ChatGPT

Beginner-Friendly Blockchain Startup Plan: IDO on Sepolia

This guide presents a complete project plan for a simple blockchain startup and its tokenized IDO on Ethereum's Sepolia testnet. We focus on an easy concept (e.g. a community loyalty token), and detail everything from business vision and tokenomics to technical steps, templates, and documentation. Each section includes references to up-to-date sources for validation. The plan is structured to allow a total beginner to complete all steps in under 24 hours and prepare them for a 15-minute interview on their project.

1. Startup Concept

Idea: A blockchain-based *community loyalty token* for local businesses. Customers earn tokens for purchases or referrals, which can be redeemed for discounts or perks across partner merchants. Blockchain makes rewards transparent and portable. For example, analytics show that **blockchain loyalty programs** can create transparent, fraud-resistant reward systems, vastly improving on traditional loyalty schemes [1].

- *Unique Value:* Unlike closed loyalty points, these tokens are real crypto assets that users can trade or hold for value. This can **increase customer engagement** by directly aligning users' spending with token benefits [1] [2].
- *Example Inspiration:* The Props project did exactly this: it launched a community loyalty token and raised over $1.07M within hours on a crowdfunding platform [2]. This demonstrates strong market interest in simple loyalty-token concepts.

2. Company Profile & Fundraising

Vision & Mission

- **Vision:** Empower local communities by turning everyday rewards into blockchain tokens, fostering loyalty and participation.
- **Mission:** Issue an easy-to-understand loyalty token that customers earn and spend within a neighborhood ecosystem, promoting local commerce.
- **Products/Services:** A mobile/web app for wallets and reward tracking, plus partnerships with stores. Users earn tokens for shopping, referrals, or social engagement.
- **Target Audience:** Tech-savvy local shoppers and small businesses in Nigeria looking for modern loyalty programs. The target is initially a few merchant partners, scaling to wider community networks.

Business Model & Revenue

The token system is free for users. The **business model** relies on:
- A **transaction fee** (e.g. 1–2%) on token purchases/sales or when converting tokens to goods.

- **Partnership fees** from businesses that join the network.
- Potential revenue from premium features (analytics for merchants, marketing tools).

This model leverages token circulation: each token purchase (during the IDO or on open markets) generates immediate funding, and a small cut on every token sale can sustain operations over time.

Company Valuation & Assumptions

As a nascent startup with minimal tangible assets, assume a **nominal valuation** of around ₦20–50 million. This is an illustrative number reflecting a seed-stage Web3 project. It factors in the size of the local market, the scope of our pilot, and the novelty of the loyalty token idea. We justify this by comparing similar early-stage tech ventures in Nigeria and modest revenue projections (e.g. service fees and token sale proceeds).

Fundraising Target (in Naira)

We plan to raise **₦10,000,000** (ten million naira). This covers initial development, marketing, and operations. The target is set by estimating costs and the token sale size. For example, if we allocate 10% of tokens to the IDO and want ₦10M, the pricing adjusts accordingly (see below). The logic is transparent: selling a portion of tokens at a fixed price should yield the target amount.

Token Design

- **Name/Symbol:** *NaijaLoyal* (symbol **NLG**).
- **Total Supply:** 1,000,000 NLG tokens (a round number easy to manage).
- **IDO Allocation:** 10% (100,000 NLG) offered in the initial DEX sale on Sepolia.
- **Distribution (Tokenomics):** See table below for a sample plan. Typical tokenomics allocate only a small portion to founders and reserve a majority for community and future use [3] [4]. For example, Uniswap's $UNI token gave 60% to the community and only ~21.5% to the team [4]. We will follow best practices:

Category	% of Total Supply	Purpose
Founders & Team	15%	Long-term incentives (4-year vesting)
Advisors/Partners	5%	Early supporters, vested over 1–2 years
Community Rewards	30%	User incentives (airdrops, promotions)
Treasury/Development	25%	Future development, liquidity reserve
IDO (Public Sale)**	10%	Raising funds (this sale)
Private Investors/Seed	10%	Early backers, locked tokens
Total	**100%**	

This allocation balances control and decentralization. Industry sources suggest keeping the **team/founders share around 15–20%** and allocating a large portion (40–50%) for public/community use [3]. Our plan leaves ~40% combined for community and sale, aligning with these guidelines [3] [4].

Raise Calculation Example

To raise ₦10,000,000 by selling 100,000 NLG (10% of supply), each token must be priced at **₦100**. (100,000 tokens × ₦100 = ₦10,000,000). In USD terms at ≈₦1,500/USD, that's about $6.67 per token. This simplistic calculation ensures the target amount is met. Adjustments (e.g. offering bonuses or tiers) can fine-tune the final raise. The pricing is intentionally low to attract small retail investors (₦100 is less than $0.07).

3. Tokenomics & Distribution Details

Proper tokenomics fosters trust. As noted, many projects now set **community allocation ~40–50%** and **team ≈15–20%** [3]. Our above table follows this approach. The *Treasury/Development* pool ensures ongoing funding, while *Founders* and *Advisors* tokens have vesting to prevent early dumping. The IDO tokens sold raise the target funds. This strategy aligns with best practices for fair distribution [3] [4].

4. Templates and Boilerplate Resources

We provide resources and templates to simplify development:

- **ERC-20 Token Contract (Sepolia):** Use OpenZeppelin's ERC20 library. The OpenZeppelin Docs show how to create a token: e.g.

```
contract MyToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("Name","SYM") {
        _mint(msg.sender, initialSupply);
    }
}
```

  This pattern mints the total supply to the deployer [5]. A MetaMask developer guide also demonstrates a similar example using Hardhat and OpenZeppelin to create and deploy a token [6]. We can start with this template and adjust name, symbol, and supply. Beginners can use the OpenZeppelin Contract Wizard for a no-code way to generate a basic ERC-20.

- **IDO (Token Sale) Contract:** A simple sale contract can be implemented following common tutorials. For instance, one pattern is a `TokenSale` contract that holds tokens and implements a `buyTokens(uint256 n)` function which checks `msg.value == price * n` and then transfers `n` tokens to the buyer [7]. This approach fixes a token price and ensures correct payment before sending tokens [7]. In practice, our IDO contract (written in Solidity) will define a set price (e.g. 0.000066 ETH per NLG = ₦100 roughly) and automatically send tokens when users pay. We can derive this code from examples like ProgramTheBlockchain's token sale tutorial [7] and adapt for the Sepolia network.

- **Frontend Boilerplate:** We can use a minimal React or HTML/JS setup. Tools like **Scaffold-ETH** provide a ready-made React frontend with wallet integration [8]. Scaffold-ETH includes pre-built UI components (connect wallet button, transaction feedback) and hot-reloads contracts in the UI [9]. Alternatively, a simple Create React App with ethers.js will suffice: include a "Connect Wallet" button and functions that call the IDO contract's `buyTokens` method. Metamask's own docs show examples of writing frontend code with ethers.js and React hooks. No matter which, the key is to use Web3 libraries (ethers.js or Web3.js) to connect to Sepolia via a wallet (MetaMask) and call contract functions.

5. Documentation Examples & Process

Product Requirements Document (PRD)

Create a PRD outlining the product's features and specs: - **Overview:** A loyalty token sale app for community rewards (from the concept above). - **Features:** ERC-20 token, IDO sale, wallet login, token balance display, purchase tokens interface, sale statistics (remaining tokens, funds raised). - **Tech Stack:** Solidity contracts (ERC20, IDO), React/HTML frontend, Sepolia network. - **User Stories:** Examples like "As a customer, I connect my MetaMask wallet, see token price, and buy tokens."

(No direct citation needed; PRD is an internal document.)

AI Agent Prompts

Define the AI's role and instructions explicitly, for example: - **Frontend Agent Prompt:** "You are a React developer. Write a simple React component that connects to MetaMask, displays the connected account, and calls an `approveAndBuy()` function on our IDO contract when the user clicks a button." - **Smart Contract Agent Prompt:** "You are a Solidity smart contract engineer. Write an ERC-20 token contract named `NaijaLoyal` with symbol `NLG` and a mint in the constructor. Also write an IDO sale contract that sells this token at a fixed price."
These prompts should specify context (Sepolia network, ERC-20 standard) and constraints (use OpenZeppelin, testnet). *(No direct citation needed; this is an example of prompt engineering rather than factual content.)*

Testing Instructions

We will write automated tests (unit/integration) using Hardhat or Truffle: - **Smart Contracts:** Use Hardhat's testing framework with ethers.js. Test cases should cover: total supply set correctly, tokens transfer properly, `buyTokens` rejects wrong payment, and final balances after buys.
- **Frontend:** Manual or automated UI tests to ensure wallet connects and purchase triggers contract call. Since testnets are used, we may also include integration tests that simulate a purchase (Hardhat can fork Sepolia or use a local node).
(Generic advice; can cite Hardhat docs for testing if needed.)

Process Report Outline

Explain step-by-step how the project was built and integrated: 1. **Prompt Crafting:** Show examples of initial prompts given to AI tools for code generation (see above).
2. **Development:** Describe using Hardhat to scaffold projects and deploy contracts (see Tool section). Debugging steps: if a contract didn't compile, adjusting OpenZeppelin versions or fixing syntax.
3. **Validation:** After deploying, verify tokens by checking balances and making a test purchase. Use Hardhat logs or block explorer (Sepolia) to confirm success.

4. **Integration:** Combining frontend with contracts: e.g. connecting React to Hardhat by updating contract addresses in the frontend config. Testing user flows (connect wallet, click "Buy").
5. **Security Checks:** Ensure no integer overflow, proper use of `require`, and that only owner can end sale. Perhaps mention basic security best practices.

(Process report is narrative; no citation needed.)

GitHub Structure Overview

We should organize the repo as follows (similar to a standard Hardhat project) [10] : - `/contracts` : Solidity source files (`NaijaLoyal.sol`, `IdoSale.sol`). - `/scripts` : Deployment scripts (e.g. `deploy_token.js`, `deploy_ido.js`). - `/test` : Test scripts (e.g. `token.test.js`, `ido.test.js`). - `hardhat.config.js` : Hardhat configuration (network settings for Sepolia).
- `frontend/` (if separate): React or HTML files with `App.js`, components for wallet connect and purchase.
- `README.md` : Explains project, how to run scripts, tests, and frontend.
This mirrors the structure in Hardhat tutorials [10] . Each folder serves a clear purpose (contracts, scripts, tests).

6. Tools & Deployment

We recommend tools that minimize manual coding setup:

- **Hardhat:** A popular Ethereum dev environment for compiling, testing, deploying contracts [11] . Hardhat automates tasks and has a local network for testing. Example QuickNode guide calls Hardhat "a dev environment that helps compile, deploy, test, and debug contracts" [11] . We'll use Hardhat for writing and migrating our contracts to Sepolia.
- **Remix IDE:** A free, browser-based Solidity IDE. Remix allows writing, compiling, and deploying contracts directly in the browser [12] . It integrates with MetaMask for deploying to Sepolia [13] . For a beginner, Remix means no local setup; you can paste your Solidity code and deploy to Sepolia by selecting "Injected Web3" with MetaMask.
- **Scaffold-ETH:** A toolkit (React + Hardhat) that scaffolds a full dApp [8] . It includes a live-react frontend and Hardhat backend out-of-the-box. Using Scaffold-ETH can save hours: it comes with wallet-connect UI components and hot-reloading contracts [9] . We can fork the Scaffold-ETH 2 repo, replace the sample contract with our token/IDO, and have a working frontend very quickly.
- **OpenZeppelin Contracts:** A library of audited smart contract templates (ERC20, Ownable, etc). Using OpenZeppelin saves building basic functionality. The OpenZeppelin docs example shows how to define a token with `ERC20("Name","SYM")` and `_mint` [5] .
- **Front-End Libraries:** `ethers.js` or `web3.js` to interact with Ethereum in the frontend. For React, libraries like wagmi or Web3Modal make wallet connection easy. (These are optional; even a simple ethers.js script can connect to MetaMask with `window.ethereum`.)

Deployment Steps

Using these tools, deployment is as simple as: 1. **Get Sepolia Test ETH:** Use a faucet to fund your wallet with Sepolia ETH (QuickNode provides one [14]).
2. **Compile Contracts:** With Hardhat or Remix, compile the ERC20 and IDO contracts. Address any

compilation errors (e.g. update Solidity version or OZ imports).
3. **Deploy Token:** Deploy the ERC20 token on Sepolia. For example, using Hardhat a script might contain `const myToken = await MyToken.deploy(1000000);` [15] (the MetaMask guide shows a similar snippet).
4. **Deploy IDO Contract:** Deploy the sale contract with the token's address and price parameters. If using Hardhat, add Sepolia network config with an Alchemy/Infura URL and your wallet's private key.
5. **Configure Frontend:** In your app, set the contract addresses and ABI. Ensure the app uses the Sepolia network (network ID 11155111).
6. **Test Purchase:** On the deployed frontend, connect MetaMask (set to Sepolia) and attempt a token purchase. Check on Etherscan Sepolia or via console logs that balances update correctly.

7. Deployment Checklist

- [ ] **Environment Setup:** Install Node.js, then install Hardhat (`npm i hardhat`) or configure Remix.
- [ ] **Contracts:** Write or generate ERC20 and IDO contracts (e.g. via OpenZeppelin). Include comments for clarity.
- [ ] **Testing:** Write basic unit tests in Hardhat (balance, transfers, sale logic). Run `npx hardhat test` to ensure all pass.
- [ ] **Wallet & Sepolia:** Set MetaMask to Sepolia network. Use a Sepolia faucet (e.g. QuickNode's) to get test ETH [14].
- [ ] **Deploy Contracts:** Use Hardhat scripts or Remix to deploy contracts to Sepolia. Record the deployed addresses.
- [ ] **Verify in Block Explorer:** (Optional) Verify contract source on Sepolia Etherscan for completeness.
- [ ] **Frontend Connection:** In the web app, add MetaMask wallet connection. Display token price and remaining tokens.
- [ ] **Sale UI:** Create a "Buy Tokens" button that calls the IDO contract's purchase function. Show live status (pending/confirmed) from MetaMask.
- [ ] **Final Testing:** Perform a test purchase to ensure end-to-end functionality (wallet ➔ contract ➔ wallet update).
- [ ] **Documentation:** Prepare the PRD, include AI prompts used, finalize test cases and instructions, and push all code/documentation to GitHub.

Each item above aligns with best practices and tools discussed: Hardhat for scripting/tests [11], Remix/MetaMask for easy deploy [12] [13], and Scaffold-ETH if chosen for rapid prototyping [8].

Sources: This plan draws on current guides for blockchain development (e.g. MetaMask developer blog [6], tokenomics analyses [3] [4], and tool documentation [11] [12]) to ensure feasibility and up-to-date practices. All technical steps and design decisions are justified by these references.

---

[1] Blockchain Loyalty: The Future of Loyalty Programs
https://www.capillarytech.com/blog/how-blockchain-technology-is-re-animating-future-of-loyalty-programs/

[2] Creating community through blockchain — Republic
https://republic.com/blog/crypto/creating-community-through-blockchain

[3] Understanding Tokenomics in Crypto: The Importance of Initial Token Distribution - BlockApps Inc.
https://blockapps.net/blog/understanding-tokenomics-in-crypto-the-importance-of-initial-token-distribution/

[4] Token allocation : What is it ? - Tokenomics learning
https://tokenomics-learning.com/en/token-allocation/

[5] ERC20 - OpenZeppelin Docs
https://docs.openzeppelin.com/contracts/4.x/erc20

[6] [15] Ethereum Token Development for Beginner Web3 Developers
https://metamask.io/news/ethereum-token-development-for-beginner-web3-developers

[7] Program the Blockchain | Writing a Token Sale Contract
https://programtheblockchain.com/posts/2018/02/02/writing-a-token-sale-contract/

[8] [9] Scaffold-ETH: The Fastest Way to Prototype Ethereum dApps with React and Solidity | by BizThon | Global Business Hackathon | Medium
https://medium.com/@BizthonOfficial/scaffold-eth-the-fastest-way-to-prototype-ethereum-dapps-with-react-and-solidity-d02b89309d3a

[10] [11] What is Hardhat? | Alchemy Docs
https://www.alchemy.com/docs/what-is-hardhat

[12] [13] Ethereum News Today: Remix Ethereum IDE Boosts Smart Contract Development With Browser-Based Accessibility
https://www.ainvest.com/news/ethereum-news-today-remix-ethereum-ide-boosts-smart-contract-development-browser-based-accessibility-2507/

[14] Create an ERC20 Token in 3 Steps and Deploy It | QuickNode Guides
https://www.quicknode.com/guides/ethereum-development/smart-contracts/how-to-create-and-deploy-an-erc20-token