

FunRideBikes Technical Documentation

Architecture Overview

FunRideBikes is a modern, high-performance web application operating as a two-sided marketplace for bicycle rentals. It connects Bike Providers (owners) with Consumers (renters).

The architecture is built on a **Serverless SPA (Single Page Application)** model:

- Frontend:** Built with React and Vite, utilizing React Router for client-side routing.
- Backend & Database:** Powered by Convex, which provides a real-time database, serverless functions (queries, mutations, and actions), and background job scheduling.
- Authentication:** Managed by Clerk, securely synced to the Convex database via webhooks and client-side syncing mechanisms.
- Payments & Subscriptions:** Handled by Stripe. Stripe Connect is used for routing marketplace payments to Providers, and Stripe Checkout is used for platform SaaS subscriptions.
- AI Integration:** Utilizes Google Gemini to power a natural language "Magic Search" feature that translates user queries into structured database filters.

Tech Stack

Frontend

Framework: React 18

- **Framework:** React 18
- **Build Tool:** Vite
- **Routing:** React Router DOM v6
- **Styling:** Tailwind CSS
- **Animations:** Framer Motion
- **Date Management:** `date-fns`, `react-datepicker`

Backend & Data

- **Backend-as-a-Service:** Convex (Real-time DB, Serverless Functions, File Storage)
- **Language:** TypeScript (End-to-end type safety)

Third-Party Services

- **Authentication:** Clerk (`@clerk/clerk-react`)
- **Payments:** Stripe (`@stripe/stripe-js`, `@stripe/react-stripe-js`)
- **AI:** Google GenAI (`@google/genai`)
- **Mock Data Generation:** Faker.js (for database seeding)

Project Structure

```
FLIPPA-BikeRental/
├── convex/                                # Backend Logic & Database schema
│   ├── _generated/                       # Auto-generated Convex types and endpoints
│   ├── ai.ts                             # Gemini API integration for Magic Search
│   ├── bikes.ts                          # Bike CRUD operations
│   ├── bookings.ts                      # Booking logic, availability checks, Stripe
│   ├── http.ts                           # HTTP routing (Stripe Webhooks)
│   ├── providers.ts                     # Provider profile & SaaS subscription management
│   ├── schema.ts                         # Database schema definition
│   ├── seed.ts                           # Database seeding script
│   ├── stripe.ts                         # Stripe Connect & Checkout integrations
│   └── users.ts                           # User syncing and role management
├── public/                               # Static assets
└── http.config.ts                        # Apache configuration for SPA routing
```

```

├── .htaccess                # Apache configuration for SPA routing
├── scripts/                 # Utility scripts
│   └── stripe.ps1          # PowerShell script for automated Stripe
├── src/                     # Frontend React Application
│   ├── components/         # Reusable UI components (Hero, Buttons,
│   ├── context/            # Global React Contexts (Auth, Settings)
│   ├── lib/                # Helper functions and utilities
│   ├── pages/              # Route components
│   │   ├── consumer/       # Renter-facing pages (Bookings, Profile
│   │   ├── provider/       # Owner-facing dashboard
│   │   └── ...             # Public pages (Home, Search, Storefront
│   ├── App.tsx             # Application router and layout wrapper
│   ├── ConvexClientProvider.tsx # Initializes Clerk & Convex providers
│   ├── constants.ts        # Application static data and configurat
│   └── index.tsx           # Application entry point
├── package.json            # Project dependencies
└── vite.config.ts          # Vite bundler configuration

```

Database Schema

The database is housed in Convex. Below are the core collections and their primary responsibilities based on ``convex/schema.ts``:

``users``

Core user accounts synced from Clerk.

- **Fields:** ``tokenId`` (Clerk ID), ``email``, ``name``, ``role`` (guest | provider | consumer | admin), ``avatar``, ``stripeAccountId`` (Connect), ``stripeCustomerId`` (SaaS), ``subscriptionStatus``, ``joinedAt``.
- **Indexes:** ``by_token``, ``by_email``, ``by_stripe_customer``.

``providers``

Profiles for users who rent out bikes (Storefronts).

- **Fields:** ``userId``, ``storeName``, ``storeSlug``, ``description``, ``logo``,

`location , amenities , stripeConnected , balance ,
`subscriptionStatus`, `subscriptionPlan`.`

- **Indexes:** ``by_user`, `by_slug`, `by_stripe_customer`.`

``bikes``

Inventory items belonging to providers.

- **Fields:** ``providerId`, `name`, `category`, `priceHourly`, `priceDaily`,
`image`, `features`, `isHotDeal`.`
- **Indexes:** ``by_provider`.`

``bookings``

Transaction records between consumers and providers.

- **Fields:** ``bikeId`, `providerId`, `consumerId`, `startDate`, `endDate`,
`totalPrice`, `status` (pending | confirmed | completed | cancelled),
`stripePaymentIntentId`, `hasReview`.`
- **Indexes:** ``by_consumer`, `by_provider`, `by_payment_intent`.`

``reviews``

Ratings and feedback left by consumers after a booking.

- **Fields:** ``bookingId`, `providerId`, `consumerId`, `consumerName`, `rating`,
`comment`.`
- **Indexes:** ``by_provider`.`

``settings``

Global platform configurations.

- **Fields:** ``platformName`, `platformLogo`, `subscriptionPrice`,
`stripePriceId`, `stripePriceIdOneDollar`.`

Setup & Installation

1. Prerequisites

- Node.js (v18 or higher recommended)
- NPM or Yarn
- Accounts for: Convex, Clerk, Stripe, and Google Gemini.

2. Environment Configuration

Create a ``.env.local`` file in the root directory and populate it with your API keys:

```
# Clerk Authentication
VITE_CLERK_PUBLISHABLE_KEY=pk_test...

# Stripe Integration
VITE_STRIPE_PUBLISHABLE_KEY=pk_test...

# Gemini AI (For Magic Search)
GEMINI_API_KEY=AIzaSy...
```

3. Initialize Backend (Convex)

Open your terminal and run the following to install dependencies and initialize your database:

```
npm install
npx convex dev
```

Note: This command will prompt you to log into Convex and create a project. It will automatically add your ``.env.local`` to ``.env.local``.

4. Configure Convex Environment Variables

Go to your Convex Dashboard -> Settings -> Environment Variables, and add the following server-side secrets:

- ``.CLERK_ISSUER_URL``: The JWT issuer URL from your Clerk dashboard.

- ``STRIPE_SECRET_KEY``: Your Stripe secret key (``sk_test_...``).
- ``STRIPE_WEBHOOK_SECRET``: Your Stripe webhook signing secret (``whsec_...``).
- ``GEMINI_API_KEY``: Your Google GenAI key.

5. Seed Initial Data

While ``npm run dev`` is running in one terminal, open a new terminal and run the seeding script to populate the database with dummy users, providers, and bikes:

```
npx convex run seed:seedBikes
```

6. Start the Development Server

```
npm run dev
```

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

7. Admin Access

To access the admin control panel, navigate to ``/admin`` in your browser and use the following credentials:

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

Deployment Guide

The application is a standard React Single Page Application (SPA) paired with a serverless Convex backend. Because Convex hosts the backend entirely, **shared hosting is enough for the frontend, no Vercel required.**

1. Deploy the Backend

Deploy your Convex backend functions and schema to production using the CLI:

```
npx convex deploy
```

Make sure all environment variables (Stripe, Clerk, Gemini) are also added to your Production environment in the Convex Dashboard.

2. Build the Frontend

Compile the React application for production:

```
npm run build
```

This will generate a `dist/` directory containing your optimized static files.

3. Upload to Shared Hosting

1. Log in to your shared hosting provider (e.g., via cPanel, FTP, or File Manager).
2. Navigate to your `public_html` or designated domain folder.
3. Upload the entire contents of the `dist/` directory.

4. Routing Configuration (.htaccess)

Since this is a client-side routed SPA, all traffic needs to be directed to `index.html`. A `.htaccess` file is already included in the `public/` folder of this repository, which automatically carries over to your `dist/` folder during the build.

If it is missing, ensure the following `.htaccess` is present in your hosting root:

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.html$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
```

```
RewriteCond %{REQUEST_FILENAME} !-f  
RewriteCond %{REQUEST_FILENAME} !-d  
RewriteRule . /index.html [L]
```

Your application is now live and fully functional on your shared hosting environment.