Full Stack Plan for Ticket System (React & Flask)

Overview

The system consists of a React frontend and a Flask backend integrated with MongoDB. It allows users to create support tickets and enables admins to manage them securely with JWT authentication.
Sample Flask Code : https://github.com/sabarinathan1611/RootRS-Ticket-System/

Technology Stack

Frontend (React)

- **React (Frontend framework)**
- **React Router (Navigation)**
- **Redux / Context API (State management for authentication & tickets)**
- **Axios (API calls)**
- **Material UI / Tailwind CSS (Styling)**
- **JWT Authentication (Handling tokens)**

Backend (Flask)

- **Flask (Lightweight Python framework)**
- **Flask-JWT-Extended (Authentication)**
- **Flask-Bcrypt (Password hashing)**
- **Flask-CORS (Cross-Origin Resource Sharing)**
- **MongoDB (Database for storing tickets & users)**

Backend Implementation

API Endpoints

Method	Endpoint	Description
POST	/create-ticket	Create a new support ticket
POST	/admin-login	Admin authentication, returns JWT token
GET	/tickets	Get all open tickets (Admin only)

Method	Endpoint	Description
GET	/closed-tickets	Get all closed tickets (Admin only)
GET	/search?query=...	Search tickets (Admin only)
POST	/close-ticket/<id>	Close a ticket (Admin only)

MongoDB Collections

- **admins (Stores admin credentials)**
- **tickets (Stores user-submitted support tickets)**

Page Structure

1. Login Page (Admin Login)

- Fields: Username, Password
- Uses Axios to send a POST request to /admin-login
- Stores JWT token in localStorage
- Redirects to Dashboard on successful login

2. Create Ticket Page (Public Access)

- Fields: Name, Mobile
- Uses Axios to send a POST request to /create-ticket
- Displays the generated ticket number on success

3. Dashboard (Admin Only)

- Uses JWT for authentication (Token required for API calls)
- Fetches and displays open tickets from /tickets
- Fetches and displays closed tickets from /closed-tickets
- Search bar to query tickets from /search
- Button to close tickets by sending a POST request to /close-ticket/:ticket_number

---

API Integration Plan

1. Authentication (Admin Login)

```
axios.post('/admin-login', { username, password })
  .then(response => {
    localStorage.setItem('token', response.data.token);
    navigate('/dashboard');
  })
  .catch(error => alert('Invalid Credentials'));
```

2. Create Ticket (Public Access)

```
axios.post('/create-ticket', { name, mobile })
  .then(response => alert(`Your ticket number: ${response.data.ticket_number}`))
  .catch(error => alert('Error creating ticket'));
```

3. Fetch Open Tickets

```
axios.get('/tickets', { headers: { Authorization: `Bearer ${token}` }})
  .then(response => setTickets(response.data))
  .catch(error => alert('Error fetching tickets'));
```

4. Search Tickets

```
axios.get(`/search?query=${searchTerm}`, { headers: { Authorization: `Bearer ${token}` }})
  .then(response => setSearchResults(response.data))
  .catch(error => alert('Error searching tickets'));
```

5. Close Ticket

```
axios.post(`/close-ticket/${ticket_number}`, {}, { headers: { Authorization: `Bearer ${token}` }})
  .then(response => alert('Ticket Closed!'))
  .catch(error => alert('Error closing ticket'));
```

State Management Plan

- Use React Context API / Redux for managing authentication and tickets.
- Store JWT token in localStorage and set Authorization headers for API calls.
- Maintain tickets state to avoid redundant API calls.

Component Breakdown

1. Auth Components

- Login.js (Handles Admin Login)
- AuthContext.js (Manages JWT authentication)

2. Ticket Components

- CreateTicket.js (Public Ticket Creation Form)
- TicketList.js (Displays open/closed tickets)
- SearchTicket.js (Search Bar & Results)

3. Admin Components

- Dashboard.js (Admin Panel)
- CloseTicketButton.js (Button to close ticket)

Deployment Plan

- Frontend: Deploy React app using Vercel / Netlify
- Backend: Host Flask API on AWS / Heroku / DigitalOcean
- Database: Use MongoDB Atlas for cloud-based storage