1. INTRODUCTION

The **Akreage Launchpad MVP** is a **decentralized crowdfunding platform** for real estate. We're implementing:

- **Fundraising Contract**: Manages milestone-based funding.
- **AUSD Stablecoin** integration for project fundraising.
- **Governance Contract**: Handles on-chain approvals and voting (using Akres token or Council NFT logic, if applicable).
- **Akres Token** governance for vetting projects and releasing funds based on milestones.

Note: We are focusing on **client-side** logic with Supabase for storing off-chain data (like project details, user profiles) and relying on direct **on-chain interactions** for financial transactions and governance.

2. PROJECT PLAN (10 WEEKS)

2.1 Week 1: Requirements Gathering

1. **Finalize MVP Scope & User Flows**
    - Confirm how users (Investors, Developers, Council) will interact with the dApp.
    - Define the exact milestone release flow (e.g., how many milestones, who authorizes them).
    - Clarify any roles or NFTs needed for governance.
2. **Technical Prep**
    - Decide which EVM chain (e.g., Polygon, Ethereum, Avax) to deploy the contracts.
    - Draft the Supabase schema for project metadata (Projects table, Milestones, Users).
    - Outline how prompts in XO will be used to generate front-end components.

Deliverables

- Final user stories & acceptance criteria
- Supabase schema draft (Projects, Users, etc.)
- High-level outline of the two smart contracts (Governance & Fundraising)

2.2 Weeks 2–8: Development

We have **6 weeks** for development, **including 1 week of buffer** to absorb unexpected tasks or minor shifts in priority.

Week 2: Frontend (Using XO)

- **Generate Screens & Components**
  - Project listing page, project detail page, user dashboards (developer, investor, council).
  - Investment flow UI (connect wallet, input amount, confirm).
- **Basic Web3 Wallet Integration**
  - Allow users to connect via MetaMask or other EVM-compatible wallets.

Week 3&4: Wallet Integration

- **Create user controlled wallet on platform**
  - Allow users to create a wallet on the platform and sign up with Google, phone number or existing crypto wallet.
  - Users will be able to use social logins to connect to this wallet, it will create a new wallet on the fly. This will ensure even non-crpyto savvy folks can use this technology seamlessly.

Week 5: Supabase

- **Set Up Supabase Project**
  - Create tables: `Users`, `Projects`, `Milestones`, `Investments` (if off-chain logging is needed).
  - Configure Supabase Auth for role-based access (developer, investor, council).
- **Direct Integration**
  - Frontend calls to Supabase for reading/writing project data, user profiles.
  - Real-time updates if desired (e.g., project status changes).

Week 6: Smart Contracts (Governance & Fundraising)

- **Fundraising Contract**
  - **Core Logic**: Holds funds (e.g., stablecoin deposits) in escrow.
  - **Milestones**: Each project has milestone triggers. When governance approves a milestone, the contract releases the allocated tranche to the developer's address.
  - **Events**: Emitted on deposit, milestone release, or any state changes.
- **Governance Contract**
  - **Voting Mechanism**: Could use Akres token weighting, or a Council NFT gating approach.
  - **Proposal Lifecycle**: Project proposals (or milestone approvals) can be opened for voting. If approved, it signals the Fundraising Contract to release funds.
  - **Roles**: Possibly define an admin role for initial setup (e.g., who can create proposals or projects).

- **Deployment to Testnet**
    - Deploy both contracts to an EVM testnet (Polygon Mumbai, for example).
    - Basic unit testing with frameworks like Hardhat or Foundry.

Week 7: Integrations

- **On-Chain ↔ Frontend**
    - Integrate the Fundraising Contract's deposit and release functions into the front-end flows.
    - Connect the Governance Contract for creating/voting on proposals (milestone checks).
- **Supabase ↔ Contract Data**
    - Store references (project IDs, milestone IDs) that match on-chain data.
    - Optionally log transaction hashes in Supabase.

Week 8: Buffer / Final Dev Tasks

- **Buffer**
    - Address any unforeseen complexities from the previous weeks (UX refinements, minor contract changes).
    - Perform additional internal testing to verify end-to-end user flows.
- **Optimization**
    - Tidy up front-end code, finalize contract gas optimizations if needed.
- **Documentation**
    - Update code comments, instruct users on connecting wallets, creating proposals, etc.

2.3 Weeks 9–10: Testing & Feedback

1. **Internal Testing**
    - Verify each user flow (project creation, investing, milestone approval).
    - Confirm that governance actions (proposal creation, voting) function correctly.
    - Check Supabase permissions and data consistency.
2. **Pilot Feedback**
    - Invite a small group of real estate developers, prospective investors, or community members to test the system with test tokens on the chosen testnet.
    - Collect UI/UX feedback, bug reports, and suggestions for improvement.
3. **Bug Fixes & Finalization**
    - Prioritize high-impact fixes discovered in pilot testing.

- Ensure the Fundraising Contract and Governance Contract interactions are stable.

Deliverables

- Fully functional MVP deployed on testnet
- Documented feedback & bug fixes
- Ready-to-use dApp with front-end (XO-based), Supabase integration, and two smart contracts on testnet

3. SOFTWARE DESIGN OVERVIEW

3.1 High-Level Architecture

1. **XO Frontend**
    - Prompt-based generation of UI components for project listing, investment flow, and governance actions.
    - This front-end will be in NextJS / Vite JS.
    - Interacts with on-chain contracts via web3 libraries (ethers.js or web3.js).
    - Uses Supabase for storing project info, user roles, and additional metadata.
2. **Supabase**
    - **Tables/Views**: `Projects`, `Milestones`, `Users` (and possibly `Votes`, `Investments`).
    - **Auth**: Manage user sessions and roles (developer/investor).
    - Minimal logic: Much of the financial & governance logic is on-chain.
3. **Smart Contracts**
    - **Governance Contract**
        - Creates project proposals or milestone-approval proposals.
        - Holds voting logic.
        - Signals the Fundraising Contract upon successful votes.
    - **Fundraising Contract**
        - Escrows stablecoins.
        - Releases funds incrementally as each milestone is approved.
        - Tracks how much each project has raised and how much is claimable.

4. HIGH LEVEL OVERVIEW OF SMART CONTRACTS

4.1 Governance Contract

- **Purpose**: Decentralize project and milestone approvals.
- **Key Functions**:
    - `createProposal(projectId, milestoneId, ... )`: Records a proposal for a new project or a milestone release.
    - `vote(proposalId, voteChoice)`: Allows token holders (or NFT holders) to vote.
    - `executeProposal(proposalId)`: If the proposal passes, triggers an on-chain call to the Fundraising Contract to release funds or mark the project as approved.
- **Storage**:
    - A struct for each proposal (proposalId, description, expiration time, status).
    - Mapping of addresses to their vote status.

4.2 Fundraising Contract

- **Purpose**: Securely hold investor funds (in stablecoin) and release them based on governance decisions.
- **Key Functions**:
    - `invest(projectId, amount)`: Investors deposit stablecoins (e.g., AUSD) into the escrow for a specific project.
    - `releaseFunds(projectId, milestoneId)`: Callable only by the Governance Contract after a successful vote.
    - `refundInvestors(projectId)`: (Optional) If a project fails or does not meet a milestone, this function can allow partial or full refunds.
- **Storage**:
    - Track total raised per project and how much has been released so far.
    - Investor balances if partial refunds are needed.

By keeping the **Governance** and **Fundraising** logic separate, we ensure modularity:

- Governance can evolve or add new voting mechanisms without changing the escrow logic.
- Fundraising focuses solely on handling money flows based on authorized calls from the governance contract.