# Job Description

<https://www.freelancer.com/projects/javascript/Implement-Staking-Functionality/details>

We are seeking an experienced developer to implement staking functionality for our blockchain-based platform. The ideal candidate will have strong expertise in React and Ethereum development, with a proven track record of building and deploying staking mechanisms on the blockchain.

Responsibilities:

- Design and implement staking functionality using Ethereum smart contracts.

- Integrate staking contracts with a React frontend for seamless user interaction.

- Ensure secure and efficient handling of staking and reward distribution.

- Collaborate with our team to understand requirements and deliver high-quality solutions.

- Perform thorough testing and debugging to ensure reliability and security.

Requirements:

- Strong proficiency in React.js and building responsive UIs.

- Extensive experience with Ethereum, Solidity, and smart contract development.

- Familiarity with Web3.js or Ethers.js for blockchain interactions.

- Knowledge of Staking mechanisms, yield farming, and DeFi protocols.

- Understanding of security best practices in blockchain applications.

- Excellent problem-solving skills and attention to detail.

Preferred Qualifications:

- Experience with Truffle, Hardhat, or similar frameworks.

- Knowledge of IPFS, The Graph, and other decentralized technologies.

- Familiarity with TypeScript and state management libraries (like Redux).

- Previous experience with crowdfunding or DeFi platforms is a plus.

How to Apply:

Please provide a brief introduction and links to relevant projects or your GitHub profile. Let us know why you're the perfect fit for this role!

# Prompt:

I would like to write a solidity smart contract protocol that provides stacking features for users. Provide a brief, concise description (without going too much into details) for the various points

Provide a brief overview of what such a staking protocol needs to achieve and how this works

Can you list all required core features for a staking protocol

List some additional features that could be implemented

------------------------------------------------------------------------

If I want a smart contract protocol with the following features:

* Deposit – user deposits token
* Withdrawal - after a defined locking period
* Apply penalties for early withdrawal
* Reward Calculation (depends on staking duration and amount) – user can calculate reward at any time
* Use a fixed rate model for reward calculation
* Claim Rewards – user can claim accrued rewards after a specific delay
* Key parameters, like locking period, minimum stacking amount, reward rate… can only be set by the admin
* Tiered Rewards – Offer higher rewards for longer staking periods or larger amounts.
* Dynamic Reward Adjustment – Adjust rewards based on network activity or token supply.

How should the project be structured – should I use several smart contract files, Interfaces…? Should the protocol be split up into various contracts for reward management, controllers… or should everything be packed into one solidity file?

Provide a suitable file structure for the project.

# Proposal Letter Prompt

**Write a Professional Proposal Letter for a Freelance Staking Smart Contract Project**

Craft a compelling and tailored proposal for a staking smart contract project listed on a freelance platform. The proposal should be professional, client-focused, and structured to capture the client's attention quickly. Follow these guidelines:

1. **Introduction:** Start with a strong opening that introduces myself and immediately establishes why I am the ideal candidate for the project. Mention relevant experience and expertise to build credibility and trust.
2. **Client Needs and Challenges:** Show that you understand the client’s specific goals and pain points. Restate the key challenges mentioned in the job description to demonstrate that you’ve carefully read and understood their requirements.
3. **Proposed Solution:** Present a clear and strategic solution to the client’s problem. Explain how your skills, experience, and approach will effectively address their needs. Keep the explanation concise yet detailed enough to highlight your expertise.
4. **Unique Value:** Emphasize what makes you stand out—whether it’s your experience, innovative approach, or track record of successful smart contract projects. Focus on how your solution will deliver value to the client.
5. **Call to Action:** Close with a confident and clear next step. Encourage the client to reach out to discuss further or schedule a call. End on a positive and professional note.
6. Use I/my instead of we/our

**Use the following information to create the proposal:**

**## About Myself:**

I’m an experienced Solidity developer and smart contract auditor with over four years of specialized blockchain development expertise and more than 20 years in software engineering.

I’ve successfully developed secure and optimized smart contracts for various DeFi, NFT, and DAO projects, making me a strong fit for your token mining and NFT reward system on BSC. My deep understanding of smart contract design and gas efficiency ensures that your project will be both secure and gas-efficient.

My experience includes developing and auditing smart contracts for DeFi and NFT platforms, optimizing gas fees, and securing contracts against common vulnerabilities like reentrancy and overflow issues. I work with Solidity, Hardhat, Foundry, and Ethers.js, ensuring efficient development and testing. My track record of delivering high-quality, well-documented code on time reflects my commitment to excellence.

If you’re looking for a developer who can deliver a secure, and well-optimized smart contract, I’m ready to help. Let’s discuss your project further—I’m confident we can build a solution that meets your needs and exceeds expectations.

## **Project Details:**

We are seeking an experienced developer to implement staking functionality for our blockchain-based platform. The ideal candidate will have strong expertise in React and Ethereum development, with a proven track record of building and deploying staking mechanisms on the blockchain.

Responsibilities:

- Design and implement staking functionality using Ethereum smart contracts.

- Integrate staking contracts with a React frontend for seamless user interaction.

- Ensure secure and efficient handling of staking and reward distribution.

- Collaborate with our team to understand requirements and deliver high-quality solutions.

- Perform thorough testing and debugging to ensure reliability and security.

✅ **Proposed Solution - Use the information above to propose a solution:**

Mention, that I recently worked on a staking protocol, called Liquid Staking, a fairly complex liquid staking protocol that supports and facilitates staking on Chainlink. Here is the link to my Github repository: LINK

Regarding the frontend integration, I use a fairly basic yet complete example on “Smart Contract Development” course I provide for companies. You can take a look on my Github. Regarding the frontend integration, there are 2 interesting files:

LINK-FILE1

LINK-FILE1

The smart contract protocol could provide the following features – right now, of course, this is a very basic proposition:

* Deposit – user deposits token
* Withdrawal - after a defined locking period
* Apply penalties for early withdrawal
* Reward Calculation (depends on staking duration and amount) – user can calculate reward at any time
* Use a fixed rate model for reward calculation
* Claim Rewards – user can claim accrued rewards after a specific delay
* Key parameters, like locking period, minimum stacking amount, reward rate… can only be set by the admin
* Tiered Rewards – Offer higher rewards for longer staking periods or larger amounts.
* Dynamic Reward Adjustment – Adjust rewards based on network activity or token supply.

**General Design Approach:**

1. **Modular Architecture** – Keep core logic separated into multiple contracts to improve flexibility, upgradeability, and security.
2. **Separation of Concerns** – Split logic across different contracts (e.g., staking, rewards, governance) to reduce complexity and enhance maintainability.
3. **Interfaces and Abstract Contracts** – Use interfaces and inheritance to define standard behaviors and enable easier testing and upgrades.
4. **Upgradeable Contracts (Optional)** – Consider using a proxy pattern if you anticipate future upgrades.

**Provide A brief Contract Breakdown**

1. **StakingPool.sol** – Core Staking Contract

* Handles staking, withdrawals, and reward claiming.
* Manages locking period, deposits, and penalty logic.
* Interfaces with other contracts for reward calculation and penalties.
* Inherits Ownable for admin-level access.

🔹 **Implements:**

* deposit()
* withdraw()
* claimRewards()
* calculateRewards()

2. **RewardManager.sol** – Reward Calculation & Distribution

* Handles fixed and dynamic reward calculation logic.
* Adjusts reward rates based on network activity or token supply.
* Handles accrual of rewards over time.

🔹 **Implements:**

* setRewardRate() (admin-only)
* getRewardAmount()
* distributeRewards()

3. **TieredRewards.sol** – Tiered Reward Model

* Offers higher rewards for larger stakes or longer lock periods.
* Connects with RewardManager for calculating tier bonuses.

🔹 **Implements:**

* calculateTier()
* getTierMultiplier()

4. **EarlyWithdrawalPenalty.sol** – Penalty Handler

* Applies penalty if withdrawal happens before the lock period ends.
* Redistributes penalties to reward pool or protocol treasury.

🔹 **Implements:**

* calculatePenalty()
* applyPenalty()

5. **interfaces/** – Standardize interactions between contracts

* IStakingPool.sol – Interface for staking contract methods.
* IRewardManager.sol – Interface for reward calculation methods.
* ITieredRewards.sol – Interface for tier logic.
* IEarlyWithdrawalPenalty.sol – Interface for penalty handling.

➡️ Keep the tone professional yet approachable. Avoid being overly formal or too casual.

➡️ Focus on the client’s needs rather than talking too much about yourself.

➡️ Make the first few sentences impactful to quickly grab the client’s attention.

# My Proposal Letter

## Version 1 – Long

**Subject: Proposal for Staking Smart Contract Development**

Hello,

I’m excited about the opportunity to develop a secure and efficient staking mechanism for your blockchain-based platform. With over four years of specialized blockchain development expertise and more than 20 years in software engineering, I have the experience and technical knowledge to deliver a high-quality solution tailored to your project’s needs.

**Regarding your project:**

I understand that your goal is to implement staking functionality using Ethereum smart contracts and integrate it with a React-based frontend. Security, gas efficiency, and reliable reward distribution are critical to ensuring user trust and long-term platform success. My experience in developing and auditing DeFi, NFT, and DAO smart contracts has prepared me to address these exact challenges effectively.

**Experience and Relevant Work:**

I recently audited a fairly complex **Liquid Staking Protocol** that facilitates staking on Chainlink. You can review the implementation details in my GitHub repository:

<https://github.com/rspadinger/CH-Liquid-Staking>

Regarding the frontend integration using **Ethers.js**, I have a practical example from a **Smart Contract Development** course that I provide for companies. Relevant files that showcase how the frontend interacts with the smart contract are available here:

<https://github.com/rspadinger/ETH-Course/blob/master/06b_NFT-Minter/src/util/interact.jsx>

<https://github.com/rspadinger/ETH-Course/blob/master/06b_NFT-Minter/src/Minter.jsx>

**Proposed Solution:**

To meet your project’s requirements, I propose a smart contract that enables users to stake and withdraw a specific ERC20 token, featuring additional functionalities to enhance the staking experience. The contract automatically compounds rewards, includes customizable reward tiers based on staked amounts, and offers flexible parameters to fine-tune the staking process.

I have published a proof-of-concept (POC) of the contract on my GitHub:

<https://github.com/rspadinger/Staking-POC>

Additionally, I have deployed a version of the contract on the Sepolia testnet:

[https://sepolia.etherscan.io/address/0x841e3b679d022dff4e86fa7b6a39ca736c2529a9#code](https://sepolia.etherscan.io/address/0x841e3b679d022dff4e86fa7b6a39ca736c2529a9" \l "code)

**Key Features:**

**Auto-Compounded Rewards:**

* Rewards are automatically compounded, meaning that any earned rewards are added to the staked amount, generating more rewards over time.

**Reward Tiers:**

* The contract supports three reward tiers, which are adjustable by the admin. Users who stake higher amounts are eligible for higher reward rates, incentivizing larger stakes.

**Minimum Stake Amount:**

* The admin can set a minimum amount required for users to participate in staking.

**Lock Period and Early Withdrawal Penalty:**

* The contract allows the admin to set a minimum lock period during which staked tokens cannot be withdrawn. If users choose to withdraw before the lock period expires, they will incur an early withdrawal penalty.
* Admin Configurations:
* The contract allows the admin to configure:
* Reward tiers and their associated reward rates.
* Minimum stake amount.
* Lock period.
* Early withdrawal penalty.
* Annual reward rate (expressed in basis points).

**Simple Proof-of-Concept:**

This is a simple proof-of-concept (POC) at this stage, and the contract is not yet production-ready. The following are some important notes:

* Some features are still missing.
* The contract has not been tested thoroughly.
* The functions lack detailed comments for easier understanding.
* The contract provides a foundation for building a more complex and feature-rich staking system, but further work and testing are required before it is ready for live deployment.

**Why Me?**

My experience in building and auditing complex smart contracts, optimizing gas efficiency, and securing against vulnerabilities makes me well-equipped to deliver a reliable and secure staking protocol. My track record of delivering high-quality, well-documented code on time reflects my commitment to excellence.

**Next Steps**

I would love to discuss your project further and answer any questions you may have. Let me know a convenient time to connect - I’m confident that I can deliver a solution that exceeds your expectations.

Thank you for considering my proposal. I look forward to the opportunity to contribute to your project’s success.

Best regards,  
[Your Name]

---------------------------------------------------------------------------------------------------

**DELETED SECTIONS:**

**Design Approach:**

I propose a **modular architecture** to maximize flexibility, maintainability and security:

1. **Separation of Concerns** – Separate staking, rewards, and penalties into distinct contracts to simplify maintenance and upgrades.
2. **Interfaces and Inheritance** – Use interfaces and abstract contracts to ensure consistent behavior and easier testing.
3. **Upgradeable Contracts** – Optionally implement a proxy pattern (for example: OpenZeppelin’s UUPS) to enable future upgrades without affecting user funds.

**Contract Breakdown:**

1. **StakingPool.sol** – Core contract handling deposits, withdrawals, and rewards.
   * Manages user stakes and locking periods.
   * Interfaces with reward and penalty contracts.
   * 🔹 Functions: deposit(), withdraw(), claimRewards(), calculateRewards()
2. **RewardManager.sol** – Manages reward calculation and distribution.
   * Handles both fixed and dynamic reward models.
   * 🔹 Functions: setRewardRate(), getRewardAmount(), distributeRewards()
3. **TieredRewards.sol** – Implements tier-based reward structure.
   * Adjusts reward rates based on stake size and duration.
   * 🔹 Functions: calculateTier(), getTierMultiplier()
4. **EarlyWithdrawalPenalty.sol** – Applies penalties for early withdrawals.
   * Redistributes penalties to the reward pool or treasury.
   * 🔹 Functions: calculatePenalty(), applyPenalty()
5. **Interfaces** – Standardizes interactions between contracts.
   * IStakingPool.sol, IRewardManager.sol, ITieredRewards.sol, IEarlyWithdrawalPenalty.sol

## PROPOSAL LETTER - FINAL 1500 CHARS

Hello,

I’m excited about the opportunity to develop a secure and efficient staking mechanism for your blockchain-based platform. With over four years of specialized blockchain development expertise I have the experience to deliver a high-quality solution tailored to your project’s needs.

I recently audited a fairly complex Liquid Staking Protocol that facilitates staking on Chainlink: https://github.com/rspadinger/CH-Liquid-Staking

Regarding the frontend integration using Ethers.js, I have a practical example from a Smart Contract Development course: https://github.com/rspadinger/ETH-Course/blob/master/06b\_NFT-Minter

PROPOSED SOLUTION:

I propose a smart contract that enables users to stake and withdraw a specific ERC20 token. The contract automatically compounds rewards, includes customizable reward tiers based on staked amounts, and offers flexible parameters to fine-tune the staking process.

POC of the contract on my GitHub: https://github.com/rspadinger/Staking-POC

Deployed to Sepolia: https://sepolia.etherscan.io/address/0x841e3b679d022dff4e86fa7b6a39ca736c2529a9#code

KEY FEATURES:

\* Auto-Compounded Rewards

\* Reward Tiers

\* Minimum Stake Amount

\* Lock Period and Early Withdrawal Penalty

I would love to discuss your project further and answer any questions you may have and I’m confident that I can deliver a solution that exceeds your expectations.

Best regards,

Robert Spadinger

# Brainstorming - SC

## Overview Staking:

A staking protocol allows users to lock tokens in a smart contract to earn rewards over time. Users deposit tokens into the contract, which then calculates and distributes rewards based on staking duration, amount, and possibly other factors like performance or protocol revenue. The protocol should ensure fairness, security, and efficiency while maintaining flexibility for different staking strategies.

**Core Features of a Staking Protocol**

1. **Deposit & Withdrawal**
   * Allow users to stake tokens securely.
   * Enable withdrawal of staked tokens after a defined period or under specific conditions.
2. **Reward Calculation**
   * Calculate rewards based on staking duration and amount.
   * Support different reward distribution models (e.g., fixed rate, dynamic rate).
3. **Reward Distribution**
   * Distribute rewards fairly and efficiently.
   * Allow for claiming of accumulated rewards.
4. **Locking Periods**
   * Define fixed or flexible staking periods.
   * Optionally apply penalties for early withdrawal.
5. **Security**
   * Prevent reentrancy attacks, overflow/underflow issues, and other vulnerabilities.
   * Ensure contract ownership and permissions are well-managed.
6. **Ownership and Governance**
   * Include admin functions for upgrading contracts and modifying parameters.
   * Allow for DAO or community-based governance if applicable.

**Additional Features**

✅ **Auto-compounding** – Automatically restake earned rewards to increase returns.  
✅ **Tiered Rewards** – Offer higher rewards for longer staking periods or larger amounts.  
✅ **Multi-token Staking** – Support staking of different tokens with independent reward pools.  
✅ **Liquidity Provision Staking** – Allow users to stake LP tokens from AMMs.  
✅ **Penalty Redistribution** – Redistribute early withdrawal penalties to active stakers.  
✅ **Referral System** – Reward users for bringing new stakers into the protocol.  
✅ **Dynamic Reward Adjustment** – Adjust rewards based on network activity or token supply.  
✅ **Emergency Unstake** – Allow emergency withdrawal with penalties under defined conditions.

## How does the protocol make money?

Great questions! Let's break down the incentives and mechanics:

**1. Advantages for the Protocol Owner**

✅ **Fees** – The protocol owner can charge fees on various activities:

* **Deposit fees** – A small fee when users stake tokens.
* **Withdrawal fees** – A penalty or fee for early withdrawals.
* **Performance fees** – A percentage of the rewards earned by stakers.

✅ **TVL (Total Value Locked) Boost** – High staking participation increases the protocol’s

TVL, which can:

* Boost the protocol's reputation and attract more users.
* Increase the protocol's market valuation and influence in the ecosystem.
* Improve liquidity and stability of the protocol’s token if it's part of a broader DeFi ecosystem.

✅ **Token Utility & Demand** – If the protocol uses its own token for staking:

* Increased staking demand creates **buy pressure** on the token.
* Reduced circulating supply through staking can drive **token price appreciation**.

**2. How the Protocol Owner Funds the Rewards**

There are several common approaches for sourcing staking rewards:

💰 **Protocol Revenue** – Rewards can be funded from:

* Trading fees (if it's a DEX or liquidity pool).
* Lending/borrowing fees (if it's a lending protocol).
* Yield generated from staked assets in other protocols (e.g., Yearn-style strategies).

🔥 **Inflationary Model** – Mint new tokens to distribute as rewards (common with native tokens).

* Can work short-term, but may lead to token dilution if not managed properly.

🔄 **Reinvestment of Staked Tokens** – Protocol could actively use staked tokens to generate returns by:

* Providing liquidity to AMMs (e.g., Uniswap).
* Lending out staked tokens in lending protocols (e.g., Aave, Compound).
* Participating in validator networks (if applicable).

**3. What Happens to Locked Tokens?**

🔒 **Held in Contract** – If the contract just holds the tokens, they sit idle and earn no yield unless otherwise specified.  
🌾 **Productive Use** – If the protocol supports it, staked tokens can be deployed to generate yield (e.g., farming, lending).  
⚠️ **Risk** – Deploying staked assets introduces smart contract and liquidity risks, so careful risk management is essential.

**Example Strategy:**

* Protocol charges a 2% withdrawal fee and a 10% performance fee on rewards.
* Staked tokens are used in an AMM to earn trading fees and additional yield.
* The yield funds the reward pool, and performance fees provide extra revenue for the protocol owner.

👉 This creates a **self-sustaining loop** where the protocol’s income covers reward payouts, and fees incentivize long-term participation. 😎

## Additional Use Cases:

✅ **1. AMM / Liquidity Staking**

* Users provide liquidity (e.g., token pairs in a pool).
* Earn trading fees + possibly additional protocol incentives (e.g., governance tokens).
* Protocol owner takes a cut of trading fees and possibly rewards to stay profitable.

👉 **Real-World Example:** Uniswap, SushiSwap, Balancer

✅ **2. Liquid Staking or External Yield Generation**

* Protocol stakes user funds on another platform (e.g., ETH staking, Aave).
* Earn yield from the external platform.
* Take a cut of the yield as protocol revenue.

👉 **Real-World Example:** Lido (ETH staking), Rocket Pool

✅ **3. Incentivized Native Token Staking**

* Stakers lock the protocol’s native token.
* Rewards come from inflation (minting) or from protocol revenue.
* Goal: Increase token demand, reduce circulating supply, and drive up price.

👉 **Real-World Example:** Synthetix, PancakeSwap

🚀 **4. Protocol Revenue Sharing**

* Staking the native token gives the right to share in protocol revenue.
* Example: Protocol earns trading or platform fees → distributes part to stakers.
* Provides a direct incentive for holding and staking tokens.

👉 **Example:** GMX (stake GMX, earn part of the platform’s trading fees)

💼 **5. Governance-Based Staking**

* Stakers earn governance rights (e.g., voting power) + possible share of protocol revenue.
* Rewards can come from protocol income or token minting.
* Stakers are incentivized to act in the protocol’s long-term interest.

👉 **Example:** Curve (veCRV model – vote-escrowed staking for governance and higher rewards)

🔄 **6. Bonding + Staking (Olympus-style)**

* Protocol sells bonds (discounted tokens) in exchange for staked tokens or stablecoins.
* Protocol builds up a treasury and uses it to back the value of the staked token.
* Stakers earn rewards from treasury growth or inflationary emissions.

👉 **Example:** Olympus DAO (OHM)

🌉 **7. Cross-Chain Staking**

* Protocol bridges tokens to other chains to stake in high-yield opportunities.
* Earn cross-chain yield, take a cut, and pay stakers the rest.

👉 **Example:** Ankr (staking across multiple chains)

🎯 **8. Real-World Asset (RWA) Staking**

* Staking protocol tokenizes real-world assets (e.g., real estate, bonds).
* Stakers earn yield from RWA performance.
* Protocol takes a management fee or share of the yield.

👉 **Example:** Ondo Finance (tokenized RWA staking)

**Summary:**

The first three use cases you mentioned cover the **core models**. The additional models (revenue sharing, governance, bonding, cross-chain, RWA) introduce more sophisticated ways to drive incentives and returns.

👉 The key question is always:  
**"Where does the yield actually come from?"** – If it’s sustainable (protocol revenue, fees, or external yield), the model can scale. If it's purely inflationary (minting), long-term sustainability can become an issue unless balanced by protocol demand or value generation. 😎

## Architecture & Design

That's a solid feature set — you're essentially building a well-structured staking protocol with dynamic and tiered rewards, which adds complexity. 👍

💡 **General Design Approach**

1. **Modular Architecture** – Keep core logic separated into multiple contracts to improve flexibility, upgradeability, and security.
2. **Separation of Concerns** – Split logic across different contracts (e.g., staking, rewards, governance) to reduce complexity and enhance maintainability.
3. **Interfaces and Abstract Contracts** – Use interfaces and inheritance to define standard behaviors and enable easier testing and upgrades.
4. **Upgradeable Contracts (Optional)** – Consider using a proxy pattern if you anticipate future upgrades.

📁 **Suggested Project Structure**

**Project Root:**

staking-protocol/

├── contracts/

│ ├── Staking/

│ │ ├── StakingPool.sol

│ │ ├── RewardManager.sol

│ │ ├── TieredRewards.sol

│ │ ├── EarlyWithdrawalPenalty.sol

│ │ ├── interfaces/

│ │ │ ├── IStakingPool.sol

│ │ │ ├── IRewardManager.sol

│ │ │ ├── ITieredRewards.sol

│ │ │ └── IEarlyWithdrawalPenalty.sol

│ │ └── utils/

│ │ ├── SafeMath.sol

│ │ └── Ownable.sol

│ └── Token/

│ └── StakingToken.sol

├── scripts/ # Deployment and configuration scripts

├── test/ # Unit tests

├── deploy/ # Deployment scripts

└── README.md

🧱 **Contract Breakdown**

1. **StakingPool.sol** – Core Staking Contract

* Handles staking, withdrawals, and reward claiming.
* Manages locking period, deposits, and penalty logic.
* Interfaces with other contracts for reward calculation and penalties.
* Inherits Ownable for admin-level access.

🔹 **Implements:**

* deposit()
* withdraw()
* claimRewards()
* calculateRewards()

2. **RewardManager.sol** – Reward Calculation & Distribution

* Handles fixed and dynamic reward calculation logic.
* Adjusts reward rates based on network activity or token supply.
* Handles accrual of rewards over time.

🔹 **Implements:**

* setRewardRate() (admin-only)
* getRewardAmount()
* distributeRewards()

3. **TieredRewards.sol** – Tiered Reward Model

* Offers higher rewards for larger stakes or longer lock periods.
* Connects with RewardManager for calculating tier bonuses.

🔹 **Implements:**

* calculateTier()
* getTierMultiplier()

4. **EarlyWithdrawalPenalty.sol** – Penalty Handler

* Applies penalty if withdrawal happens before the lock period ends.
* Redistributes penalties to reward pool or protocol treasury.

🔹 **Implements:**

* calculatePenalty()
* applyPenalty()

5. **interfaces/** – Standardize interactions between contracts

* IStakingPool.sol – Interface for staking contract methods.
* IRewardManager.sol – Interface for reward calculation methods.
* ITieredRewards.sol – Interface for tier logic.
* IEarlyWithdrawalPenalty.sol – Interface for penalty handling.

6. **utils/** – Reusable Libraries

* **SafeMath.sol** – Math operations with overflow/underflow protection.
* **Ownable.sol** – Access control to restrict admin functions.

7. **StakingToken.sol** – ERC20-Compatible Token

* The staking token (could be an existing token or a new one).
* Optionally include minting/burning logic if needed for inflationary rewards.

🚀 **Why This Modular Setup Works:**

✅ Easier to test and upgrade components independently.  
✅ Cleaner separation of logic → reduces complexity and attack surface.  
✅ Reusability → RewardManager, TieredRewards, and PenaltyHandler can be reused in other staking pools or protocols.  
✅ Flexibility → New reward models or staking mechanisms can be integrated without modifying the core staking contract.

🔑 **Deployment Order:**

1. Deploy StakingToken (if new token).
2. Deploy RewardManager, TieredRewards, and EarlyWithdrawalPenalty.
3. Deploy StakingPool, connecting it to the reward and penalty contracts.
4. Set reward rates, lock periods, and other parameters via Ownable functions.

💡 **Pro Tip:**

* Start with StakingPool.sol as the main entry point — other contracts should remain loosely coupled to allow independent upgrades.
* You can also use a **proxy contract** (like OpenZeppelin’s TransparentUpgradeableProxy) if upgradeability is a key requirement.

This structure is scalable, secure, and flexible — exactly what you need for a professional-grade staking protocol. 😎

# ****Brainstorming - UI****

## Features Overview

stake: min, rewards+tier

withdraw & withdrawAll: penalty, lock time left

I will ask V0 to create a public UI (without any admin functions). Can you write a great prompt that allows V0 to create the design of the site. At the moment I only care about the design aspects (UI/UX) without worrying about the functions to be implemented (eg: calling certain smart contract functions – I’ll add this later on.

Here are some details about the required UI features:

1. Add a connect wallet button that allows the user to connect a MetaMask wallet (no other wallets are required.

2. Staking/Withdrawal functionality:

The attached screenshot: Stake-Main.png provides some ideas on on how the main staking features could be implemented

a: on top, display the currently selected chain (eg: Ethereum Mainnet) – ask the user to change to Ethereum Mainnet if this is not already selected

b: below, display the title: “Staking” and below the title a short paragraph tht tells the user that he can stack TokenABC and that there is a 3% penalty if the user wants to withdraw before the 30 day lockup period.

C: below is the actual staking area in a nicely designed card element.

C1: On top is a title: “Stake TokenABC”, below the currently total staked amount is displayed.

C2: Below, display the token icon and token ticker, next to it the staking APR, the users wallet balance and a “Stake” button”

If the user clicks the “Stake” button, a modal is displayed with the following details:

* A title on the top: “Stake TokenABC”
* Below, the user can enter the amount that should be staked into an input element
* The total amount the can be staked should be displayed (wallet balance) togeter with an “Stake All” button, that automatically fills out the input
* According to the staked amount, calculate the total annual rate in % (base rate + tier rate if appliccable)
* At the bottom, add the “Stake” button

C3: Below is the withdrawal area: Dispaly the currently staked amount of TokenABC including all accumulated gains (if any), display also the value in USD. Further, display the “Withdraw” button.

If the user clicks the “Withdraw” button, a modal is displayed with the following details:

* A title on the top: “Withdraw TokenABC”
* Below, the user can enter the amount that should be withdrawn into an input element
* The total amount the can be withdrawn should be displayed (staked amount + gains) togeter with an “Withdraw All” button, that automatically fills out the input
* If the user is inside the lock period, display the penalty percentage and calculate the penalty in tokens that will be withheld and display how maany days are left in the locck period
* At the bottom, add the “Withdraw” button

Furthermore, provide any additional ideas for the UI:

Did I forget anything important?

Should certain elements be displayed in a different way?

## V0 Prompt

Create a sleek and professional staking user interface for a DeFi platform. The goal is to design an intuitive and responsive UI where users can stake and withdraw tokens. Focus on clean design, user-friendly layout, and effective presentation of information. Use modern UI/UX principles similar to the style in the attached screenshot for consistency and professional appearance.

I’m using a Next.js boilerplate with typescript, Tailwind and shadcn. Generate components into a structured component library.

There are several screenshots attached – design the various pages and components In a similar fashion, but, still, make it different. I don’t want my page to look identical as the one in the screenshot, I prefer to have a uniquely designed page. Keep the design clean and professional, without too many colors.

Use he following 2 colors for certain elements, like card borders and other items, but use the sparingly: #3CC0DD and #203152 . Make sure, to pick appropriate colors for the Header and Footer that mix well with those 2 colors.

## \*\*1. Create a Layout page with a simple Header and Footer

### Header Section

- Display a \*\*Connect Wallet\*\* button (MetaMask only) on the top right of the header section. Style it prominently, but keep it consistent with the platform’s color scheme.

- On the left side of the header, display the company log – just use a placeholder for now, I’ll add the image later.

The attached image “Header.png” shows an example of how this could look like

### Footer Section

- On the left side of the Footer, display a few basic links, like: Terms, Privacy, Docs… I’ll add the correct link urls later

- On the right side are some social media icons, like; X, Github, Discord

The attached image “Footer.png” shows an example of how this could look like

---

### \*\*2. Staking Section\*\*

Create a main page (eg: page.tsx) that contains the Staking/Withdrawal features.

The attached image “Stake-Main.png” shows an example of how this could look like, but there will be a few differences.

#### \*\*(a) Title and Info\*\*

- Title: \*\*"Staking"\*\*

- Below the title, display a short description:

> "You can stake TokenABC and earn rewards. A 3% penalty applies for early withdrawals (before 30 days)."

---

#### \*\*(b) Staking Card\*\*

Use a centered, card-based layout for a clean and organized appearance.

- \*\*Card Header:\*\*

- Title: "Stake TokenABC"

- Display total staked amount (in tokens + USD equivalent).

- \*\*Main Card Body:\*\*

- Token Logo + Ticker

- Staking APR (in bold)

- User’s Wallet Balance

- \*\*Stake Button\*\* (prominent, but disabled if wallet not connected or balance insufficient)

---

#### \*\*(c) Withdrawal Card\*\*

- \*\*Card Header:\*\*

- Title: "Staked TokenABC"

- Display total staked amount (in tokens + USD equivalent)

- \*\*Main Card Body:\*\*

- \*\*Withdraw Button\*\* (prominent, but disabled if staked balance is zero)

---

### \*\*3. Additional UX Considerations\*\*

✅ Provide a subtle loading animation while staking/withdrawing.

✅ Use clear, color-coded feedback (e.g., green for success, red for errors).

✅ Ensure the UI is responsive on mobile and desktop.

✅ Add a tooltip icon next to APR, staking duration, and penalty percentage to provide extra info when hovered over.

✅ Keep the UI consistent with modern DeFi platform designs — clean typography, light/dark mode options, and soft hover effects.

**Add Later:**

- \*\*On Stake Button Click → Open Modal:\*\*

- Title: "Stake TokenABC"

- Input field for amount (with placeholder text "Enter amount to stake")

- Show \*\*"Max" button\*\* to auto-fill the maximum stakable amount

- Display calculated \*\*total annual reward rate\*\* based on base rate + tier rate

- \*\*Confirm Stake\*\* button at the bottom (highlight the button if all conditions are met)

- \*\*On Withdraw Button Click → Open Modal:\*\*

- Title: "Withdraw TokenABC"

- Input field for amount (with placeholder text "Enter amount to withdraw")

- Show \*\*"Max" button\*\* to auto-fill the maximum withdrawable amount

- If inside lockup period:

- Display \*\*penalty percentage\*\*

- Calculate and display the penalty amount

- Show how many days are left in the lockup period

- \*\*Confirm Withdraw\*\* button at the bottom

---

### \*\*Additional Ideas/Considerations:\*\*

- Consider adding \*\*a staking history section\*\* where users can see a list of past staking and withdrawal events.

- Allow the user to \*\*toggle between base and tiered reward models\*\* in the UI to show how different staking amounts/durations would impact earnings.

- Provide a \*\*“Learn More”\*\* link next to APR or staking terms that opens a modal with FAQs or external documentation.

Token contract: 0xdAcA9A0186C17A9B7772771D8C275f19279Ae125

Acc2: 0x46F98920C5896Eff11BB90d784D6D6001d74c073

Currently, the walletBalance in stake-card.tsx displays a mock value. Modify stake-card.tsx to display the actual token balance of the current user. The smart contract address of the token contract is in .env.local at :STAKING\_TOKEN

The following part needs to be modified:

<p className="text-sm text-muted-foreground">

Wallet Balance: {walletBalance} {tokenSymbol}

</p>

Later on we need other smart contract functions, so,, I would like you to create a specific file that handles all smart contract interactions at the appropriate location in the staking-ui project

Make sure to use ethers.js version 6. The contract ABI is at:

ln -s ../staking-contract/artifacts/contracts/StakingToken.sol/StakingToken.json ./abi/StakingToken.json

"postcompile": "cp artifacts/contracts/StakingToken.sol/StakingToken.json ../token-ui/abi/StakingToken.json"