CrankOver - Technical Documentation

1. Architecture Overview

CrankOver is a modern Vertical SaaS marketplace tailored for classic automobile parts. It utilizes a **serverless, reactive architecture** to ensure real-time data synchronization and high performance.

High-Level Data Flow

1. **Client (SPA):** The application is built as a Single Page Application using React and Vite. It communicates directly with the Convex backend via WebSocket for real-time updates.
2. **Backend (BaaS):** Convex acts as the backend, database, and file storage provider. It handles API functions (Queries, Mutations, Actions) and HTTP webhooks.
3. **Authentication:** Clerk handles user identity. Upon login, user data is synchronized to the Convex `users` table via a webhook or client-side sync hook.
4. **Payments:** Stripe Connect (Express) is used.
    - **Buyers** pay via Stripe Checkout.
    - **Providers** (Sellers) receive payouts via Stripe Connect transfers.
    - **Platform** retains a transaction fee (configurable).
5. **AI Integration:** Google Gemini is integrated via Convex Actions to generate product descriptions automatically.

2. Tech Stack

Frontend

- **Framework:** React 19
- **Build Tool:** Vite
- **Language:** TypeScript
- **Styling:** TailwindCSS (Inline configuration)
- **Animations:** Framer Motion

- **Icons:** Lucide React
- **Routing:** React Router DOM v7

Backend & Services

- **Database & Backend:** Convex (Reactive Database)
- **Authentication:** Clerk (`@clerk/clerk-react`)
- **Payments:** Stripe SDK (Node.js)
- **Image Processing:** `browser-image-compression`
- **AI:** Google GenAI SDK (Gemini)
- **Mock Data:** FakerJS

3. Project Structure

```
FLIPPA-CrankOver/
├── components/          # Reusable UI components
│   ├── Header/          # Navigation and Topbar logic
│   └── ...              # Widgets (ProductCard, Hero, etc.)
├── context/             # Global State (NavigationContext)
├── convex/              # Backend Logic (Serverless Functions)
│   ├── _generated/      # Convex auto-generated types
│   ├── schema.ts        # Database Schema definition
│   ├── stripe.ts        # Payment processing logic
│   ├── users.ts         # User management logic
│   ├── products.ts      # Product CRUD & AI generation
│   └── ...              # Other feature-specific modules
├── pages/               # Route Views (Home, Shop, Dashboard, Admin)
├── public/              # Static assets and .htaccess for routing
├── scripts/             # PowerShell automation scripts (Stripe setup)
├── App.tsx              # Main Application Entry & Routing Logic
└── index.tsx            # React DOM mounting & Provider wrapping
```

4. Database Schema

The database is managed by Convex. The schema is defined in `convex/schema.ts`.

Table Name	Description	Key Relationships
`users`		`clerkId` `email`

`providers`	`users` `userId`
`products`	`providers` `providerId`
`orders`	`consumerId` `providerId`
`platformSettings`	
`reviews`	`products` `users`
`posts`	`users`
`comments`	`posts`
`blogs`	
`testimonials`	

5. Setup & Installation

Prerequisites

1. **Node.js** (v18 or higher)
2. **Convex Account** (Create at convex.dev)
3. **Clerk Account** (Create at clerk.com)
4. **Stripe Account** (with Connect enabled)
5. **Google Gemini API Key** (Optional, for AI features)

Local Development Steps

1. **Install Dependencies:**

```
npm install
```

2. **Environment Variables:** Create a `.env.local` file in the root directory:

```
VITE_CLERK_PUBLISHABLE_KEY=pk_test_...
VITE_CONVEX_URL=https://...
# The following are set in the Convex Dashboard, NOT .env.local
# STRIPE_SECRET_KEY, STRIPE_WEBHOOK_SECRET, CLERK_JWT_ISSUER_DOMAIN, GEMINI_AP
```

3. **Initialize Convex:**

```
npx convex dev
```

   - This will prompt you to log in and select a project.
   - Go to the Convex Dashboard > Settings > Environment Variables and add:
     - `STRIPE_SECRET_KEY`
     - `STRIPE_API_KEY`
     - `GEMINI_API_KEY`
     - `CLERK_ISSUER_URL` (Found in Clerk > API Keys > Advanced > JWT Issuer)

4. **Configure Stripe:**

   - Run the provided script to sync keys (Windows/PowerShell):

```
.\scripts\stripe.ps1
```

   - Or manually configure the Webhook URL in Stripe to point to your Convex HTTP action: `https://<your-convex-url>.convex.site/stripe/webhook`.

5. **Seed Data:** Populate the database with mock data:

   - Go to Convex Dashboard > Functions > `seed:seedData`.
   - Click "Run Function".

6. **Run the App:**

```
npm run dev
```

6. Admin Credentials

To access the `/admin` route for platform management (CMS, Global Settings, User moderation), use the following internal credentials:

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

7. Deployment Guide

7. Deployment Guide

Since this is a React SPA using Convex for the backend, you do **not** need specialized Node.js hosting (like Vercel or Netlify), though they work fine. Standard Shared Hosting (Apache/Nginx) is sufficient.

Build the Application

Run the build command to generate static files:

```
npm run build
```

This creates a `dist/` folder containing optimized HTML, CSS, and JS.

Hosting on Shared Hosting (Apache)

1. **Upload:** Upload the contents of the `dist/` folder to your server's `public_html` or www root.
2. **Routing:** The project includes a `.htaccess` file in the `public/` folder. Ensure this file is uploaded to the root of your server. This handles rewrites so that deep links (e.g., `yourdomain.com/shop`) are routed back to `index.html` for React to handle.

Convex Backend Deployment

The backend is hosted by Convex cloud. To deploy your local functions to production:

```
npx convex deploy
```

- Ensure you set your Production Environment Variables in the Convex Dashboard (same keys as development: Stripe, Clerk, Gemini).

Stripe Production Configuration

1. In Stripe Dashboard, switch to **Live Mode**.
2. Update the **Redirect URL** in Connect Settings to your production domain: `https://yourdomain.com/dashboard`.
3. Update the **Webhook Endpoint** to your production Convex URL.