

Build a complete, functional and responsive ****Doctor Appointment Booking Web App****.

Use the temporary brand name ****MediBook****. Keep the brand name in a central configuration file or component so it can easily be changed later.

Important project goal

This is a beginner-friendly MVP that will be demonstrated in a YouTube tutorial.

The app should be:

- * Simple to understand
- * Visually professional
- * Fully functional
- * Mobile responsive
- * Connected to my own Supabase account
- * Ready to deploy on a Hostinger Node.js server
- * Easy to explain in a video

Do not make the project unnecessarily complex.

1. Required technology stack

Use the following stack:

- * React
- * TypeScript
- * Vite
- * Tailwind CSS
- * React Router
- * Supabase JavaScript SDK
- * React Hook Form
- * Zod validation
- * Lucide React icons
- * Node.js
- * Express only for serving the production build on Hostinger

Do not use:

- * Replit Database
- * Firebase
- * MongoDB
- * Prisma
- * PostgreSQL hosted directly on Replit
- * Replit authentication
- * Replit deployment-specific databases
- * Hard-coded authentication
- * Mock authentication

Supabase must handle:

- * PostgreSQL database
- * User authentication
- * User sessions
- * Authorization
- * Row Level Security
- * Doctors
- * Availability slots

- * Appointments
- * Patient profiles

The production Node.js server should only serve the compiled React application. The frontend can communicate directly with Supabase using the Supabase anon key and properly configured Row Level Security policies.

2. Environment variables

Create an `.env.example` file containing:

```
``env
VITE_SUPABASE_URL=
VITE_SUPABASE_ANON_KEY=
``
```

Use these variables to initialize the Supabase client.

Do not place real keys directly in the source code.

Do not use or expose the Supabase service role key in the frontend.

When the required variables are missing, show a friendly setup screen explaining that Supabase environment variables need to be added. Do not allow the application to crash.

3. User roles

Keep the role system simple.

There should be two roles:

1. ``patient``
2. ``admin``

Patient

A patient can:

- * Create an account
- * Log in
- * Browse doctors
- * View doctor profiles
- * View available appointment slots
- * Book an appointment
- * View their own appointments
- * Cancel their own upcoming appointment
- * Update their own basic profile
- * Log out

Admin

The admin represents the clinic or doctor management team.

An admin can:

- * Log in through the same login page

- * Access a protected admin dashboard
- * Add doctors
- * Edit doctors
- * Activate or deactivate doctors
- * Manage doctor availability
- * Generate appointment slots
- * View all appointments
- * Confirm an appointment
- * Reschedule an appointment
- * Cancel an appointment
- * Mark an appointment as completed
- * View basic patient booking information

Doctors do not need individual login accounts in this MVP. Doctors should be stored as public doctor records managed by the admin.

4. Authentication

Use Supabase email and password authentication.

Create:

- * Patient registration page
- * Login page
- * Forgot password page
- * Reset password page
- * Logout functionality
- * Session persistence
- * Protected patient routes
- * Protected admin routes

Registration fields:

- * Full name
- * Email address
- * Phone number
- * Password
- * Confirm password

All new registrations must receive the `patient` role by default.

Do not allow users to select the admin role during registration.

After registration:

- * Create the corresponding profile automatically
- * Show the correct message when email confirmation is required
- * Redirect authenticated patients to their dashboard
- * Redirect authenticated admins to the admin dashboard

Create a reusable authentication context or provider that handles:

- * Current user
- * Current session
- * User profile
- * Role
- * Loading state
- * Login

- * Signup
- * Logout

5. Database structure

Create a Supabase SQL migration file at:

```
```text
supabase/schema.sql
```
```

The file must create the complete database structure, indexes, constraints, triggers, functions and Row Level Security policies.

Profiles table

Create a `profiles` table with:

- * `id` UUID primary key referencing `auth.users`
- * `full\_name` text
- * `phone` text
- * `role` text with allowed values `patient` and `admin`
- * `created\_at`
- * `updated\_at`

Automatically create a profile whenever a new Supabase Auth user registers.

The default role must always be `patient`.

Doctors table

Create a `doctors` table with:

- * `id`
- * `name`
- * `specialization`
- * `qualification`
- * `experience\_years`
- * `bio`
- * `clinic\_address`
- * `consultation\_fee`
- * `avatar\_url`
- * `rating`
- * `review\_count`
- * `is\_active`
- * `created\_at`
- * `updated\_at`

Availability slots table

Create an `availability\_slots` table with:

- * `id`
- * `doctor\_id`
- * `slot\_date`
- * `start\_time`
- * `end\_time`
- * `is\_available`

* `created\_at`

Prevent duplicate slots for the same doctor, date and start time.

Appointments table

Create an `appointments` table with:

* `id`
* `booking\_reference`
* `patient\_id`
* `doctor\_id`
* `slot\_id`
* `patient\_name`
* `patient\_phone`
* `reason\_for\_visit`
* `status`
* `admin\_notes`
* `created\_at`
* `updated\_at`

Allowed appointment statuses:

* `pending`
* `confirmed`
* `rescheduled`
* `cancelled`
* `completed`

A single slot must never be assigned to more than one active appointment.

The booking reference should be a short, readable value such as:

```
"""text
MED-8F2K91
"""
```

Do not store prescriptions, medical reports, diagnoses or detailed medical histories in this MVP.

6. Secure booking logic

Create a secure Supabase PostgreSQL function or RPC for booking appointments.

The booking operation must:

1. Confirm that the user is authenticated.
2. Confirm that the slot exists.
3. Confirm that the slot is available.
4. Prevent two patients from booking the same slot.
5. Create the appointment.
6. Mark the slot as unavailable.
7. Return the newly created appointment and booking reference.

The operation should be atomic so that double bookings cannot occur.

Also create secure functions or equivalent protected logic for:

* Patient cancelling their own appointment

- * Admin rescheduling an appointment
- * Admin cancelling an appointment
- * Admin confirming an appointment

When an appointment is cancelled, make the slot available again when appropriate.

When an appointment is rescheduled:

- * Release the previous slot
- * Reserve the new slot
- * Update the appointment
- * Prevent conflicts

7. Row Level Security

Enable Row Level Security on every public table.

Create secure policies so that:

Public visitors

- * Can view active doctors
- * Can view future available slots
- * Cannot view patient profiles
- * Cannot view appointments
- * Cannot modify database records

Authenticated patients

- * Can view and update only their own profile
- * Can view only their own appointments
- * Can create bookings for themselves through the secure booking function
- * Can cancel only their own eligible appointments
- * Cannot view another patient's appointment
- * Cannot create doctors
- * Cannot edit doctors
- * Cannot create or edit availability slots
- * Cannot promote themselves to admin

Admin users

- * Can view and manage all doctors
- * Can manage availability slots
- * Can view and update all appointments
- * Can view patient profile information required for bookings

Create a secure helper function such as ``is_admin()`` to check whether the authenticated user has the admin role.

Avoid recursive RLS policies.

Do not base authorization on editable user metadata.

8. Seed data

Create a separate file:

```
```text
supabase/seed.sql
```
```

Add at least four realistic demo doctors:

1. General Physician
2. Dermatologist
3. Pediatrician
4. Dentist

Include realistic:

- * Names
- * Qualifications
- * Experience
- * Short biographies
- * Consultation fees
- * Ratings
- * Clinic addresses
- * Placeholder profile images

Create sample available appointment slots for the coming seven days.

Use future dates dynamically when possible.

Do not create fake patient accounts or admin passwords in the seed file.

9. Public application pages

Home page

Create a clean medical-themed home page containing:

- * Responsive navigation bar
- * MediBook logo or simple medical icon
- * Hero section
- * Heading such as “Book Your Doctor Appointment Online”
- * Short supporting description
- * “Find a Doctor” button
- * “My Appointments” button
- * Search input
- * Specialization filter
- * Featured doctors
- * Simple three-step explanation:
 1. Choose a doctor
 2. Select a time
 3. Confirm appointment
- * Footer

Do not make the hero section excessively tall.

Doctors page

Display doctor cards containing:

- * Doctor image
- * Name
- * Specialization
- * Qualification
- * Experience
- * Rating
- * Consultation fee
- * "View Profile" button
- * "Book Appointment" button

Add:

- * Doctor search
- * Specialization filter
- * Loading state
- * Empty state
- * Error state

Doctor details page

Display:

- * Doctor image
- * Doctor name
- * Specialization
- * Qualifications
- * Experience
- * Biography
- * Clinic address
- * Consultation fee
- * Rating
- * Available appointment dates
- * Available appointment times
- * Book appointment button

Only show future available slots.

Booking page

Create a simple booking flow:

1. Doctor selected
2. Date selected
3. Time selected
4. Patient details entered
5. Booking confirmed

If the visitor is not logged in, redirect them to login and then return them to the selected doctor or booking flow.

Booking form fields:

- * Patient name
- * Phone number
- * Optional reason for visit

Pre-fill name and phone number from the logged-in patient profile.

Before confirming, show a booking summary containing:

- * Doctor
- * Specialization
- * Date
- * Time
- * Consultation fee
- * Patient name
- * Phone number

The user should click a final “Confirm Appointment” button.

After a successful booking, show:

- * Success icon
- * Booking reference
- * Doctor name
- * Appointment date
- * Appointment time
- * Status
- * “View My Appointments” button
- * “Back to Home” button

Do not implement online payment in this MVP.

10. Patient dashboard

Create a protected patient dashboard.

Dashboard sections:

- * Profile summary
- * Upcoming appointments
- * Previous appointments
- * Cancelled appointments

Each appointment card should show:

- * Booking reference
- * Doctor
- * Specialization
- * Date
- * Time
- * Status
- * Clinic address
- * Consultation fee
- * Cancel appointment button when eligible

Add clear status badges:

- * Pending
- * Confirmed
- * Rescheduled
- * Cancelled
- * Completed

Ask for confirmation before cancelling an appointment.

Patients must never see appointments belonging to other users.

11. Admin dashboard

Create a protected admin dashboard with a responsive sidebar.

Sidebar items:

- * Overview
- * Doctors
- * Availability
- * Appointments
- * Patients
- * Settings or Profile
- * Logout

Admin overview

Show summary cards:

- * Total doctors
- * Today's appointments
- * Pending appointments
- * Confirmed appointments
- * Completed appointments

Also show a simple recent appointments table.

Do not add complex analytics or charts.

Doctor management

Allow admin to:

- * Add doctor
- * Edit doctor
- * Activate doctor
- * Deactivate doctor
- * View doctor
- * Delete doctor only when safe

Use a clean form with proper validation.

Availability management

Allow admin to:

- * Select a doctor
- * Select a date
- * Set start time
- * Set end time
- * Select interval such as 15, 20, 30 or 60 minutes
- * Generate slots automatically
- * View generated slots
- * Delete an unbooked slot
- * Mark a slot available or unavailable

Prevent duplicate slots.

Appointment management

Show appointments in a table or responsive cards.

Include:

- * Booking reference
- * Patient
- * Phone
- * Doctor
- * Date
- * Time
- * Status
- * Actions

Add filters for:

- * Date
- * Doctor
- * Status
- * Search by patient or booking reference

Admin actions:

- * Confirm
- * Reschedule
- * Cancel
- * Mark completed
- * View details

For rescheduling, open a modal that displays only available future slots for the selected doctor.

Ask for confirmation before destructive actions.

Patient list

Show only basic details needed for appointment management:

- * Name
- * Email
- * Phone
- * Number of appointments
- * Registration date

Do not create medical records or medical history functionality.

12. User interface design

Use a professional medical-tech design.

Color palette:

- * White background
- * Dark navy text
- * Medical blue as the primary color
- * Teal as the secondary accent
- * Light blue or light gray section backgrounds
- * Green for success
- * Amber for pending

- * Red for cancelled or destructive actions

Design requirements:

- * Rounded cards
- * Soft shadows
- * Consistent spacing
- * Clear form labels
- * Professional typography
- * Visible focus states
- * Mobile responsive layout
- * Accessible color contrast
- * Simple icons
- * No unnecessary animations
- * No excessive gradients
- * No glassmorphism
- * No giant headings
- * No cluttered dashboard

Use toast notifications for:

- * Successful booking
- * Booking failure
- * Profile updated
- * Appointment cancelled
- * Doctor saved
- * Slots generated
- * Authentication errors

Create reusable components for:

- * Navbar
- * Footer
- * Doctor card
- * Appointment card
- * Status badge
- * Loading spinner or skeleton
- * Empty state
- * Error message
- * Confirmation dialog
- * Form input
- * Modal
- * Protected route

13. Validation and error handling

Use Zod and React Hook Form for forms.

Validate:

- * Valid email
- * Required full name
- * Valid phone number
- * Password length
- * Password confirmation
- * Required doctor details
- * Positive consultation fee
- * Valid slot start and end times

* Required appointment date and time

Handle:

- * Supabase connection errors
- * Authentication errors
- * Expired sessions
- * Missing user profiles
- * No available slots
- * Slot already booked
- * Duplicate slots
- * Unauthorized access
- * Network errors
- * Invalid URLs
- * Missing environment variables

Never display raw database errors directly to normal users.

Log useful technical errors to the browser console during development.

14. Routing

Use routes similar to:

```
``text
/
/doctors
/doctors/:doctorId
/login
/register
/forgot-password
/reset-password
/book/:doctorId
/booking-success/:appointmentId
/dashboard
/dashboard/profile
/dashboard/appointments
/admin
/admin/doctors
/admin/availability
/admin/appointments
/admin/patients
``
```

Create a proper 404 page.

Protect all dashboard and admin routes.

Redirect users according to their authentication status and role.

15. Production and Hostinger deployment

The project must be deployable as a Node.js application on Hostinger.

Create:

```
```text
server.mjs
```
```

The production server must:

- * Use Express
- * Serve the Vite `dist` folder
- * Support React Router SPA fallback
- * Listen on `process.env.PORT || 3000`
- * Bind to `0.0.0.0`
- * Include basic security headers
- * Include compression when appropriate
- * Include a health endpoint at `/health`

The health endpoint should return JSON similar to:

```
```json
{
 "status": "ok",
 "app": "MediBook"
}
```
```

Create these package scripts:

```
```json
{
 "dev": "vite --host 0.0.0.0",
 "build": "tsc -b && vite build",
 "start": "node server.mjs"
}
```
```

Make sure:

- * `npm run dev` works in Replit
- * `npm run build` completes successfully
- * `npm start` serves the production application
- * Directly opening a React Router URL does not return a 404
- * The application uses the port supplied by Hostinger

Do not make the production app dependent on Replit.

16. Documentation

Create a detailed `README.md`.

Include:

Local/Replit setup

1. Install dependencies.
2. Create a Supabase project.
3. Run `supabase/schema.sql` in the Supabase SQL Editor.
4. Run `supabase/seed.sql`.
5. Add Supabase URL and anon key to Replit Secrets.
6. Start the app.

7. Register a normal account.

Creating the first admin

Explain how to:

1. Register a normal account.
2. Locate the user in Supabase Authentication.
3. Update that user's profile role to `admin` through a safe SQL command.
4. Log out and log in again.
5. Access `/admin`.

Include the exact SQL command with an email placeholder, such as:

```
```sql
update public.profiles
set role = 'admin'
where id = (
 select id
 from auth.users
 where email = 'YOUR_ADMIN_EMAIL'
);
```
```

Supabase authentication configuration

Explain:

- * Site URL
- * Redirect URLs
- * Email confirmation behaviour
- * Password reset redirect URL

Hostinger deployment

Explain:

- * Push the code to GitHub or upload the project
- * Select Node.js web app deployment
- * Install dependencies
- * Add environment variables before building
- * Build command
- * Start command
- * Required Node.js version
- * Domain connection
- * Supabase Site URL update
- * Supabase redirect URL update
- * How to test `/health`
- * How to troubleshoot blank pages or route 404 errors

Recommended Hostinger settings:

```
```text
Build command: npm install && npm run build
Start command: npm start
Node.js version: 20 or newer
```
```

Explain that these variables must be added on Hostinger:

```
```env
VITE_SUPABASE_URL=
VITE_SUPABASE_ANON_KEY=
```
```

Mention that Vite environment variables are included during the build, so the variables must exist before running the production build.

17. Project structure

Organize the code clearly.

Use a structure similar to:

```
```text
src/
 components/
 pages/
 layouts/
 hooks/
 contexts/
 lib/
 services/
 types/
 utils/
 config/
supabase/
 schema.sql
 seed.sql
public/
server.mjs
.env.example
README.md
```
```

Keep components reasonably small.

Create shared TypeScript types for:

- * Profile
- * Doctor
- * Availability slot
- * Appointment
- * Appointment status

Do not put the complete application inside one large component.

18. MVP limitations

Do not add these features:

- * Online payments
- * Video consultations
- * Live chat
- * Prescription generation
- * Medical records

- * Uploading medical reports
- * Insurance
- * Multiple clinics
- * Multiple admin levels
- * SMS integration
- * WhatsApp integration
- * Complex email automation
- * AI diagnosis
- * Maps integration
- * Reviews submission
- * Advanced analytics

The application should only focus on:

- * Authentication
- * Doctor listing
- * Doctor profiles
- * Availability
- * Appointment booking
- * Patient appointments
- * Admin appointment management

19. Testing requirements

After building the project, test the following flows:

Public flow

- * Home page loads
- * Doctor listing loads
- * Search and specialization filters work
- * Doctor profile loads
- * Available slots display correctly

Patient flow

- * Patient can register
- * Patient can log in
- * Patient profile is created
- * Patient can book an available slot
- * Booking confirmation appears
- * Appointment appears in patient dashboard
- * Patient cannot book the same unavailable slot
- * Patient can cancel an eligible appointment
- * Patient cannot access admin routes
- * Patient cannot see another patient's appointments

Admin flow

- * Admin can access admin dashboard
- * Admin can add and edit doctors
- * Admin can generate slots
- * Admin can view appointments
- * Admin can confirm appointments
- * Admin can reschedule appointments
- * Admin can cancel appointments
- * Admin can mark appointments completed

Production flow

- * `npm run build` succeeds
- * `npm start` succeeds
- * `/health` responds correctly
- * React Router refresh works
- * No real secrets are committed
- * No service role key appears in client code

Fix all build errors, TypeScript errors and obvious console errors before considering the project complete.

20. Final delivery

Before coding, create a short implementation plan and database plan.

Then build the complete application without repeatedly asking me questions. Use sensible defaults wherever a small design decision is required.

At completion, provide:

1. Summary of features built
2. List of important files
3. Environment variables required
4. Supabase setup instructions
5. Admin creation instructions
6. Replit testing instructions
7. Hostinger deployment instructions
8. Any remaining limitations

Do not claim that a feature is complete unless it has actually been implemented and tested.

Production file prompt:

Now, I want to host this entire application on my hostinger node.js server. Hostinger understands the .zip file format. So, make sure to organize the files and folders accordingly and prepare the production file.