## Openticket Project:-

**Project Overview :**

OpenTicket is designed to be a ticketing system where users can create, view, and manage tickets. It can be tailored for customer support, event booking, or any situation requiring ticket generation and management.

**Technology Stack :**

1. **Front-end** : React with Bootstrap for responsive design and UI elements.

2. **Back-end :** Express.js to handle HTTP requests and connect front-end with database.

3. **Database :** MySQL to store ticket data, user information, and logs.

**Project Structure:-**

1. **Front-end (React + Bootstrap):**

**Components :**

**Ticket Form :** Form for creating new tickets (fields for title, description, priority, etc.).

**TicketList :** Displays a list of tickets in a table format with sorting and filtering options.

**TicketDetail :** Shows individual ticket details and allows updates or comments.

**Pages :**

**Dashboard :** Summary of open, closed, or pending tickets with some visual stats.

**TicketPage :** Contains `TicketList` and filters, search options for managing tickets.

**UserProfile :** Page for users to manage their profile details.

**UI Enhancements :** Use Bootstrap components (Cards, Modals, Forms) to build a consistent, responsive layout.

1. **Back-end (Express.js) :**

**Routes :**

- /api/tickets :

- `POST`: Create a new ticket.

- `GET`: Get all tickets (optionally filtered by status or priority).

- `GET /:id`: Get details for a single ticket.

- `PUT /:id`: Update ticket information.

- `DELETE /:id`: Delete a ticket.

- /api/users :

- User management routes (sign-up, login, profile management).

**Middleware :**

- Authentication middleware (JWT or session-based).

- Validation for request data (e.g., using `express-validator`).

**Error Handling :**

- Centralized error handler to send structured error responses.

1. **Database (MySQL) :**

Tables :

**Users :**

Stores user details (user\_id, username, password, role, etc.).

**Tickets :**

Stores ticket information (ticket\_id, title, description, status,

priority, user\_id as foreign key).

**Comments :**

For user comments on tickets (comment\_id, ticket\_id,

user\_id, content, timestamp).

**Relationships :**

**One-to-Many :** A user can have multiple tickets; a ticket can have multiple comments.

**Key Functionalities :**

**1. User Authentication :**

- Allow users to sign up and log in.

- Role-based access (e.g., Admin, Support Staff, User).

**2. Ticket Creation and Management :**

- Users can create, edit, and delete tickets.

- Filters by status (open, closed, pending), priority, and search functionality.

**3. Comment System :**

- Users can add comments on tickets to provide updates or communicate.

**4. Dashboard :**

- Shows a summary of ticket statistics (open vs. closed tickets, priority distribution).

**Front-end & Back-end Interaction :**

1. **API Integration :**

Use Axios or Fetch API in React components to interact with Express API routes.

1. **State Management :**

Utilize React's state or context API to manage ticket data, user info, and authentication tokens.

**Setup Notes :**

**1. Environment Configuration :**

- `.env` files for sensitive information like MySQL connection credentials, JWT secret, etc.

**2. Database Connection :**

- Use a library like Sequelize or knex.js for ORM or direct MySQL connection through `mysql2` for query building.

**3. Testing :**

- Unit tests for back-end API s (Jest or Mocha with Supertest).

- Front-end component tests using Jest and React Testing Library.