VS Code Terminal.
- Run the following commands:

  cd backend

  npm init -y

  npm install express mysql2 cors

Step 8: Run the Backend Server

Description: This starts your backend so that the frontend can send login requests.

Steps: In VS Code Terminal, run:

node server.js

9. Cohesion and Coupling in the BUP Alumni System

Cohesion:

Cohesion refers to how closely the components within a single module or layer are related to one another. In a highly cohesive system, all elements of a module work together to perform a specific task effectively.

In Our Project:

The BUP Alumni System maintains strong cohesion because each layer focuses on a well-defined responsibility:

- **Presentation Layer:** Handles all user interactions and UI rendering.
  Example: The Login Page component manages only user input and displays it doesn't contain business logic or database operations.
- **Application Layer:** Manages authentication, data validation, and communication with the database.
  Example: The loginController.js file handles login verification logic only, without dealing with how data is displayed.
- **Data Layer (MySQL):** Stores and retrieves alumni data.
  Example: The bupalumni.sql file organizes all alumni information (name, email, password, events) without handling any UI or logic code.

This separation ensures that each layer's internal elements work together toward a single purpose — achieving high cohesion.

Coupling:

Coupling refers to the level of dependency between different modules or layers. Low (loose) coupling means changes in one module have minimal impact on others.

In Our Project:

The BUP Alumni System maintains loose coupling because each layer interacts only through defined interfaces or APIs:

- The front end never directly connects to the database; it communicates with the backend using RESTful APIs (/login, /signup, /profile).
- The backend interacts with the database using SQL queries via a configuration file (db.js), without knowing database implementation details.
- If the database changes (MySQL), only the data access code in the backend needs modification — the frontend remains unaffected.

Example:
If we replace the existing MySQL database with a NoSQL database, only the backend's db.js and query logic would need updating. The React frontend would still send the same API requests, showing loose coupling between layers.

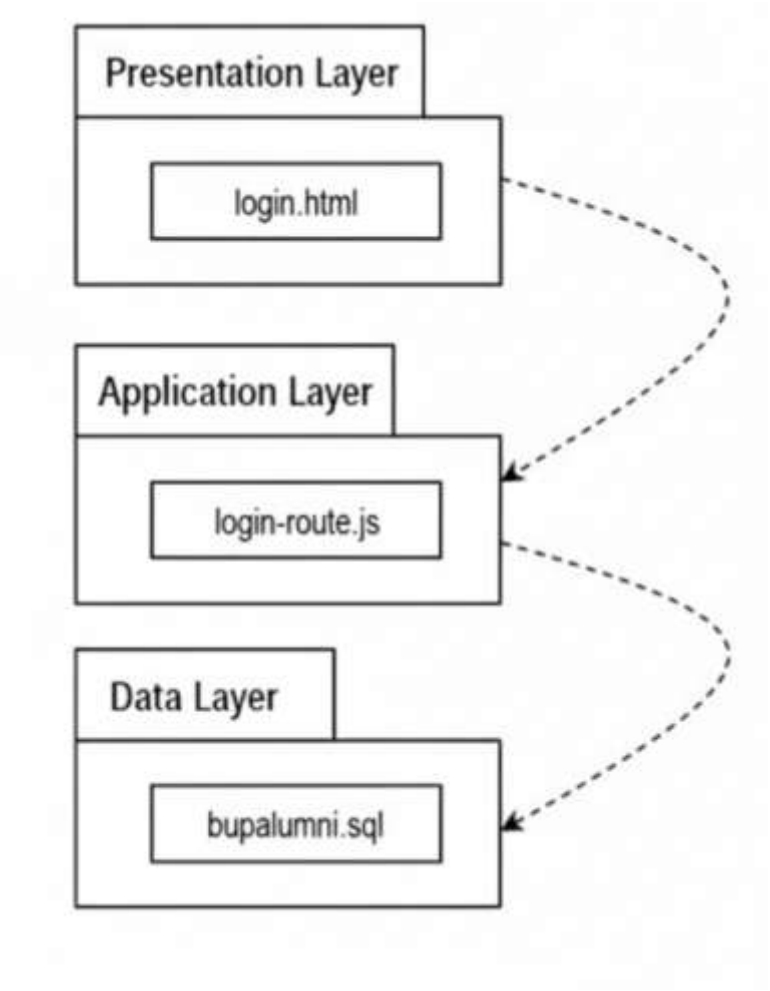


Fig: Cohesion and Coupling example

The figure shows that each layer of the BUP Alumni System has high cohesion because all components within a layer work together for one purpose — for example, login.html handles only user input, login-route.js processes login logic, and bupalumni.sql manages data. It also shows low coupling, as layers communicate through defined interfaces instead of direct connections. This makes the system modular, easy to maintain, and scalable.

10. Advantages of Three-Tier Architecture

- **Separation of Concerns:** Each layer focuses on one responsibility, leading to cleaner and more modular code.
- **Scalability:** Developers can upgrade the frontend, backend, or database independently as the system grows.
- **Maintainability:**Bugs or updates can be fixed in one layer without affecting others.
- **Security:** The database is fully protected behind the backend, minimizing unauthorized access.
- **Reusability:** APIs and business logic can be reused for mobile or other platforms in the future.

11. Disadvantages of Three-Tier Architecture

- **Slight Performance Overhead:** Requests must pass through multiple layers, causing minimal delay.
- **Setup Complexity:** Requires careful configuration for communication between layers.
- **Increased Development Time:** Beginners may find managing multiple tiers slightly challenging initially.

12. Conclusion

The Three-Tier Architecture provides a strong foundation for the BUP Alumni System, combining clarity, modularity, and long-term scalability. It aligns perfectly with the selected technology stack — React.js for the Presentation Layer, Node.js/Express.js for the Application Layer, and MySQL for the Data Layer. This design ensures that the system remains easy to develop, maintain, and extend in future versions, making it the most appropriate architectural pattern for our project.

13. References:

- What is three-tier architecture?
  https://www.ibm.com/think/topics/three-tier-architecture
- Three-Tier Client Server Architecture
  https://www.geeksforgeeks.org/computer-networks/three-tier-client-server-architecture-in-distributed-system/