PixelGrin Technical Documentation

Architecture Overview

PixelGrin is a full-stack Vertical SaaS marketplace designed for digital assets. It utilizes a **Single Page Application (SPA)** architecture for the frontend and a **Backend-as-a-Service (BaaS)** architecture for the backend.

High-Level Data Flow

1. **Frontend (Client):** The React application handles UI rendering, routing, and user interaction. It maintains state using React Hooks and directly subscribes to database changes via websockets.
2. **Backend (Serverless):** Convex acts as the unified backend, handling the database, file storage, and API functions (queries/mutations). It pushes real-time updates to the client.
3. **Authentication:** Clerk handles identity management. The frontend obtains a token from Clerk, which is passed to Convex to authenticate database requests.
4. **Payments:** Stripe handles payment processing.
    - **Direct Payments:** Stripe Checkout sessions for purchasing assets or paying onboarding fees.
    - **Payouts:** Stripe Connect is used to pay sellers (Providers) their share of sales.
    - **Webhooks:** A Convex HTTP action (`/stripe/webhook`) listens for payment confirmations to fulfill orders and update balances.

Tech Stack

Frontend Core

- **Framework:** React 19 (via Vite 6)
- **Language:** TypeScript (ESNext)
- **Styling:** Tailwind CSS (Utility-first CSS)
- **Routing:** React Router DOM v7
- **Animations:** Framer Motion

Backend & Data

- **BaaS Platform:** Convex (Database, Storage, Serverless Functions)
- **Database Type:** Real-time Document Store (JSON-like)
- **File Storage:** Convex File Storage (Images, Asset Zips)

Integrations & Services

- **Authentication:** Clerk (React SDK)
- **Payments:** Stripe (Standard & Connect)
- **PDF Generation:** jsPDF / jspdf-autotable (For license certificates)
- **Image Optimization:** browser-image-compression

Project Structure

```
FLIPPA-PixelGrin/
├── convex/                # Backend Logic (Serverless)
│   ├── _generated/        # Convex auto-generated types
│   ├── assets.ts          # Asset CRUD & Storage logic
│   ├── auth.config.ts     # Clerk JWT configuration
│   ├── http.ts            # Stripe Webhook endpoint
│   ├── schema.ts          # Database Schema definition
│   ├── seed.ts            # Data seeding script
│   ├── stripe.ts          # Stripe API integration
│   └── users.ts           # User & Profile management
├── public/                # Static assets & .htaccess
├── src/                   # Frontend Application
│   ├── components/        # Reusable UI components (Hero, Sidebar, etc.)
│   ├── lib/               # Utilities
│   ├── pages/             # Route views (HomePage, Dashboard, Admin, etc.)
│   ├── types.ts           # TypeScript definitions
│   ├── App.tsx            # Main Router & Auth Guards
│   └── main.tsx           # Entry point
├── .env.local             # Environment variables
├── package.json           # Dependencies
└── vite.config.ts         # Build configuration
```

Database Schema

The database is defined in `convex/schema.ts`. Below are the primary tables:

Table	Description	Key Fields

users	`clerkId` `role` `walletBalance` `stripeAccountId` `storefrontSlug` `isOnboarded`
assets	`creatorId` `price` `title` `storageId` `assetFileId` `category` `tags`
library	`userId` `assetId` `purchaseDate` `license`
transactions	`userId` `type` `amount` `status`
reviews	`assetId` `userId` `rating` `comment`
settings	`platformName` `onboardingPrice` `platformFee` `brandingLogoId`
posts	`authorId` `title` `content` `tags`
comments	`postId` `authorId` `content`

Setup & Installation

1. Prerequisites

- Node.js (v18 or higher)
- npm or yarn

2. Install Dependencies

Navigate to the root directory and run:

```
npm install
```

3. Environment Configuration

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

```
# Convex (Automatically set when initializing Convex)
VITE_CONVEX_URL=https://your-convex-instance.convex.cloud

# Clerk Auth
VITE_CLERK_PUBLISHABLE_KEY=pk_test_...
```

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

# Stripe Secret (Set inside Convex Dashboard, not here)
STRIPE_SECRET_KEY=sk_test_...
```

4. Initialize Convex Backend

Run the dev command to set up the remote backend and sync functions:

```
npx convex dev
```

- This command will prompt you to log in to Convex.
- **Important:** Go to your **Convex Dashboard** > **Settings** > **Environment Variables** and add:
  - `STRIPE_SECRET_KEY`: Your Stripe Secret Key (sk_test...).
  - `CLERK_SIGNING_KEY`: Your Clerk Issuer URL (e.g., `https://your-app.clerk.accounts.dev`).

5. Seed Data

Populate the database with test users, assets, and community posts:

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

6. Run Local Server

```
npm run dev
```

Access the app at `http://localhost:5173`.

Admin Credentials

The application includes a built-in admin panel located at `/admin`.

- **URL:** `http://your-domain.com/admin`
- **Username:** `codeblix`
- **Password:** `codeblix`

Note: Logging in triggers a mutation promoting the current authenticated Clerk user

to the 'admin' role in the database.

Deployment Guide

Backend Deployment

The backend logic resides on the Convex cloud platform. To deploy your functions to production:

```
npx convex deploy
```

Ensure you have set your production Environment Variables in the Convex Dashboard for the production deployment.

Frontend Deployment (Shared Hosting)

The frontend is a static React application. You do **not** need Vercel or Node.js hosting. Standard shared hosting (cPanel, Apache, Nginx) is sufficient.

1. **Build the Project:**

   ```
   npm run build
   ```

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

2. **Upload to Server:** Upload the **contents** of the `dist/` folder to your server's `public_html` (or www) directory.

3. **Routing Configuration:** The project includes a `.htaccess` file in the `public/` folder (which is copied to `dist/` during build). This file ensures that all routes (e.g., `/dashboard`, `/asset/123`) are directed to `index.html` so React Router can handle them.

   Ensure your host supports `.htaccess` (Apache).

Stripe Webhook Configuration

For payments to work in production:

1. Go to the Stripe Dashboard > Developers > Webhooks.
2. Add an endpoint: `https://<your-prod-convex-url>.convex.site/stripe/webhook`
3. Select events: `checkout.session.completed`.
4. Copy the **Signing Secret** (whsec_...).

4. Copy the **Signing secret** (`whsec_...`).
5. Add this secret as `STRIPE_WEBHOOK_SECRET` in your Convex Production Environment Variables.