Technical Documentation: FLIPPA-PayByDeal

Architecture Overview

FLIPPA-PayByDeal is a modern Single Page Application (SPA) designed to facilitate a digital marketplace or deal-matching platform. The application is entirely decoupled from the backend architecture, relying on a client-side rendering approach. State management and routing are handled on the client, ensuring fast, seamless transitions between views such as the catalog, user dashboards, and administrative panels.

The application leverages component-based architecture, promoting maximum reusability and isolated testing. Features like "SmartSearch" and "MagicMatch" are encapsulated in their respective components, allowing the UI to dynamically respond to user interactions without requiring full page reloads.

Tech Stack

Based on the repository configuration, the project utilizes the following technologies:

- **Core Framework:** React 18+
- **Language:** TypeScript
- **Build Tool / Bundler:** Vite (`vite.config.ts`)
- **Routing:** React Router DOM (inferred from the extensive `pages/` directory)
- **Styling:** Modern CSS / Tailwind CSS (`index.css` / UI components)
- **CI/CD:** GitHub Actions (`.github/workflows/main.yml`)
- **Web Server Configuration:** Apache (`public/.htaccess` for SPA

routing fallback)

Project Structure

The codebase follows a standard, scalable React project structure:

```
FLIPPA-PayByDeal/
├── .github/workflows/      # Automated CI/CD pipelines (GitHub Actions)
├── public/                 # Static assets copied directly to the build
│   ├── .htaccess           # Apache configuration for routing SPA URLs
│   └── assets/             # Images, fonts, and uncompiled static files
├── src/                    # Main application source code
│   ├── components/         # Reusable UI blocks (Hero, Catalog, SmartSe
│   ├── pages/              # Route-level components (Landing, Admin, Da
│   ├── App.tsx             # Root component and application router
│   ├── index.css           # Global stylesheets and utility layer impor
│   └── main.tsx            # Application entry point / React DOM attach
├── metadata.json           # Application metadata and configuration def
├── package.json            # Project dependencies and NPM scripts
├── tsconfig.json           # TypeScript compiler configuration
└── vite.config.ts          # Vite build and dev server configuration
```

Database Schema

While the current repository acts as the frontend client, the following relational database schema is modeled to support the application's domain based on the implemented pages (`Store`, `Dashboard`, `Admin`, `MagicMatch`, `Catalog`):

`users`

Column	Type	Description
`id`	UUID (PK)	Unique identifier for the user
`email`	VARCHAR	User's email address
`password_hash`	VARCHAR	Encrypted password

`role`	ENUM	`admin`, `buyer`, `seller`
`created_at`	TIMESTAMP	Account creation date

`deals` (Listings/Catalog)

Column	Type	Description
`id`	UUID (PK)	Unique identifier for the deal
`seller_id`	UUID (FK)	Reference to `users.id`
`title`	VARCHAR	Title of the deal/product
`description`	TEXT	Detailed description
`price`	DECIMAL	Price of the deal
`status`	ENUM	`active`, `pending`, `sold`

`transactions`

Column	Type	Description
`id`	UUID (PK)	Unique transaction ID
`deal_id`	UUID (FK)	Reference to `deals.id`
`buyer_id`	UUID (FK)	Reference to `users.id`
`amount`	DECIMAL	Final transaction amount
`status`	ENUM	`completed`, `refunded`, `failed`

`magic_matches`

Column	Type	Description
`id`	UUID (PK)	Unique match ID
`user_id`	UUID (FK)	Reference to `users.id` (Buyer)
`search_criteria`	JSONB	Saved criteria for automatic matching

Setup & Installation

Follow these steps to set up the development environment locally.

1. **Prerequisites:** Ensure you have Node.js (v18 or higher) and npm/yarn installed.
2. **Clone the repository:**

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

3. **Install dependencies:**

   ```
   npm install
   ```

4. **Environment Setup:** Copy the sample environment file and adjust any local variables if needed.

   ```
   cp .env.example .env
   ```

5. **Start the Development Server:**

   ```
   npm run dev
   ```

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

Admin Access: To access the `AdminPage.tsx` and administrative features, use the following credentials:

- **Username/Email:** `codeblix`
- **Password:** `codeblix`

Deployment Guide

Deploying this application is highly straightforward. **Shared hosting is completely enough; no Vercel or complex edge-networking is required.**

1. **Build the Application:** Run the Vite build command to compile the React application into static HTML, CSS, and JavaScript.

   ```
   npm run build
   ```

   This will generate a `dist/` directory containing your optimized, production-ready files.

2. **Prepare the Server:** Ensure your shared hosting environment (e.g., cPanel, DirectAdmin) is set up with a domain pointing to your `public_html` or equivalent root directory.

3. **Upload Files:** Upload the *contents* of the `dist/` directory directly into your server's public web directory.

4. **Routing Configuration (.htaccess):** Because this is a Single Page Application, navigating directly to a sub-route (like `/dashboard`) will result in a 404 error on a standard Apache server unless requests are routed back to `index.html`.

   The codebase already includes a `.htaccess` file inside the `public/` folder, which is automatically moved to your `dist/` folder during the build. This ensures that Apache seamlessly handles the React Router paths. Just verify that the `.htaccess` file was successfully uploaded to your shared hosting environment along with the rest of the build files.