HEALTHBRIDGE: Care bridge to better healthcare

1. User Authentication

Objective:

Provide a secure and seamless authentication system for doctors using Firebase Authentication. Doctors can log in or sign-in using Google.

Approach:

Firebase Authentication Integration:

- **Google Login**: Use Firebase's built-in Google Authentication for a quick and reliable sign-in experience.

Flow:

1. **Google Login:**
    - Firebase Authentication is integrated with Google Sign-In.
    - When a doctor logs in, Firebase generates a unique ID token for authentication.
    - The front end sends this token with API requests for user identification.
2. **Token Management:**
    - Firebase ID tokens are included in each API request header (Authorization: Bearer <Firebase Token>).
    - The server validates these tokens using Firebase Admin SDK.

2. Patient Onboarding

Objective:
Collect essential patient details securely, including personal and medical information.

Implementation Details:

Authentication and Initial Login:

- Use **Firebase Authentication** to handle user registration and login.
- Upon successful login, basic user data (e.g., email, UID, displayName) will be fetched from Firebase and stored in the database for reference.

Schema Design:

- **Database**: Use **MongoDB** for patient records for scalability and flexibility.
- **Patient Table Schema**:
    - Fields include:
        - firebaseUID: Link to the Firebase user.
        - name, age, gender, contactDetails, medicalHistory, and allergies.
        - Optional fields for future enhancements, such as profilePicture etc.

Form Design:

- **Frontend**:
    - Build a patient form using **React**.
    - Pre-fill fields like email and UID using data fetched from Firebase.
    - Enable updating additional fields (e.g., medical history) through a dedicated **Profile Page**.
- **API Integration**:
    - The form will connect to a secure API endpoint to create or update patient records in MongoDB.

Data Validation:

- Use backend validation with **Express-validator** to ensure all required fields are accurate and complete.
- Frontend validation for real-time feedback to the user.

Secure Submission:

- **Data Transmission**: Use **HTTPS** for encrypted communication.
- **Token-Based Security**:
    - Leverage **Firebase Tokens** instead of JWT for authenticating API requests.
    - Verify the Firebase token on the backend to ensure requests are secure and associated with authenticated users.

API Endpoints:

- **POST /api/patients**: Create or update a patient record.
- **GET /api/patients**: Retrieve all patient records.
- **PUT /api/patients/:patientId**: Update specific patient by their Firebase UID.
- **DELETE /api/patients/:patientId**: Update specific patient by their Firebase UID.

3. Dashboard with AI Caller Integration

Objective:
Build a comprehensive dashboard that displays patient analytics and integrates **Retell AI Caller** for handling patient-related inquiries.

Features:

Patient Overview Analytics

1. **Common Medical Conditions**:
    - Analyze patient records to identify and display the most frequently reported medical conditions.
2. **Recent Patient Onboarding**:
    - Fetch and display the most recently onboarded patients, sorted by creation date.
    - Include details like name, age, and onboarding date for quick reference.

Retell AI Caller Integration

1. **Retell AI SDK Setup**:
    - Install the SDK:

      npm install @retellai/sdk
    - Initialize the SDK in the backend:
        - Configure the Retell AI API key and settings during initialization.
        - Set up environment variables for sensitive credentials.
2. **API Endpoints**:
    - **POST /api/ retellai**:
        - Initiates an AI-powered call for patient-related queries, such as appointment reminders or follow-ups.
3. **AI Caller Interaction on Dashboard**:
    - **Chatbox Integration**:
        - Add a text-based AI chatbox to the dashboard for answering common clinic-related queries like working hours, contact details, or doctor availability.
    - **Voice Assistant**:
        - Provide a voice-based interface where users can speak their questions.
        - Use Retell AI's voice recognition capabilities to process and respond.

Benefits of Retell AI:

- Helps in resolving patient and clinic queries efficiently.
- Supports both basic informational queries (e.g., working hours, location) and advanced patient-specific interactions (e.g., appointment scheduling).

Future Enhancements:

- Extend AI capabilities to provide personalized medical advice based on patient history.

- Use AI insights to improve patient engagement and clinic workflows.

4. Stripe Payment Integration

Objective: Implement a secure payment system for doctors to pay a $100 platform fee.

Setup

1. **Install Stripe**: Add Stripe to the project and configure API keys in environment variables.
2. **Environment Configuration**: Store sensitive Stripe keys securely to ensure secure payment processing.

Payment Flow

Frontend

1. Use Stripe Elements to create a user-friendly and secure payment form to collect payment details.
2. Include form validation to ensure accurate and complete input.
3. Upon successful payment, communicate with the backend to finalize the process and store payment data.

Backend

1. Create an endpoint to handle payment processing securely using Stripe's paymentIntents API.
2. Validate payment information, process transactions, and return a response to the frontend.
3. Save payment details in the database, including payment date, amount, and status.

Payment Confirmation

1. Display a payment confirmation screen summarizing the transaction details for the user.
2. Send a confirmation email to the user upon successful payment.

Key Features

- **Security**: Use Stripe's PCI-compliant infrastructure for safe transactions.
- **Transparency**: Maintain clear and reliable payment records in the database.
- **User Experience**: Provide smooth payment interactions with responsive forms and detailed feedback.

5. UI/UX Design

Objective: Create a user-friendly and intuitive interface for the patient onboarding form and the dashboard, ensuring a smooth experience for both patients and doctors.

Patient Onboarding Form

1. **Clean Layout**:
    - Use **Tailwind CSS** to create a responsive and modern form layout.
    - Ensure the form elements are well-spaced, easy to navigate, and mobile-friendly.
    - Group related fields together (e.g., personal info, medical history) for better readability.
2. **Input Validation**:
    - Implement inline error messages that appear when a user submits invalid or incomplete fields.
    - Display helpful, concise error messages next to the relevant input fields.
    - Use real-time validation where possible, highlighting errors as the user types.
3. **Secure Data Entry**:
    - Use secure input fields (e.g., password fields for sensitive information).

Dashboard

1. **Patient Information**:
    - **Analytics**: Display patients' data in an organized, easy-to-read format.
    - Show key patient statistics such as the total number of patients, the most common medical conditions, and recent onboarding.
    - Use charts or tables to display analytics with filtering and sorting options.
2. **CRUD Operations**:
    - Allow doctors or admins to **create, read, update, and delete** patient records.
    - Each patient's record should be displayed with options for editing or deleting.
    - Provide confirmation modals or notifications when actions like deleting or updating patient records occur.
3. **AI Chatbot**:
    - **AI Caller Integration**: Add an interactive chatbot interface where users can ask questions about the clinic, appointment scheduling, and more.
    - Use the **Retell AI Caller** for seamless conversation, allowing users to interact with the bot through text or voice.
    - Keep the chat interface simple with clear call-to-action buttons, and display previous interactions for user convenience.

General Design Considerations

- **Consistency**: Maintain a consistent design style across the entire application (e.g., button styles, font choices, and spacing).
- **Responsive Design**: Ensure the application adapts seamlessly across different screen sizes using **Tailwind CSS** breakpoints.
- **User Feedback**: Provide real-time feedback on user actions (e.g., success or error messages after submitting forms or saving data).

Project Folder Structure

```
project-root/
├── client/
│   ├── public/
│   │   ├── index.html
│   │   └── ... (other static files like favicon, manifest, etc.)
│   ├── src/
│   │   ├── api/                          # API integration files
│   │   │   ├── authApi.jsx
│   │   │   ├── doctorApi.jsx
│   │   │   ├── retellAi.jsx
│   │   │   ├── stripeApi.jsx
│   │   │   ├── userApi.jsx
│   │   │   └── index.jsx
│   │   ├── assets/                       # Images, logos, fonts, etc.
│   │   │   └── ... (images, logos, fonts)
│   │   ├── conf/
│   │   │   ├── conf.jsx
│   │   │   ├── firebase-conf.jsx
│   │   │   ├── retellAi-conf.jsx
│   │   │   └── stripe-conf.jsx
│   │   ├── data/                         # Data for website
│   │   │   ├── MainData.jsx
│   │   │   ├── navBarData.jsx
│   │   │   ├── patientFormData.jsx
│   │   │   ├── PaymentCancelData.jsx
│   │   │   ├── PaymentSuccessData.jsx
│   │   │   └── PremiumData.jsx
│   │   ├── firebase/
│   │   │   ├── firebase-config.jsx
│   │   ├── layout/
│   │   │   ├── index.jsx
│   │   ├── components/                   # UI components
│   │   │   ├── Footer/
│   │   │   │   └── Footer.jsx
│   │   │   ├── Header/
│   │   │   │   ├── Header.jsx
│   │   │   │   ├── LogoutBtn.jsx
│   │   │   │   └── Navbar.jsx
│   │   │   ├── Home/
```

```
│   │   │   │   ├── ActionOption.jsx
│   │   │   │   ├── Carousel.jsx
│   │   │   │   └── Motive.jsx
│   │   │   ├── PatientsRecords/
│   │   │   │   ├── PatientDetails.jsx
│   │   │   │   ├── PatientForm.jsx
│   │   │   │   ├── PatientItem.jsx
│   │   │   │   └── PatientList.jsx
│   │   │   ├── Profile/
│   │   │   │   ├── ProfileDetails.jsx
│   │   │   │   ├── ProfileForm.jsx
│   │   │   │   ├── ProfilePage.jsx
│   │   │   │   └── AuthLayout.jsx
│   │   ├── pages/                          # Pages of website
│   │   │   ├── Home.jsx
│   │   │   ├── Login.jsx
│   │   │   ├── PatientsRecords.jsx
│   │   │   ├── Payment.jsx
│   │   │   └── UserProfile.jsx
│   │   ├── retellAi/
│   │   │   ├── RetellAi.jsx
│   │   │   └── RetellAIConnect.jsx
│   │   ├── routes/                         # Routes
│   │   │   └── index.jsx
│   │   ├── store/
│   │   │   ├── authSlice.js
│   │   │   └── store.js
│   │   ├── stripe/
│   │   │   ├── PaymentCancel.jsx
│   │   │   ├── PaymentForm.jsx
│   │   │   └── PaymentSuccess.jsx
│   │   ├── utils/
│   │   │   ├── authUtils.jsx
│   │   │   └── handleApiError.jsx
│   │   ├── App.jsx
│   │   ├── index.css
│   │   └── main.jsx
│   ├── .env
│   ├── .gitignore
│   ├── package.json
│   └── README.md
```

Key Points:

- **client/**: Contains all the client-side code for your React application.
- **public/**: Stores static files accessible by the browser.
- **src/**: Contains the source code for your React application.
- **api/**: Houses files for interacting with various APIs.
- **assets/**: Stores images, logos, fonts, and other visual assets.
- **auth/**: Contains authentication-related files and configurations.
- **Data/**: Contains static data used by the application.
- **FireBase/**: Contains components and configuration related to Firebase.
- **Layout/**: Contains components for the main application layout.
- **components/**: Houses reusable React components organized by functionality.
- **pages/**: Contains components for different pages within the application.
- **RetellAi/**: Contains components or logic specific to the RetellAi platform.
- **routes/**: Contains routing configuration for the application.
- **store/**: Contains Redux store configuration and slices.
- **Stripe/**: Contains components related to Stripe payment integration.
- **utils/**: Contains utility functions and helper components.
- **App.jsx**: The main application component.
- **index.css**: Global CSS styles.
- **main.jsx**: The entry point for the application.
- **.env**: Stores environment variables.
- **.gitignore**: Specifies files to be ignored by Git.
- **package.json**: Lists project dependencies and scripts.
- **README.md**: Provides project description and instructions.

```
├── server/
│   ├── conf/
│   │   ├── conf.js
│   │   ├── firebase-admin-config.js
│   │   ├── retellAi-conf.js
│   ├── conf/
│   │   ├── index.js
│   ├── auth/
│   │   ├── index.js
│   ├── controllers/
│   │   ├── patient.controller.js
│   │   ├── payment.controller.js
│   │   ├── retellai.controller.js
│   │   └── subscription.controller.js
│   │   └── user.controller.js
│   ├── models/
│   │   ├── doctor.models.js
│   │   ├── patient.models.js
│   │   └── user.models.js
│   ├── routes/
│   │   ├── auth.routes.js
│   │   ├── index.js
│   │   ├── patient.routes.js
│   │   └── retellai.routes.js
│   │   ├── stripe.routes.js
│   │   ├── user.routes.js
│   ├── middlewares/
│   │   ├── auth.middleware.js
│   ├── utils/
│   │   ├── index.js
│   ├──index.js
│   ├── .env
│   ├── .gitignore
│   ├── package.json
│   └── README.md
```

Explanation:

- **server/**: The root directory of the server-side code.
- **node_modules/**: Contains dependencies installed using npm or yarn.

- **src/**: Contains the source code for the application.
    - **auth/**: Likely contains authentication-related logic.
    - **conf/**: Contains configuration files.
    - **controllers/**: Contains controller functions that handle business logic and interact with models.
    - **db/**: Contains database-related logic.
    - **middlewares/**: Contains middleware functions that can be used to modify requests and responses.
    - **models/**: Contains data models that define the structure of data.
    - **routes/**: Contains route definitions that map HTTP requests to specific controllers or handlers.