high-level plan

High-Level Project Plan: The Agentic App Genesis Engine

This plan outlines the architecture, phases, and technologies required to build a system that creates interactive web applications from natural language, inspired by v0.dev.

Project Vision

To create a multi-agent system that translates a user's abstract ideas into a functional web application prototype. The system will guide the user through a structured discovery process, visualize the application's architecture, and generate the initial codebase, with the ability to produce detailed development documentation for further expansion.

Core Architecture Philosophy

We will build this system on a **stateful, multi-agent architecture**. This is crucial because each step depends on the validated output of the previous one.

- **LangGraph** will be the backbone of our agentic flow. Its ability to create cyclical graphs allows agents to reflect, retry, and pass a shared "state" object (containing all user answers, wireframes, and decisions) between them. This is superior to a simple sequential chain for this complex task.
- **Pydantic** will be used extensively to define the structure of the data passed between agents. This ensures reliability and type safety for our state object, follow-up quizzes, and tool inputs.

Development Phases & Agent Responsibilities

This project will be built in distinct phases, with each phase managed by a specialized agent (or a graph of agents).

Phase 1: The "Discovery & Scoping" Agent

This agent's primary goal is to transform a vague user request into a concrete set of application requirements. It executes the first three steps of your flow.

- **Task:**
  1. **Initial Input:** Present the user with two options: a free-form text input or a guided, branching quiz to select a general app category (e.g., "e-commerce," "social media," "productivity tool").
  2. **Clarification & Enrichment:**
     - It analyzes the initial input. An internal "prompt evaluator" checks for key information.
     - If the input is insufficient, the agent asks up to three targeted follow-up questions. **These questions will be dynamically generated based on a master prompt.**
  3. **Structured Confirmation:**
     - Based on the answers, the agent generates two rounds of structured, multiple-choice quizzes (using Pydantic models to define the format) to lock in key decisions (e.g., "Target Audience: A) General Public, B) Businesses", "Monetization: A) Free, B) Subscription").
- **System Instruction Snippet for this Agent's LLM:**

```
You are a friendly and insightful Product Manager AI. Your goal is to help a user define their
app idea.
- If the user's request is vague, you MUST ask clarifying questions about the app's PURPOSE,
DESIGN STYLE, and KEY FEATURES.
- You will then generate structured multiple-choice questions to confirm these details.
- Your final output must be a JSON object containing all the user's validated choices.
```

- **Output:** A rich JSON object (`app_brief.json`) containing a structured summary of the user's requirements. This object becomes part of the main state passed to the next agent.

Phase 2: The "UI/UX Architect" Agent

This agent takes the confirmed requirements and visualizes the application's structure and flow.

- **Task:**
    1. Ingest the `app_brief.json` from the previous phase.
    2. Generate two distinct versions of **wireframes** for the main screens (e.g., homepage, dashboard).
    3. Generate two distinct versions of **user flow diagrams** for the core user journey (e.g., sign-up process, posting content).
    4. Present these visual options to the user for selection.
- **Technology for Image Generation:**
    - To fulfill this, you should use a powerful multimodal model. **Google's Gemini 1.5 Pro** is perfectly suited for this. It can understand the complex request and generate visual representations.
    - **Practical Implementation:** Instead of direct pixel-perfect images, it's more robust to have Gemini output a description in a format that can be rendered, such as:
        - **Mermaid.js or PlantUML syntax:** This is text-based code that can be easily rendered into clean flowcharts and diagrams.
        - **SVG code:** A vector format that can be directly embedded in HTML.
        - **A detailed JSON structure** describing the layout, which a frontend component can then render.
- **System Instruction Snippet:**

```
You are a Senior UI/UX Designer AI. Your input is a JSON app brief.
- Your task is to create wireframes and user flow diagrams based on this brief.
- You MUST output your diagrams in Mermaid.js syntax.
- For wireframes, you MUST output a structured JSON describing the layout of UI components
(e.g., { "type": "header", "children": [...] }, { "type": "button", "label": "Sign Up" }).
- Provide two distinct versions for each deliverable.
```

- **Output:** The selected wireframe/flow versions are added to the state object.

Phase 3: The "Code Generation" Agent

This is the final execution agent for the MVP. It synthesizes all prior decisions into a tangible code prototype.

- **Task:**
    1. Receive the final state object containing the app brief and the chosen UI/UX designs.
    2. Translate the structured wireframe JSON and user flows into code.
    3. Generate the code for **HTML, CSS, and JavaScript** files. The AI should create a clean, modular structure (e.g., `index.html`, `style.css`, `app.js`).
    4. Pass the generated file structure and content to the Sandbox for rendering.
- **System Instruction Snippet:**

```
You are an Expert Frontend Developer AI. Your input is a final project specification including a
JSON-based wireframe definition.
- You must write clean, standards-compliant HTML, CSS, and vanilla JavaScript.
- Create separate files for structure (HTML), style (CSS), and logic (JS).
- Your output MUST be a JSON object where keys are filenames (e.g., "index.html") and values are
the code content as strings.
- Do not invent new features; strictly adhere to the provided specification.
```

- **Output:** A JSON object representing the file system of the generated code.

Phase 4: The "Documentation & Roadmap" Agent (Post-MVP)

This agent provides the high-level documentation needed to turn the prototype into a real project.

- **Task:**
    1. Based on the final project state, generate a series of professional documents.
    2. The user can select which documents to create from a list.
- **Deliverables:**
    - **Higher-Level Plan:** A strategic overview.
    - **Project Architecture:** A technical diagram and explanation.
    - **Product Requirements Document (PRD):** A formal doc for product teams.
    - **Tech Specification:** Detailed spec for developers.
    - **Project Phases & Roadmap:** A timeline for development.
    - **Development Sprints & Tasks:** Break down the project into actionable sprints and tasks with sample code snippets.

Technology Stack & Key Decisions

- **Backend & Agent Orchestration:**
    - **Framework: FastAPI** (Python). It's modern, fast, and integrates seamlessly with the Python AI ecosystem.
    - **Agent Logic: LangGraph** to manage the stateful, multi-step flow between agents.
    - **Data Validation: Pydantic** to define all data structures.
- **AI & LLMs:**
    - **Core Logic & Text: Claude 3.5 Sonnet** or **GPT-4o** are excellent choices for reasoning and structured data generation.
    - **Multimodality (Phase 2): Gemini 1.5 Pro** for generating the wireframes and user flows.
- **Sandbox for Rendering:**
    - **The Problem:** You need to safely execute the generated code (HTML/CSS/JS) and display the result.
    - **Recommended Solution (for MVP): Docker**. It's the most straightforward and robust way to start. Your FastAPI backend can use the `docker-py` library to:
        1. Create a temporary directory with the generated files.
        2. Create a simple `Dockerfile` on the fly to serve these static files (e.g., using a basic Python HTTP server or Nginx).
        3. Build a Docker image and run a container from it, mapping a port.
        4. The frontend can then display the result from that container's URL in an `<iframe>`.
- **Frontend (For the Agentic App itself):**
    - **Framework: Next.js (React)** with TypeScript.
    - **UI Kit: Shadcn/UI** and **Tailwind CSS** for a modern, component-based interface.