HALO Luxury Car Marketplace -

Technical Documentation

Architecture Overview

HALO is a high-end, multi-tenant SaaS marketplace designed for independent luxury car rental agencies. The application relies on a modern, serverless architecture to ensure high performance, real-time data synchronization, and scalability.

High-Level Data Flow

1. **Client (Frontend):** A Single Page Application (SPA) built with React and Vite. It handles the UI, routing, and user interactions.
2. **Authentication:** User identity is managed by **Clerk**. Clerk tokens are passed to the backend to authenticate API requests via JWT.
3. **Backend & Database (Convex):** A serverless backend acting as both the database (reactive document store) and the API layer. It handles data persistence, file storage, and business logic.
4. **Payments (Stripe):** Handles marketplace transactions.
    - **Stripe Connect:** Onboards Providers (agencies) to receive payouts.
    - **Payment Intents:** Processes payments from Consumers, splitting funds between the Platform and the Provider.

Tech Stack

Frontend

- **Framework:** React v19 (via Vite)
- **Language:** TypeScript
- **Styling:** Tailwind CSS (with custom font families: Montserrat & Michroma)
- **Routing:** React Router DOM v7
- **State Management:** React Context API (Auth, Music, Platform settings)
- **Motion/Effects:** CSS Animations & HTML5 Canvas (Audio Visualizer)

Backend & Infrastructure

- **Backend-as-a-Service:** Convex
- **Database:** Convex (Real-time, document-oriented)

- **Database:** Convex (Real-time, document-oriented)
- **File Storage:** Convex Storage & Cloudinary (for specific assets)
- **Authentication:** Clerk
- **Payments:** Stripe API (Connect & Checkout)

Key Packages

- `convex`: Backend SDK
- `@clerk/clerk-react`: Auth integration
- `@stripe/react-stripe-js`: Payment UI components
- `@faker-js/faker`: For generating seed data

Project Structure

```
FLIPPA-Car/
├── components/           # Reusable UI components
│   ├── layouts/          # Layout wrappers (Admin, Provider)
│   └── ui/               # Base UI elements (Button, Input)
├── context/              # Global React Context providers
│   ├── AuthContext.tsx   # Unifies Clerk and Convex user data
│   ├── MusicContext.tsx  # Handles site audio/visualizer logic
│   └── PlatformContext.tsx # White-labeling settings
├── convex/               # Backend Logic
│   ├── schema.ts         # Database Schema definition
│   ├── users.ts          # User management functions
│   ├── bookings.ts       # Booking logic
│   └── ...               # Other feature-specific API modules
├── pages/                # Route Views
│   ├── admin/            # Super Admin panels
│   ├── consumer/         # User dashboard views
│   ├── provider/         # Agency dashboard views
│   ├── onboarding/       # Stripe Connect & Store setup flows
│   └── ...               # Public pages (Home, Vehicles, etc.)
├── App.tsx               # Main Router configuration
├── main.tsx              # App Entry point
└── index.css             # Global Tailwind styles
```

Database Schema

The database is schema-enforced via `convex/schema.ts`.

Tables

1. `users`
   - `tokenIdentifier`: Link to Clerk Identity.
   - `role`: `consumer`, `provider`, or `admin`.
   - `email`, `name`, `avatarStorageId`.
2. `providers`
   - `userId`: Foreign Key to `users`.
   - `stripeAccountId`: Connected Stripe ID for payouts.
   - `storeName`, `slug`, `location`, `subscriptionTier`.
   - `stripeConnected`: Boolean status.
3. `vehicles`
   - `providerId`: Foreign Key to `users` (Owner).
   - `make`, `model`, `year`, `pricePerDay`.
   - `specs`: Object (engine, acceleration, power).
   - `isListed`: Visibility toggle.
4. `bookings`
   - `vehicleId`: Linked Vehicle.
   - `consumerId`: Renter.
   - `providerId`: Agency.
   - `status`: `pending`, `confirmed`, `completed`, `cancelled`.
   - `paymentIntentId`: Stripe Transaction ID.
5. `brands`
   - Lookup table for car brands and image assets used in the landing page.
6. `platform_settings`
   - Stores white-label configuration (Platform Name, Logo, Favicon).

Setup & Installation

Setup & Installation

Prerequisites

- Node.js (v18 or higher)
- npm

1. Installation

Install project dependencies:

```
npm install
```

2. Environment Variables

Create a `.env.local` file in the root directory. You will need keys from **Convex**, **Clerk**, and **Stripe**.

```
# Convex (Automatically set by npx convex dev)
VITE_CONVEX_URL="https://your-project.convex.cloud"

# Clerk Auth
VITE_CLERK_PUBLISHABLE_KEY="pk_test_..."

# Stripe (Public Key)
VITE_STRIPE_PUBLISHABLE_KEY="pk_test_..."
```

3. Backend Configuration

1. Initialize Convex:

   ```
   npx convex dev
   ```

2. **Clerk Integration:**
   - Go to the Convex Dashboard > Settings > Environment Variables.
   - Add `CLERK_SIGNING_KEY` (From Clerk JWT Templates).
   - Add `STRIPE_SECRET_KEY` (Stripe Secret Key).
   - Add `HOST_URL` (e.g., `http://localhost:5173`).

4. Seeding Data

To populate the database with demo agencies, cars, and testimonials:

```
npx convex run seed:seedData
```

5. Running the App

```
npm run dev
```

Admin Access

To access the platform configuration panel:

- URL: `/admin/login`
- Username: `codeblix`
- Password: `codeblix`

Deployment Guide

Since this application utilizes a Client-Side Rendering (CSR) approach with a separate serverless backend, deployment is straightforward.

1. Deploy Backend

Push your Convex functions and schema to production:

```
npx convex deploy
```