Technical Documentation: Carpet Cleaning Services Platform

1. Architecture Overview

This application is a full-stack On-Demand Service Marketplace designed to connect service providers (carpet cleaners) with consumers.

The architecture follows a **Serverless, Client-Side Rendered (CSR)** model:

- **Frontend:** A Single Page Application (SPA) built with React and TypeScript, bundled by Vite. It handles routing, UI rendering, and state management.
- **Backend & Database:** Powered by **Convex**, a backend-as-a-service platform. Convex handles the database (real-time document storage), API functions, file storage, and authentication logic.
- **Payments:** Integrated with **Stripe** for handling booking payments and provider payouts.
- **Hosting:** The application builds to static files (`` `html` ``, `` `css` ``, `` `js` ``) making it deployable on any standard shared hosting environment (Apache/Nginx).

2. Tech Stack

Frontend Core

- **Framework:** React (v18+)
- **Language:** TypeScript
- **Bundler:** Vite
- **Routing:** React Router DOM (inferred from multi-page structure)
- **State Management:** React Context + Convex React Hooks

Backend & Data

- **BaaS Provider:** Convex
- **Database:** Convex (Relational Document Store)
- **File Storage:** Convex File Storage (for profile images/service photos)
- **API Interface:** Convex Remote Procedure Calls (Query/Mutation)

External Services

- **Payments:** Stripe API
- **Authentication:** Convex Auth (Custom logic implemented in `convex/auth.config.ts`)

3. Project Structure

```
├── components/          # Reusable UI components (Buttons, Inputs, Navbar)
├── convex/              # Backend Logic (Database Schema & API Functions)
│   ├── _generated/      # Auto-generated type definitions from Convex
│   ├── bookings.ts      # Booking logic (CRUD)
│   ├── schema.ts        # Database schema definition
│   ├── stripe.ts        # Stripe payment webhooks and intent logic
│   └── ...              # Other feature-specific backend logic
├── lib/                 # Client-side utilities and contexts
│   ├── authContext.tsx  # Authentication state provider
│   └── constants.ts     # Global configuration constants
├── pages/               # Application Views (Screens)
│   ├── consumer/        # Consumer-specific dashboards
│   ├── provider/        # Service Provider dashboards & settings
│   ├── static/          # Static content (About, Terms, Privacy)
│   ├── AdminDashboard.tsx
│   └── ...
├── public/              # Static assets and server config (.htaccess)
├── App.tsx              # Main Application Component & Routing Setup
└── vite.config.ts       # Vite bundler configuration
```

4. Database Schema

The database is defined in `convex/schema.ts`. Below is the logical data model implemented in the system:

`users`

Stores all registered entities (Consumers, Providers, and Admins).

- `_id`: System ID
- `name`: String
- `email`: String
- `role`: String ('consumer' | 'provider' | 'admin')
- `profileImage`: String (URL)

- `profileImage`: String (URL)

`providers`

Extension of user data specific to service providers.

- `userId`: Reference to `users`
- `businessName`: String
- `description`: String
- `serviceArea`: String
- `pricing`: Object (Rate per room/sqft)
- `stripeAccountId`: String (for payouts)

`bookings`

Transactional records between consumers and providers.

- `consumerId`: Reference to `users`
- `providerId`: Reference to `users`
- `serviceDate`: Timestamp
- `status`: String ('pending', 'confirmed', 'completed', 'cancelled')
- `totalAmount`: Number
- `paymentStatus`: String

`reviews`

Feedback left by consumers.

- `bookingId`: Reference to `bookings`
- `providerId`: Reference to `users`
- `rating`: Number (1-5)
- `comment`: String

`content`

Dynamic CMS content for the landing page.

- `section`: String (e.g., 'hero', 'features')
- `data`: JSON Object

5. Setup & Installation

Prerequisites

- Node.js (v16 or higher)
- npm or yarn

Local Development Steps

1. **Clone the repository:**

   ```
   git clone <repository-url>
   cd FLIPPA-CarpetCleaningServices
   ```

2. **Install dependencies:**

   ```
   npm install
   ```

3. **Initialize Convex:** This connects your local code to a hosted Convex backend instance.

   ```
   npx convex dev
   ```

   Follow the prompts to log in and create a new project.

4. **Configure Environment Variables:** Create a `.env.local` file in the root directory. You need to populate the Stripe keys (refer to `STRIPE_SETUP.md` for details):

   ```
   VITE_CONVEX_URL=https://your-convex-url.convex.cloud
   VITE_STRIPE_PUBLISHABLE_KEY=pk_test_...
   ```

5. **Seed the Database (Optional):** If the project includes a seed script:

   ```
   npx convex run seed
   ```

6. **Start the Frontend:**

   ```
   npm run dev
   ```

   The app will be available at `http://localhost:5173`.

6. Admin Access

6. Admin Access

The application comes with a pre-configured administrator account for managing the platform, users, and content.

- **Username:** `codeblix`
- **Password:** `codeblix`

7. Deployment Guide

This application is designed to be hosted on standard **Shared Hosting** (cPanel, Hostinger, GoDaddy, etc.). You do **not** need Vercel or Netlify.

Build Instructions

1. **Compile the Production Build:** Run the build command to generate static files.

   ```
   npm run build
   ```

2. **Locate Output:** This will create a `dist/` folder in your project root. This folder contains:
   - `index.html`
   - `assets/` (bundled JS and CSS)
   - `.htaccess` (Required for routing)

Uploading to Shared Hosting

1. **Access your File Manager:** Log in to your hosting provider's cPanel or use an FTP client (like FileZilla).
2. **Navigate to Public Directory:** Go to `public_html` or the specific folder for your domain.
3. **Upload:** Upload **all files** from inside the `dist/` folder to your server directory.
4. **Verify Routing:** Ensure the `.htaccess` file (located in the `dist` folder and sourced from `public/.htaccess`) is uploaded. This file is critical; it directs all traffic to `index.html`, allowing the React Router to handle page navigation without 404 errors.

Backend Deployment

The backend logic resides on the Convex cloud platform. When you run `npx convex deploy`, your backend functions are pushed to production automatically. No server

`deploy`, your backend functions are pushed to production automatically. No server management is required for the backend.