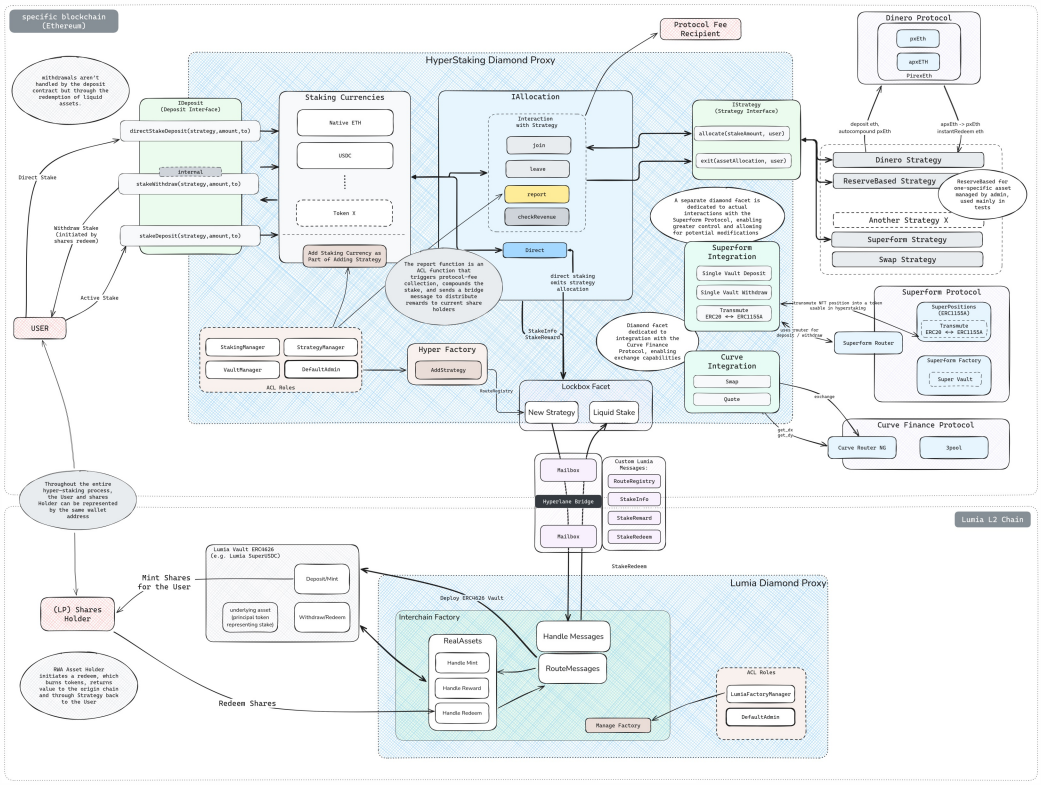
Lumia Smart Contracts

This specification describes the architecture and key features of the Lumia Smart Contracts, which manage staking pools, cross-chain asset handling, and contract upgrades.

The system follows a star architecture, centered on a Diamond Proxy contract deployed on the Lumia Chain to handle interchain communication, ERC-4626 shares minting, and revenue distribution, and multiple Diamond Proxy contracts deployed on various origin chains (e.g., Ethereum, Base, Arbitrum) to manage local deposits, staking pools, and revenue strategies.



0. Diamond Proxy Architecture

Lumia smart contracts is developed using the Diamond Proxy pattern:

- **Modularity and Upgradeability**: The architecture supports adding or replacing functionalities without affecting ongoing operations, ensuring flexible contract modularity.
- **Second Diamond Proxy on Lumia Chain**: The secondary Diamond Proxy operates on the Lumia Chain to manage cross-chain asset handling, effectively functioning as a bridge. It facilitates Hyperlane interchain messaging and ensures that liquid RWA tokens accurately represent real value.

Access Control List (ACL)

The **Access Control List (ACL)** defines roles like `StakingManager`, and `VaultManager`, 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 add, replace and delete specific contracts functionality.

Hyperlane Integration

Hyperlane provides secure cross-chain messaging between origin-chain Diamond Proxies and the central Lumia Chain Diamond Proxy. This ensures: +

- **Direct Staking**: Direct staking allows users to stake assets directly on the Lumia Chain without needing to pass through the revenue strategy. This enables faster processing and more flexible staking options, while still maintaining cross-chain compatibility through Hyperlane's messaging infrastructure.
- **Stake Synchronization**: When a user stakes on an origin chain, a Hyperlane message is sent to mint ERC-4626 shares on the Lumia Chain representing the stake plus accrued revenue.
- **Redemption Flow**: When a user redeems or exits shares on the Lumia Chain, Hyperlane relays a burn-and-withdraw instruction to the origin-chain proxy to burn the shares and release the underlying assets and rewards.

This design centralizes staking and redemption logic around Hyperlane messaging, enabling users to seamlessly access multi-chain revenue streams without interacting directly with multiple protocols or bridges.

1. Handling Multiple Deposit Currencies

The Lumia protocol supports staking with both native assets (e.g., ETH) and ERC-20 tokens. In some cases, even NFTs are indirectly supported through certain strategies (e.g., **Superform**). This allows for flexible staking operations across different asset types.

- **Native and ERC-20 Tokens:** The protocol distinguishes between native chain coins (such as ETH) and ERC-20 tokens using the `Currency` struct:

```
/**
 * The Currency struct represents a token
 * If `token` is address(0), it represents native coins (e.g. ETH)
 * @param token Address of the token, address(0) means native chain coin (e.g. ETH)
 */
struct Currency {
    address token;
}
```

- **Strategy Flexibility:** Each strategy can support different types of assets, including both native and tokenized assets. Strategies like **Superform** can even support NFTs indirectly by wrapping them in a compatible format.
- **Unified Management:** Despite supporting various asset types, the system maintains a unified interface for deposits, ensuring consistent behavior across different strategies and asset classes.

2. Revenue Strategies

Low-risk strategies are designed to safely deploy staked assets into yield-generating protocols or simply hold value in a compatible form. Each strategy adheres to a common interface, allowing seamless integration with vaults and staking pools while remaining decoupled from the core diamond proxy logic. New strategies can be added without affecting the other ones.

2.1 Core Interface: Allocation & Exit

These two functions form the backbone of any stake strategy:

```
    /**
     * @notice Allocates a specified amount of staked assets into the strategy.
     * @param stakeAmount_ The amount of stake (ERC-20 or native) provided for allocation.
     * @param user_ The address of the depositor initiating the allocation.
     * @return allocation The effective amount successfully allocated to the strategy.
     */
    function allocate(
        uint256 stakeAmount_,
        address user_
    ) external payable returns (uint256 allocation);

    /**
     * @notice Redeems strategy shares and returns underlying assets to the vault.
     * @param assetAllocation_ The amount of the strategy's share tokens to withdraw.
     * @param user_ The address of the vault or user requesting the exit.
     * @return exitAmount The amount of underlying assets returned to the caller.
     */
    function exit(
        uint256 assetAllocation_,
        address user_
    ) external returns (uint256 exitAmount);
```

- **allocate:** Moves assets from the vault into the strategy, issuing or updating internal share accounting. Must handle asset deposits appropriate to the strategy (e.g., ERC-20 tokens, native currency, or other supported asset types).
- **exit:** Converts strategy shares back into underlying assets and transfers them to the vault. Determines the current exchange price and handles asset valuation to compute `exitAmount`.

These functions allow the vault to treat all strategies uniformly, regardless of the underlying protocol or mechanics.

2.2 Strategy Type Flags

To support different flow patterns, each strategy also exposes two boolean flags:

```
    /**
     * @notice Indicates whether the strategy is a DirectStakeStrategy.
     * @dev  Direct stake strategies bypass yield-generating logic and
     *       exist solely to allow 1:1 deposits into the vault. They:
     *         - Store currency info for compatibility with the vault.
     *         - Revert on any `allocate`, `exit`, or preview calls.
     *         - Perform no token or native transfers.
     * @return Always `true` for direct stake strategies, `false` otherwise.
     */
    function isDirectStakeStrategy() external view returns (bool);

    /**
     * @notice Returns true if this stake strategy is an integrated strategy.
     * @dev  Integrated strategies delegate all asset movements to the
     *       IntegrationFacet within the same diamond. As a result:
     *         - No calls to `transferFrom` or native pull operations.
     *         - No ERC-20 approvals managed by the strategy itself.
     *         - `allocate(...)` and `exit(...)` simply forward to the facet,
     *           which executes internal transfers between strategy and vault.
     * @return Always `true` for integrated strategies, `false` otherwise.
     */
    function isIntegratedStakeStrategy() external view returns (bool);
```

- **Direct strategies** are effectively placeholders: they exist to satisfy the vault's strategy interface without moving funds into any external protocol. Perfect for simple 1:1 deposit/withdraw flows.
- **Integrated strategies** rely on the diamond's IntegrationFacet to handle every asset movement internally. No external allowance setup or approval calls are needed; the facet's permission covers both ERC-20 and native transfers.

This separation of allocation/exit logic from strategy-type flags ensures clarity: the vault always calls `allocate` and `exit` in the same way, while each strategy declares its operational mode via its flags.

- **Shares:** Are minted on the Lumia Chain as ERC-4626 tokens, representing users' stake within a strategy and used to distribute revenue generated by that strategy.

Reward Distribution via `report()` Function

A central `report()` function for each strategy aggregates yields from and updates share value.

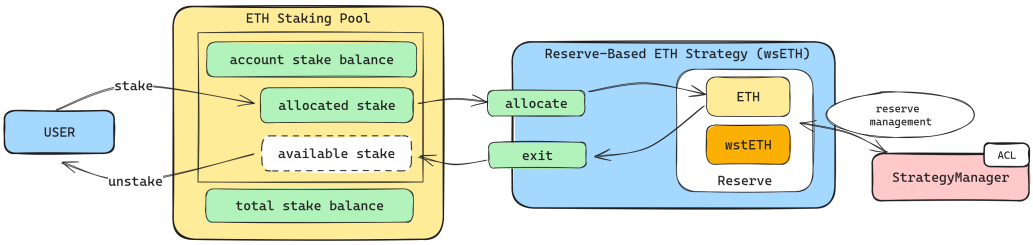
```
 /**
  * @notice Harvests and compounds revenue for a given strategy
  */
function report(address strategy) external;
```

Flow:

1. **Trigger**: authorized manager calls `report()`.
2. **Accounting**: Increases total assets; recalculates `pricePerShare = totalAssets / totalShares`.
3. **Cross-chain Distribution**: Emits `RewardsReported(amount)` event; Hyperlane relays report to origin proxies.
4. **User Update**: Share price bump represents distributed rewards; user share holdings automatically reflect yield.

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.



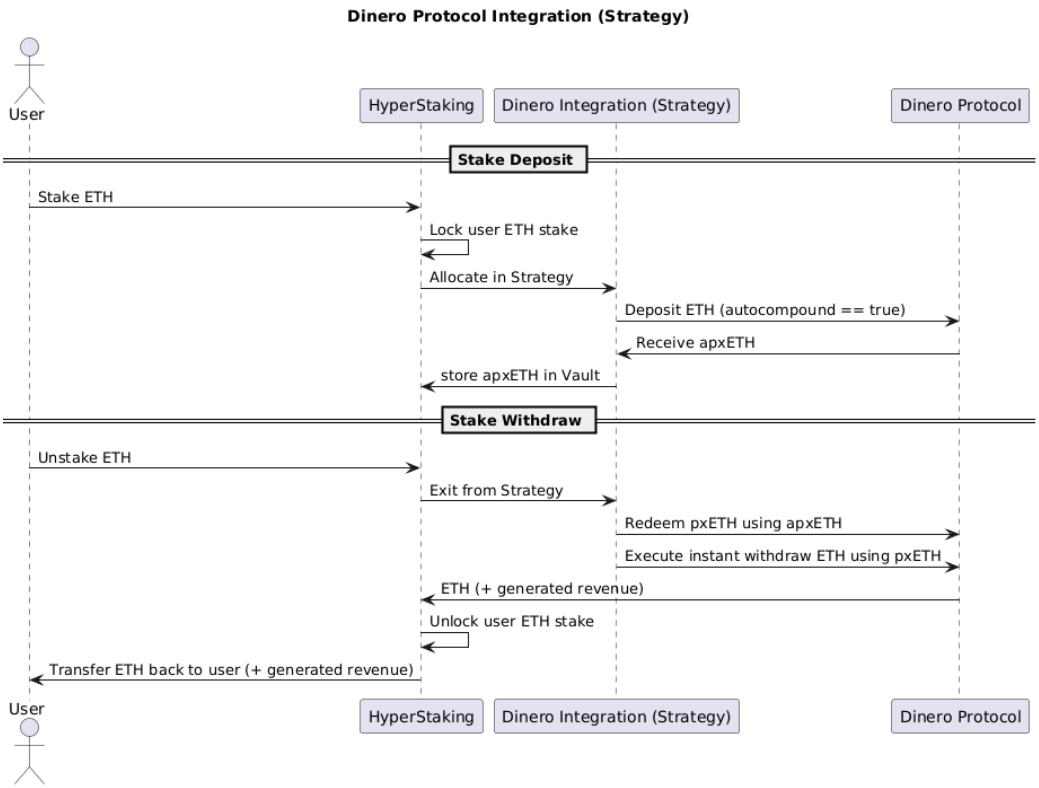
When users exit, the strategy returns their initial ETH plus the generated income, ensuring smooth exits without needing to interact with external protocols for each individual transaction. This approach minimizes transaction costs and optimizes the use of liquidity within the pool.

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.

Strategy Examples: Superform Strategy

The Superform Strategy is an **integrated** strategy that interacts with its dedicated `SuperformIntegrationFacet` within the diamond. On allocation, it:

1. Receives USDC deposits via the `allocate` call.
2. Queries the Superform protocol's on-chain for the current USDC -> SuperUSDC conversion rate, which reflects accrued interest and floating rates.
3. Calculates the exact amount of SuperUSDC to mint.
4. Invokes the `SuperformIntegrationFacet` to perform the internal transfer: moving USDC out of the allocation vault and minting SuperUSDC directly by the protocol.

On exit:

1. The strategy calculates the redeemable USDC amount from SuperUSDC using the latest rate.
2. Calls the `SuperformIntegrationFacet` to transmute ERC-20 SuperUSDC back into its ERC-1155 NFT representation.
3. Invokes the `SuperformIntegrationFacet` to burn the ERC-1155 SuperUSDC and credit the corresponding USDC back to the user.
4. All transfers remain within the diamond, leveraging its internal asset registry and permissioning.

This approach ensures:

- Tight integration with Superform's yield accrual.
- Zero on-chain approval overhead.
- Accurate valuation via on-chain rate feeds.

Strategy Examples: Swap Super Strategy

The Swap Super Strategy is also **integrated** and uses two dedicated facets— `CurveIntegrationFacet` and `SuperformIntegrationFacet` to orchestrate a two-step flow:

1. **USDT -> USDC (Curve 3pool):** Uses Curve's large, highly liquid 3pool to swap incoming USDT for USDC in a single step, executed internally via `CurveIntegrationFacet`.
2. **USDC -> SuperUSDC (Superform):** Forwards the obtained USDC to `SuperformIntegrationFacet`, which mints SuperUSDC as in the Superform Strategy.

On exit, the strategy reverses these steps:

1. Burns SuperUSDC via the Superform facet to receive USDC.
2. Swaps USDC back to USDT using the Curve 3pool facet's optimal path logic.
3. Returns USDT to the user, all through internal facet calls.

By combining Curve's efficient pool mechanics with Superform's yield-boosting wrapper, the Swap Super Strategy offers a seamless entry from USDT into interest-bearing SuperUSDC and exit back to USDT, **fully integrated** within the diamond's logic and asset management framework.