Technical Documentation: GlamMyPet

Architecture Overview

GlamMyPet is a modern, vertical Software-as-a-Service (SaaS) marketplace designed for the luxury pet grooming industry. The application functions as a dual-sided platform:

1. **Consumer-Facing Marketplace:** Pet parents can discover groomers, view storefronts, and book appointments securely.
2. **Provider Business-in-a-Box:** Groomers are equipped with a dedicated dashboard to manage their services, working hours, appointments, and payments.

The platform is architected as a Single Page Application (SPA). It leverages a serverless backend for real-time data synchronization, secure file storage, and server-side execution of third-party integrations.

- **Authentication:** Delegated to Clerk for secure identity management.
- **Payments:** Powered by Stripe (Stripe Connect for provider payouts, Stripe Billing for platform subscriptions).
- **AI Features:** Integrates Google Gemini AI to help providers generate professional biographies.

Tech Stack

Frontend

- **Framework:** React 18 with TypeScript
- **Build Tool:** Vite
- **Styling:** Tailwind CSS

- **Animations:** Framer Motion
- **Routing:** React Router DOM
- **Icons:** Lucide React

Backend & Database

- **Serverless Backend / Database:** Convex (Handles real-time DB, file storage, and backend functions)
- **Auth:** Clerk (`@clerk/clerk-react`)
- **Payments:** Stripe Node.js SDK
- **AI / LLM:** Google Gen AI SDK (`@google/genai`)

Project Structure

```
FLIPPA-GlamMyPet/
├── .github/workflows/   # CI/CD deployment actions (FTP deployment)
├── components/          # Reusable UI components (Navbar, GroomerCard,
├── context/             # React Context for global state (AppContext ir
├── convex/              # Serverless Backend
│   ├── _generated/      # Auto-generated Convex types
│   ├── ai.ts            # Gemini AI integration for bio generation
│   ├── auth.config.ts   # Clerk authentication configuration
│   ├── bookings.ts      # Appointment creation, retrieval, and status u
│   ├── groomers.ts      # Provider profiles, services, and review manag
│   ├── http.ts          # Webhook receivers (Stripe events, File servir
│   ├── schema.ts        # Database schema definitions
│   ├── seed.ts          # Database population script with realistic Pex
│   ├── stripe.ts        # Stripe Connect and Checkout backend logic
│   └── users.ts         # User synchronization, role assignment, and pr
├── dashboard/           # Provider dashboard sub-components (Services,
├── pages/               # Application Routes (Home, AdminPanel, Storefr
├── public/              # Static assets and images
├── scripts/             # Utility and setup scripts
└── index.tsx            # Application entry point & Provider wrapping
```

Database Schema

The database is built on Convex and consists of the following core tables:

- `users`: Core identity table synced with Clerk. Tracks `role` (`provider`, `consumer`, `admin`) and `onboarded` status.
- `providers`: Groomer profiles linked to `users`. Stores `businessName`, `description`, `gallery`, `workingHours`, Stripe IDs, and visibility (`isPublished`).
- `consumers`: Pet parent profiles linked to `users`. Stores an array of `petInfo` (name, breed, age, special notes) for seamless booking.
- `services`: Grooming packages offered by specific providers, including `price` and `duration`.
- `orders`: Appointment records linking a `consumer`, `provider`, and `service`. Tracks the `appointmentTimestamp` and `status` (`pending`, `confirmed`, `completed`, `cancelled`).
- `reviews`: Ratings (1-5) and comments left by consumers on completed orders. Automatically recalibrates the provider's aggregate rating.
- `blockedSlots`: Granular calendar management for providers to block specific time ranges.
- `settings`: Global platform configuration (e.g., site name, platform subscription fee).

Setup & Installation

1. Prerequisites

- Node.js (v18 or v20 recommended)
- A Convex account
- A Clerk account
- A Stripe account
- A Google Gemini API Key

2. Install Dependencies

Clone the repository and install the Node modules:

```
npm install
```

3. Environment Variables

You need to configure variables in two places: local frontend (`.env.local`) and the Convex backend.

Create a `.env.local` file in the root directory:

```
VITE_CLERK_PUBLISHABLE_KEY=pk_test_...
VITE_CONVEX_URL=https://your-convex-deployment.convex.cloud
```

Set Convex Backend Variables: Run the following commands in your terminal to securely store backend secrets in Convex:

```
npx convex env set CLERK_SIGNING_KEY=your_clerk_jwt_key
npx convex env set STRIPE_SECRET_KEY=sk_test_...
npx convex env set STRIPE_WEBHOOK_SECRET=whsec_...
npx convex env set GEMINI_API_KEY=your_google_gemini_key
npx convex env set HOST_URL=http://localhost:3000
```

4. Start Development Servers

Start the Convex backend (this will continually sync your `convex/` folder to the cloud):

```
npx convex dev
```

In a separate terminal, start the Vite frontend:

```
npm run dev
```

5. Seeding the Database

To populate the database with realistic placeholder data (groomers, services, reviews), run the seed action via the Convex dashboard, or via CLI:

```
npx convex run seed:runSeed
```

Admin Access

To access the Platform Administration Panel (manage global settings and users), navigate to `/admin`.

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

Deployment Guide

Deploying this application is straightforward and highly cost-effective. Because the entire backend, database, and file storage are hosted on Convex, **the frontend is a static bundle**.

Standard shared hosting (cPanel, HostGator, Bluehost, etc.) is perfectly sufficient for the frontend. **No Vercel, Netlify, or dedicated Node.js server is required.**

1. Build the Frontend

Compile the React application into static files:

```
npm run build
```

This generates a `dist/` folder containing your `index.html`, JavaScript, and CSS assets.

2. Deploy Backend to Production

Push your backend functions and schema to your Convex production environment:

```
npx convex deploy
```

Note: Ensure you have set your production environment variables inside the Convex Production Dashboard (including updating the `HOST_URL` to your live domain).

3. Deploy Frontend via FTP

1. Log into your shared hosting account.
2. Open your File Manager or connect via an FTP client (like FileZilla).
3. Upload the entire contents of the `dist/` folder directly into your `public_html` or primary domain folder.
4. *Important:* The repository includes a `.htaccess` file configured for SPA routing. Ensure this `.htaccess` file is uploaded alongside your `index.html` so that direct URL visits (like `yourdomain.com/dashboard`) route properly to the React app instead of throwing a 404 error.

(Optional) Automated FTP Deployment: The codebase already includes a GitHub Action (`.github/workflows/deploy.yml`) configured for automated FTP deployments. If you wish to use this:

1. Go to your GitHub repository settings > Secrets and variables > Actions.
2. Add `FTP_SERVER`, `FTP_USERNAME`, and `FTP_PASSWORD`.
3. Add your production `VITE_CLERK_PUBLISHABLE_KEY` and `VITE_CONVEX_URL` secrets.
4. Pushing to the `main` branch will now automatically build and upload the files to your shared hosting environment.