Developer Roadmap: From Prototype to Production

This document outlines the necessary steps, tasks, and resources to evolve the AI Liaison Hub HTML prototype into a full-stack, production-ready web application.

Phase 1: Project Setup & Foundation (2-3 hours)

This phase establishes the development environment, version control, and the foundational structure for both the frontend and backend.

- **1.1. Set Up Development Environment**
  - **Task:** Configure a standardized local development environment to ensure consistency.
  - **Sub-Tasks:**
    - Install and configure Docker Desktop.
    - Create a docker-compose.yml file to manage the backend, database, and AI service containers.
    - Set up Visual Studio Code with the "Dev Containers" extension to work within the Docker environment.
  - **Resources:**
    - Docker
    - Visual Studio Code
    - VS Code Dev Containers
- **1.2. Initialize Version Control**
  - **Task:** Create a central code repository for tracking changes and collaboration.
  - **Sub-Tasks:**
    - Create a new private repository on GitHub.
    - Initialize a local Git repository in the project folder.
    - Create main and develop branches.
    - Add a .gitignore file to exclude unnecessary files (e.g., __pycache__, node_modules, .env).
    - Push the initial project structure to the GitHub repository.
  - **Resources:**
    - GitHub
    - Git Documentation
- **1.3. Scaffold Backend & Frontend Projects**
  - **Task:** Create the initial file and folder structures for the Django backend and React frontend.
  - **Sub-Tasks:**
    - Inside the project root, create a /backend directory and initialize a Django project.
    - Inside the project root, create a /frontend directory and scaffold a new React application using Vite.
    - Create top-level README.md and LICENSE files.
  - **Resources:**
    - Django Documentation
    - Vite

Phase 2: Backend Development (8-10 hours)

This phase focuses on building the server-side logic, database models, user authentication, and the core AI integration.

- **2.1. Build Database Schema & Models**
  - **Task:** Define the data structure for the application.
  - **Sub-Tasks:**
    - Configure Django to connect to a PostgreSQL database running in Docker.
    - Create Django apps for different functionalities (e.g., users, appointments, tools).
    - Define Django models for User, Appointment, ClientProfile, etc.
    - Run makemigrations and migrate to create the database tables.
  - **Resources:**
    - Django Models
    - PostgreSQL
- **2.2. Implement User Authentication & API**
  - **Task:** Create a secure way for users to sign up, log in, and manage their accounts.
  - **Sub-Tasks:**
    - Set up Django Rest Framework to build APIs.
    - Implement token-based authentication (e.g., JWT) for securing API endpoints.
    - Create API endpoints for user registration, login, logout, and profile management.
    - Implement a client portal view that is only accessible to authenticated users.
  - **Resources:**
    - Django Rest Framework
    - Django Rest Framework Simple JWT
- **2.3. Integrate Self-Hosted LLM for AI Tools**
  - **Task:** Set up the local AI model and create an API to interact with it.
  - **Sub-Tasks:**
    - Add Ollama and a selected LLM (e.g., Qwen3, Llama3.1) to the docker-compose.yml file.
    - Create a Django service or utility to send prompts to the Ollama API endpoint.
    - Build API endpoints for each of the free tools (Email Optimizer, Policy Generator) that take user input and return a response from the LLM.
    - Implement basic prompt engineering to guide the LLM's output for each tool.
  - **Resources:**
    - Ollama
    - Python requests library

Phase 3: Frontend Development (10-12 hours)

This phase involves translating the static HTML prototype into a dynamic, component-based React application.

- **3.1. Convert HTML/CSS to React Components**
  - **Task:** Break down the static prototype into reusable UI components.

    - **Sub-Tasks:**
        - Set up Tailwind CSS in the Vite/React project.
        - Create React components for each section of the page (e.g., Navbar.jsx, Hero.jsx, ServiceCard.jsx, Chatbot.jsx).
        - Use props to pass data to components dynamically.
        - Implement client-side routing (e.g., using React Router) if multiple pages are needed (e.g., for a client portal).
    - **Resources:**
        - React Documentation
        - Tailwind CSS
- **3.2. Implement State Management**
    - **Task:** Manage the application's data and UI state effectively.
    - **Sub-Tasks:**
        - Use React hooks (useState, useEffect, useContext) for managing local and global state.
        - Implement logic for handling user authentication state (e.g., storing tokens, showing/hiding content).
        - Manage the state for interactive elements like the chatbot and the tool modals.
    - **Resources:**
        - React State and Lifecycle
        - React Context
- **3.3. Connect Frontend to Backend APIs**
    - **Task:** Enable communication between the React client and the Django server.
    - **Sub-Tasks:**
        - Use a library like axios to make API requests from the frontend.
        - Create functions to handle user login/registration by calling the backend endpoints.
        - Wire up the tool modals to send user input to the backend AI tool endpoints and display the results.
        - Implement logic to handle API loading states, successes, and errors gracefully.
    - **Resources:**
        - Axios

Phase 4: Feature Integration & Finalization (4-6 hours)

This phase integrates third-party services for scheduling and payments and conducts final testing.

- **4.1. Integrate Appointment Scheduling**
    - **Task:** Add a self-hosted booking system for consultations.
    - **Sub-Tasks:**
        - Deploy the Easy!Appointments application using its provided Docker image.
        - Customize the look and feel to match the website's branding.
        - Embed the scheduling interface into a dedicated page or modal on the React frontend.
    - **Resources:**
        - Easy!Appointments

- **4.2. Implement Payment Processing**
  - **Task:** Allow the business to accept payments for services or donations.
  - **Sub-Tasks:**
    - Sign up for a Stripe developer account.
    - Integrate the Stripe API on the backend to create payment intents.
    - Use Stripe Elements on the frontend to create a secure payment form.
    - (Alternative) Deploy and configure an open-source solution like Lago.
  - **Resources:**
    - Stripe Docs
    - Lago
- **4.3. End-to-End Testing & SEO**
  - **Task:** Ensure all features work together seamlessly and the site is discoverable.
  - **Sub-Tasks:**
    - Test the full user workflow: registration -> login -> using a tool -> booking an appointment.
    - Ensure the site is fully responsive on mobile and tablet devices.
    - Add appropriate meta tags, titles, and descriptions for SEO using a library like React Helmet.
    - Generate a sitemap.xml and robots.txt.
  - **Resources:**
    - React Helmet

Phase 5: Deployment & Operations (3-4 hours)

This phase makes the application live and sets up processes for monitoring and maintenance.

- **5.1. Deploy Backend Services**
  - **Task:** Host the Django backend, PostgreSQL database, and AI service on a cloud platform.
  - **Sub-Tasks:**
    - Create an account on a platform like Render.
    - Create a new Web Service for the Django application, linking it to the GitHub repository.
    - Create a new PostgreSQL instance on Render.
    - Configure environment variables for SECRET_KEY, database credentials, etc.
    - Set up a background worker or separate service for the Ollama LLM if deploying it publicly.
  - **Resources:**
    - Render
- **5.2. Deploy Frontend Application**
  - **Task:** Host the static React application on a global CDN for speed.
  - **Sub-Tasks:**
    - Create an account on a platform like Netlify or Vercel.
    - Connect the platform to the GitHub repository.
    - Configure the build settings (e.g., npm run build) and the publish directory (dist).
    - Set up environment variables to point to the live backend API URL.
  - **Resources:**

        - Netlify
        - Vercel
- **5.3. Final Configuration & Go-Live**
    - **Task:** Point a custom domain to the deployed application and set up monitoring.
    - **Sub-Tasks:**
        - Purchase a custom domain name.
        - Update DNS records to point to the Netlify/Vercel frontend and potentially a subdomain for the backend API.
        - Set up basic monitoring and logging to track application health and errors.
        - Perform a final smoke test on the live production URL.
    - **Resources:**
        - Google Search Console