**Sujan Kumar-Booking Page**

Booking Page Documentation

1. Overview

The Booking Page is the central feature of a Navatar application. It enables users to book, view, and manage their consultation slots with ease. The system is designed for both user-friendly interaction and robust backend reliability.

2. What Was Implemented

Backend

* API Development:
  + Built using FastAPI, a modern Python web framework.
  + Provides endpoints for creating, viewing, and canceling bookings.
* Database Integration:
  + Initially used SQLite for local development and testing.
  + Migrated to a cloud-based PostgreSQL database (Neon) for production, ensuring scalability and reliability.
* Cloud Deployment:
  + Deployed the backend to Vercel, leveraging serverless functions for efficient scaling.
  + Managed environment variables for secure database access and configuration.
* CORS Configuration:
  + Configured CORS middleware to allow secure communication between the frontend and backend, resolving cross-origin issues.

Frontend

* User Interface:
  + Developed with React and Vite, offering a responsive and intuitive booking interface.
  + Users can select dates, choose flexible time slots, and manage their bookings.
* Booking Logic:
  + Implemented logic for checking overlapping bookings and preventing double bookings.
  + Added validation for past and overlapping time slots.
* Notification System:
  + Enhanced with reminder notifications for upcoming consultations.
  + Added loading and processing indicators during booking and cancellation, improving user experience and preventing multiple submissions.
* Error Handling:
  + Clear error messages and feedback for user actions, such as invalid time slots or failed booking attempts.

3. How It Works

Booking Flow

1. User Authentication:
   * Users log in using their credentials.
2. Booking Creation:
   * Users select a date and start-time and end-time.
   * The system checks for overlaps and validity before confirming the booking.
3. Booking Management:
   * Users can view their upcoming bookings.
   * Bookings can be canceled at any time.
4. Notifications:
   * Users receive reminders for upcoming consultations.
   * Loading indicators are shown during booking and cancellation processes.

Technical Highlights

* Backend:
  + FastAPI handles all booking logic and data persistence.
  + PostgreSQL on Neon ensures reliable and scalable data storage.
  + CORS middleware allows secure cross-origin requests between the frontend and backend.
* Frontend:
  + React provides a dynamic and responsive user interface.
  + Vite ensures fast development and build times.
  + Notification and loading logic enhance user experience and prevent errors.

4. User Experience

* Intuitive Interface:
  + Simple navigation and clear feedback for user actions.
* Reliable Operations:
  + Robust error handling and validation.
  + Instant feedback during booking and cancellation.
* Timely Reminders:
  + Users are notified of upcoming consultations, reducing missed appointments.

Repository Links

* Backend (FastAPI):  
  <https://github.com/Suja2004/Navatar>
* Frontend (React/Vite):  
  <https://github.com/Suja2004/Navtar>
* Project Documentation

**Yogin Kumar ,Shashank, Srushti Rao-Hardware**

# Modules Completed

## 1. Hardware Setup

* - Successfully completed the wiring of the four-wheel drive bot.  
  - Motors and wheels are properly connected and tested.  
  - Control is currently operational using TX and RX serial communication.

## 2. Frontend Interface

* - A web-based frontend has been developed.  
  - The interface includes joystick-based control elements.  
  - Communication between frontend and backend is handled using Pub/Sub pattern.

## 3. MQTT Integration

* - Implemented MQTT server to act as a bridge between the frontend and the ESP32.  
  - Remote access and control logic over the network are functioning correctly.  
  - MQTT topics are being used to send directional commands from the frontend to the bot.

# Next Steps / Modules To Be Completed

## 1. ESP32 Wiring & Integration

* - Wire the ESP32 directly to the motor driver for control signal transmission.  
  - Ensure all pins are properly mapped for directional motor control.

## 2. ESP32 Motor Control Code

* - Write and upload the code to the ESP32 to receive MQTT messages.  
  - Decoded messages as to be translated into motor control signals (e.g., forward, backward, left, right).  
  - Implement basic speed control if required.

**Justin Aroza**

Overview of the Module

The module I am working on is the Authentication system for the Navatar web application. It enables users (specifically doctors) to sign up, log in, and securely access protected parts of the application like video consultations and appointment booking.

We are using Clerk as the authentication service provider. Clerk provides a secure, plug-and-play solution for authentication, which includes:

* Prebuilt UI for sign-in and sign-up
* Email verification with 6-digit code (email code)
* Session management
* Account management UI
* React hooks for user data and session control

Completed Features

1. Sign-In and Sign-Up Pages

* Developed two dedicated components:

Login.jsx → For users to log in

Signup.jsx → For new users to register

* Clerk's prebuilt components (<SignIn /> and <SignUp />) are used for:

Email address collection

Sending verification codes

Creating user sessions after authentication

* Both components are wrapped in styled containers (.login-page, .login-container, .login-card) for consistent UI.

Code Example: Login.jsx

<SignIn

signUpUrl="/signup"

forceRedirectUrl="/landingpage"

/>

Code Example: Signup.jsx

<SignUp

signInUrl="/login"

forceRedirectUrl="/landingpage"

/>

2. Account Management in Navbar

* Integrated Clerk's UserButton component in the Navbar.jsx for signed-in users.
* When signed in, users will see:

A welcome message

A dropdown account menu with options to manage their account and log out

Dynamic Display Based on Session:

* SignedIn → Shows user info and UserButton
* SignedOut → Shows “Login” button

3. Route Protection

* Built a ProtectedRoute.jsx component using Clerk’s useAuth() hook to protect sensitive pages like:

/booking

/consultation

Unauthenticated users are automatically redirected to the login page.

Code Snippet:

if (!isSignedIn) {

return <Navigate to="/login" replace />;

}

4. Routing Integration

* The authentication routes (/login, /signup) and protected routes are properly added in App.jsx.
* After login or sign-up, users are redirected to /landingpage.

5. Session and Verification Code Handling

* Clerk handles session persistence and verification code (OTP) delivery.
* The sign-in and sign-up flows are fully functional.
* The UI responds correctly to state changes (SignedIn/SignedOut).

The Authentication module is integrated using Clerk, and operational with email code verification, secure login/signup, and user session handling.