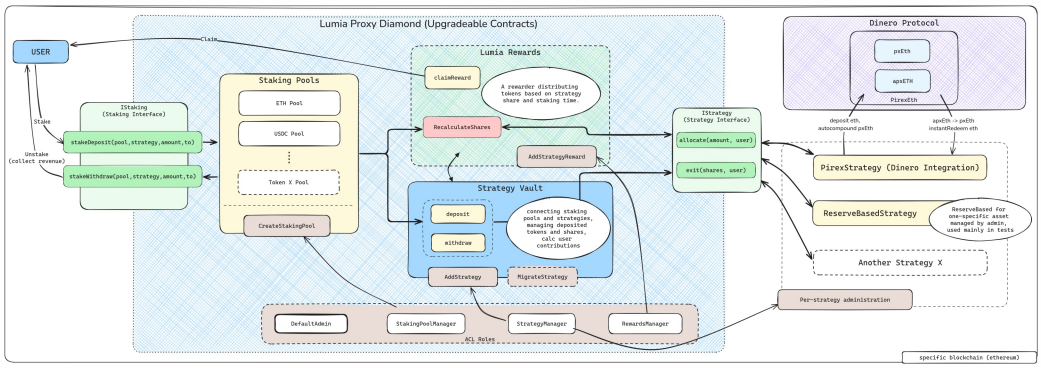
Lumia Smart Contracts

This specification proposal outlines the initial phase of development, focusing on staking, reward distribution, and revenue generation strategies. The contracts will be deployed on the ethereum blockchain and will utilize a Diamond Proxy architecture to allow modularity and upgradability.



0. Diamond Proxy Architecture

Lumia smart contracts will be developed using the Diamond Proxy pattern:

- **Modularity and Upgradability:** This architecture will allow for flexible upgrades and the addition of new functionalities without disrupting existing operations, ensuring the protocol remains adaptable.

Access Control List (ACL)

The **Access Control List (ACL)** in Lumia Smart Contracts defines roles like `LumiaRewardsManager`, `StakingPoolManager`, and `StrategyManager`, each with specific permissions to manage different parts of the protocol. The **DefaultAdmin** role holds the authority to assign and revoke these roles, ensuring controlled access to critical functions. Additionally, the **DefaultAdmin** role is also responsible for managing **proxy upgrades**, allowing it to oversee the upgrade process for contract proxies.

1. Staking Pools

The Lumia protocol features multiple staking pools, allowing tokens like ETH to be staked across different pools. Each pool operates independently, even for the same asset.

- **ETH Pools:** ETH can be staked into different pools, with rewards earned in both Ethereum and Lumia (Lumia Rewards) tokens. Multiple ETH pools can exist, each linked to different strategies.
- **Pool Identification:** Each staking pool is assigned a unique `poolId`, generated using the following Solidity function:

```
function generatePoolId(address stakeToken, uint96 idx) public pure returns (uint256) {
 return uint256(keccak256(
     abi.encodePacked(
         stakeToken,
         idx
     )
 ));
}
```

The `poolId` is based on the token's address and an index (`idx`), which represents the pool number for that asset. This ensures unique identification for each pool and provides an efficient way to manage contract storage. The `generatePoolId` function is publicly available for external systems to calculate the `poolId` as needed.

- **Strategy Assignment:** Each staking pool is assigned a strategy at creation. Only one instance of a strategy is connected to a pool, and the system validates that a strategy is not already assigned to another pool, preventing contract storage collisions
- **Staking Operations:** Since multiple pools can exist for the same token, users are required to choose which pool and strategy to use when staking. Users interact with the staking pools through the `stakeDeposit` and `stakeWithdraw` functions.
- **Adding New Pools:** New staking pools are added using the `AddStakingPool` function, which is restricted to the `StakingPoolManager` ACL role, ensuring that only authorized entities can create and configure new pools.

3. Lumia Rewards

Lumia will implement a rewards system to incentivize staking participants:

- **Token Airdrops:** Stakers will receive Lumia tokens as airdrops (other tokens can also be distributed). The distribution is based on each staker's contribution to a specific investment strategy and the period their stake is locked

within that strategy. Once tokens are unlocked from the strategy, reward accumulation stops for the unlocked portion.

- **Reward Distribution:** A reward distribution method similar to Synthetix's staking rewards model (as detailed in the video series) is used, customized to fit the specific needs of the Lumia project. This method involves calculating `rewardPerSecond`, dynamically adjusted based on the total stake in the pool at any given time.
- **Multi-Token Distribution:** It will be possible to distribute multiple tokens at the same time to the same user's strategy allocation. However, the number of these distributions will be limited to a configurable amount set by the manager (between 3 and 10). This limitation is in place because calculating distributions for each token is done separately, increasing the total transaction costs for the staking user.

The `addPoolReward` function, which adds rewards to a staking pool, is restricted. Only the LumiaRewardsManager ACL role is permitted to use this function to ensure controlled management of rewards and to prevent unauthorized changes. Additionally, the rewards are assigned properties such as a start and end time, allowing them to be defined in advance before they begin.

3. Revenue Strategies

Low-risk strategies are used to generate optimal income for stakers. All strategies implement a common interface, enabling seamless integration with staking pools and allowing new strategies to be added over time. The interface is defined in the IStrategy.sol contract and includes the following key functions:

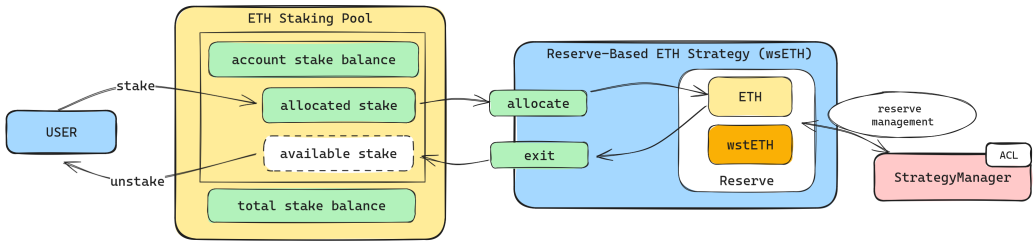
```
    /**
     * @notice Allocates a specified amount of the stake to the strategy
     * @param amount_ The amount of the asset to allocate
     * @param user_ The address of the user making the allocation
     * @return allocation The amount successfully allocated
     */
    function allocate(uint256 amount_, address user_) external payable returns (uint256 allocation);

    /**
     * @notice Exits a specified amount of the strategy shares to the vault
     * @param shares_ The amount of the strategy-specific asset (shares) to withdraw
     * @param user_ The address of the user requesting the exit
     * @return exitAmount The amount successfully exited
     */
    function exit(uint256 shares_, address user_) external returns (uint256 exitAmount);
```

This interface allows strategies to be deployed independently of the main upgradeable Proxy Diamond code and linked with new staking pools (migration of liquidity to different strategies for existing staking pools is planned)

- **APR Calculation:** Each strategy provides the necessary data for calculating the Annual Percentage Rate (APR), allowing for transparent tracking of returns for stakers.
- **Shares:** Are emitted by the strategies and moved to the `StrategyVault`. These shares represent the user's stake in the strategy, (vault may potentially issue derivative tradable liquidity tokens). Specific strategies examples are detailed below.

Example Strategy: Reserve-Based Strategy

One example of a strategy is a **reserve-based strategy** focused on yield generation through a specific defined asset (e.g., stETH from the Lido Protocol). This reserve is managed to ensure sufficient liquidity for staking and unstaking operations. When users stake ETH, the strategy allocates a portion of the available wstETH from the reserve to the user, allowing them to benefit from staking rewards generated by Lido.



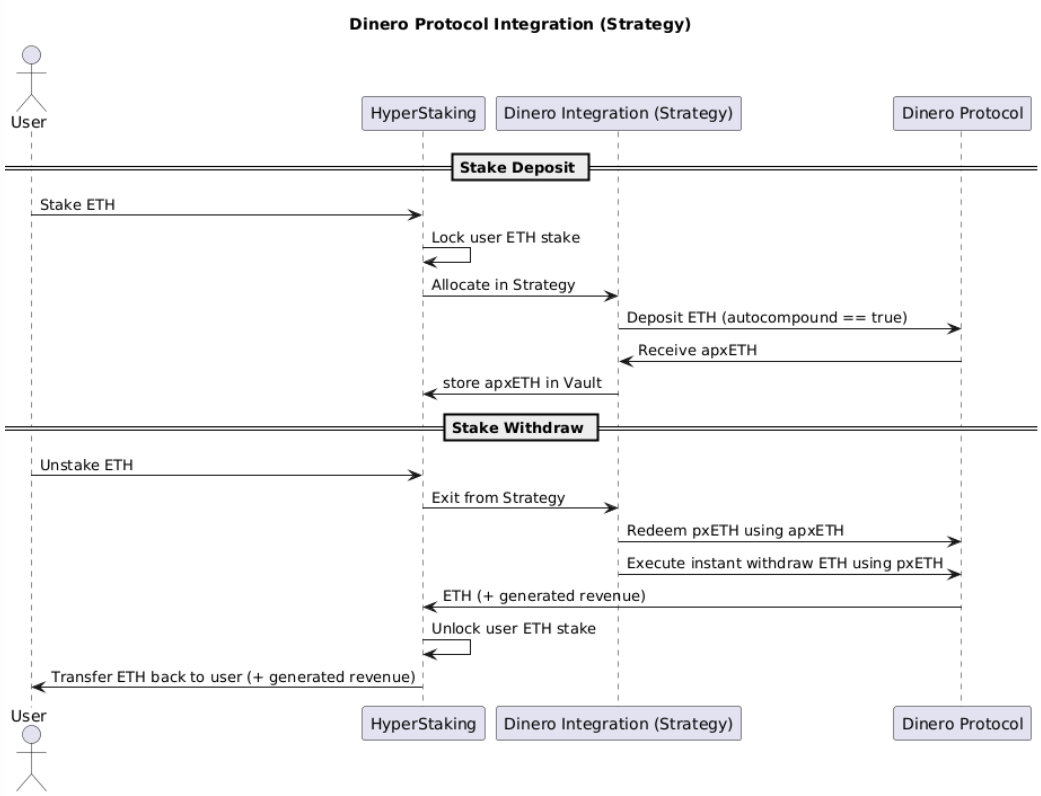
When users exit, the strategy returns their initial ETH plus the generated income, ensuring smooth exits without needing to interact with external protocols for each individual transaction. This approach minimizes transaction costs and optimizes the use of liquidity within the pool.

However, this solution has its limitations. It's possible that the strategy may not have full ETH coverage at certain times. In such cases, the user will still be able to perform a partial unstake. Additionally, the user will not lose any accrued revenue, as it is tracked within the contract, allowing them to claim their rewards once the reserve is replenished.

A simplified version of this strategy has been implemented and is currently being used for testing purposes.

Example Strategy: Dinero Protocol Integration

Another example strategy is the **Dinero Protocol Integration**, focused on yield generation through the **apxETH** token, emitted by the `PirexETH` contract from the Dinero Protocol. The strategy auto-compounds pxETH into apxETH to maximize returns, generating around 8% APY, and is stored in the Lumia `StrategyVault`.



When users stake ETH, the strategy interacts directly with the Dinero Protocol, converting ETH into pxETH, which is then auto-compounded into apxETH. This allows users to benefit from the compounding returns offered by the Dinero Protocol.

When users unstake, the Dinero Protocol is used to redeem pxETH from apxETH (an ERC-4626 vault). pxETH is then converted to ETH for withdrawal, plus accumulated interest, with a 0.5% fee applied.

In the future, the fee could be reduced by implementing a delayed unstake option, creating an unstake buffer for ongoing operations, similar to the model used in the Dinero Protocol.