0.1 Architecture Design

0.1.1 Overall design

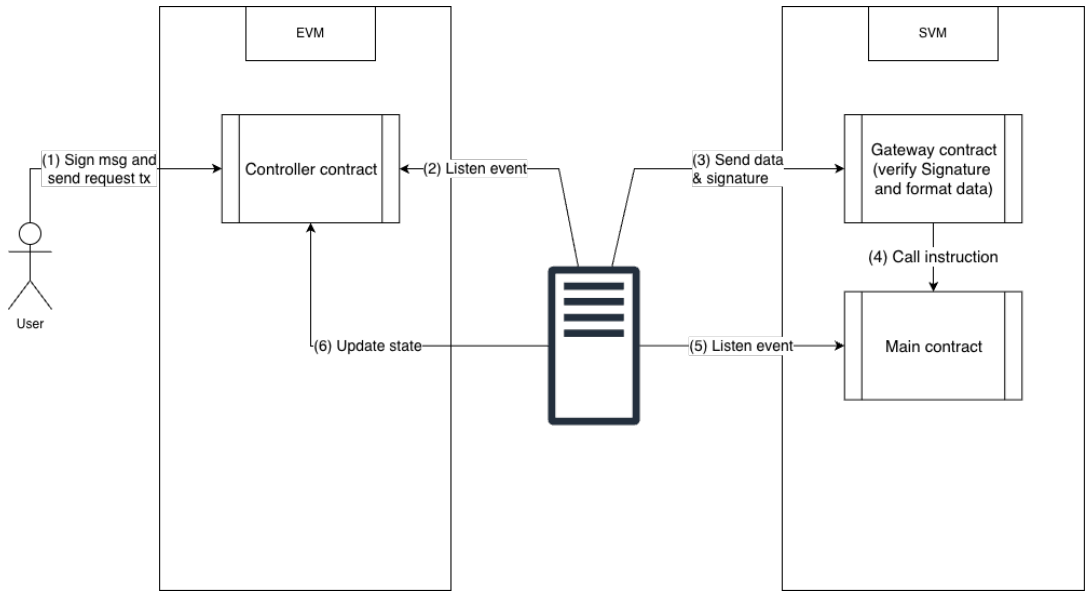
The architectural design of the Multi-chain Stablecoin Protocol is founded upon a Hybrid Event-Driven Microservices framework, structured within a Hub-and-Spoke topology. This sophisticated architecture is engineered to resolve the inherent challenges of liquidity fragmentation and state synchronization across heterogeneous blockchain networks. By strictly decoupling the Asset Custody Layer (residing on EVM chains) from the State Execution Layer (residing on the Solana SVM), the system achieves a high-performance, scalable solution that leverages the distinct advantages of each blockchain environment.

The architecture is composed of three primary execution environments, each functioning as an autonomous system component yet interconnected through a secure event-driven pipeline. The first environment is the EVM Spoke, which hosts the Controller Contract. This contract functions as the user's primary interface and asset vault, designed to be lightweight and gas-efficient. Its responsibilities are strictly limited to asset locking, event emission, and state updates based on authorized callbacks. The second environment is the Solana Hub (SVM), which acts as the system's "Brain." It hosts the Gateway Contract for cryptographic verification and data formatting, and the Main Contract for executing the core business logic, such as managing the Universal Wallet and calculating dynamic Health Factors. The third environment, bridging the deterministic worlds of these blockchains, is the Off-chain Guardian Infrastructure. This middleware operates as an active listener and orchestrator, ensuring that state transitions on one chain are accurately and securely reflected on the other.

To visualize the interaction between these components and the directional flow of data, Figure 0.1 presents the detailed system architecture diagram.

The operational workflow of the system, as depicted in the diagram, follows a rigorous six-step process designed to ensure atomicity, security, and eventual consistency.

Step 1: Initiation and Request Encapsulation The process begins on the EVM chain where the user intends to perform an action, such as depositing collateral. The user constructs a message M containing the action parameters (e.g., Token Address, Amount, Target Chain ID) and a unique nonce. Crucially, the user signs this message with their EVM private key, generating a signature σ. The user then submits a transaction to the **Controller Contract**. Upon receipt, the Controller



Hình 0.1: Detailed Architecture of the Multi-chain Stablecoin Protocol

does not immediately execute the cross-chain logic but performs two critical local actions: it locks the user's assets (in a Vault) and emits a 'RequestCreated' event containing the message M and signature σ. This emission acts as a signal flare to the off-chain infrastructure.

Step 2: Event Ingestion (The Listening Phase) Unlike traditional polling mechanisms that can be resource-intensive, the Guardian Server employs a reactive Event Listening model. It maintains an active WebSocket or HTTP connection to the EVM RPC nodes. When the 'RequestCreated' event is emitted by the Controller, the Guardian immediately captures the log data. This step represents the "Event" in the Event-Driven Architecture. The Guardian parses the log to extract the raw data necessary for the state transition on the destination chain.

Step 3: Relaying and Submission Once the data is captured, the Guardian packages the original message M and the signature σ into a new transaction payload compatible with the Solana runtime. It then submits this transaction to the **Gateway Contract** on the SVM. In this phase, the Guardian acts strictly as a courier (Relayer); it does not modify the message content, ensuring that the user's original intent remains tamper-proof. The Guardian also manages the payment of SOL gas fees, abstracting the complexity of holding multiple native tokens from the end-user.

Step 4: Verification and Instruction Execution The **Gateway Contract** serves as the entry point to the Solana environment. Its primary responsibility is secu-

rity. Before any business logic is executed, the Gateway invokes Solana's native 'Secp256k1' program to cryptographically verify that the signature σ corresponds to the user's EVM address contained in message M. This verification is performed on-chain, providing a trustless guarantee of identity. Upon successful verification, the Gateway formats the data into a structured Instruction and calls the **Main Contract**. The Main Contract then executes the core logic, such as creating a Universal Wallet PDA or updating the user's debt position.

Step 5: Outcome Observation Similar to the ingestion phase on the EVM side, the Guardian Server also maintains a listener on the Solana Blockchain. When the Main Contract finishes execution, it emits a specific event - either 'ExecutionSuccess' (indicating the state was updated) or 'ExecutionFailure' (indicating a logic error, such as low health factor). The Guardian listens for these specific outcome events to determine the final status of the cross-chain operation. This asynchronous confirmation step is vital for handling the probabilistic finality of blockchain networks.

Step 6: State Synchronization and Finalization Based on the event received from the Solana Main Contract, the Guardian initiates a final callback transaction to the EVM **Controller Contract** to close the loop. If the Solana execution was successful, the Guardian calls the 'updateState' function to finalize the process (e.g., minting stablecoins to the user). If the Solana execution failed, the Guardian triggers a rollback mechanism, unlocking the user's assets and resetting their nonce. This ensures that the system maintains data consistency between the Asset Layer and the State Layer, preventing any funds from being permanently locked in transit.

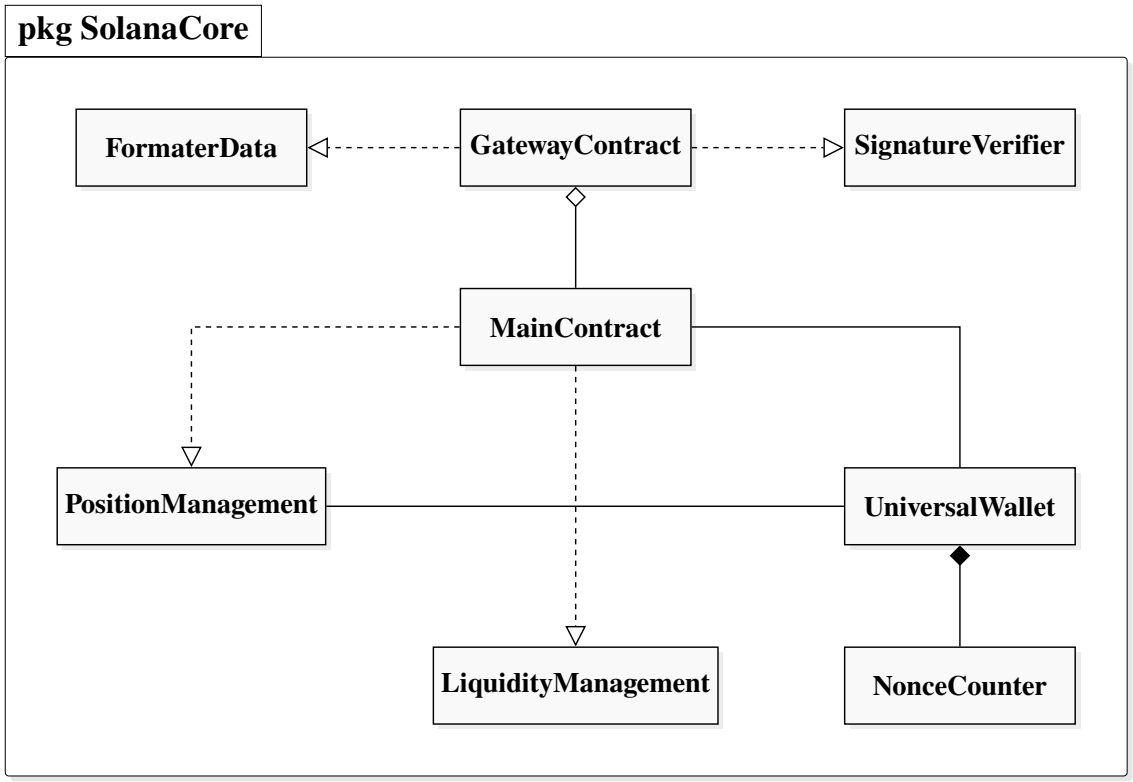
0.1.2 Detailed Package Design

Based on the overall architecture, this section details the internal design of the critical subsystems. The design is visualized using Class Diagrams grouped by their respective execution environments.

a, Solana Hub Package Design

The Solana Hub Package, designated as package SolanaCore, encapsulates the system's central business logic and state management. This package is architected to ensure strict separation between the interface layer (Gateway), the logic layer (Main Contract and Managers), and the data layer (Wallet and Counters). Figure 0.2 illustrates the internal structure and class relationships within this package.

Class Descriptions and Relationships:



Hình 0.2: Detailed Design of Solana Hub Package

- **GatewayContract:** This class acts as the single entry point for all cross-chain transactions initiated by the Guardian. It serves as an orchestrator that sanitizes inputs before passing control to the core logic.
  - Implementation Relationships: The Gateway implements logic from Formater Data class to deserialize incoming payloads and utilizes the Signature Verifier to perform cryptographic checks (Secp256k1 recovery) on the user's signature.
  - Aggregation: It maintains an aggregation relationship with the Main Contract, indicating that the Gateway coordinates the execution flow but delegates the actual financial state transitions to the Main Contract.
- **MainContract:** This is the core controller of the system. It manages the lifecycle of CDPs and coordinates liquidity across chains. To maintain modularity, it splits complex operations into specialized management modules:
  - It implements the Position Management module to handle collateral locking, debt minting, and health factor calculations.
  - It implements the Liquidity Management module to handle the rebalancing of assets between the Hub and Spokes during liquidations.
- **UniversalWallet:** This class represents the persistent state (Program Derived

Address - PDA) of a user. It stores the aggregated data of collateral and debt.

  - Association: Both the Main Contract and Position Management have direct associations with the Universal Wallet to read and modify the user's financial position.

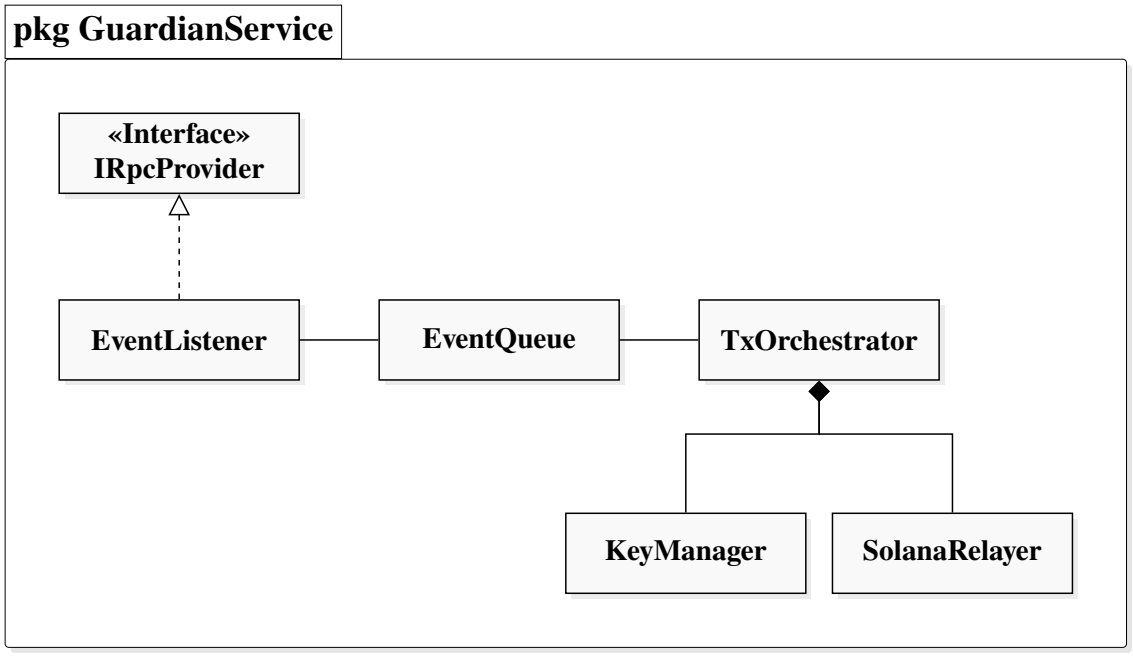
- **NonceCounter:** This class tracks the sequence number of transactions to prevent replay attacks.

  - Composition: The diagram defines a strict composition relationship (filled diamond) between Universal Wallet and Nonce Counter. This implies that the Nonce Counter is an intrinsic part of the Wallet; it cannot exist independently, and its lifecycle is bound to the existence of the Universal Wallet.

b, Guardian Middleware Package Design

The Guardian Middleware Package, designated as package GuardianService, acts as the off-chain orchestration layer. It is designed using an Event-Driven architecture to handle the asynchronous nature of cross-chain communication reliably.

Figure 0.3 depicts the internal class design of the Guardian node.



Hình 0.3: Detailed Design of Guardian Middleware Package

Class Descriptions and Relationships:

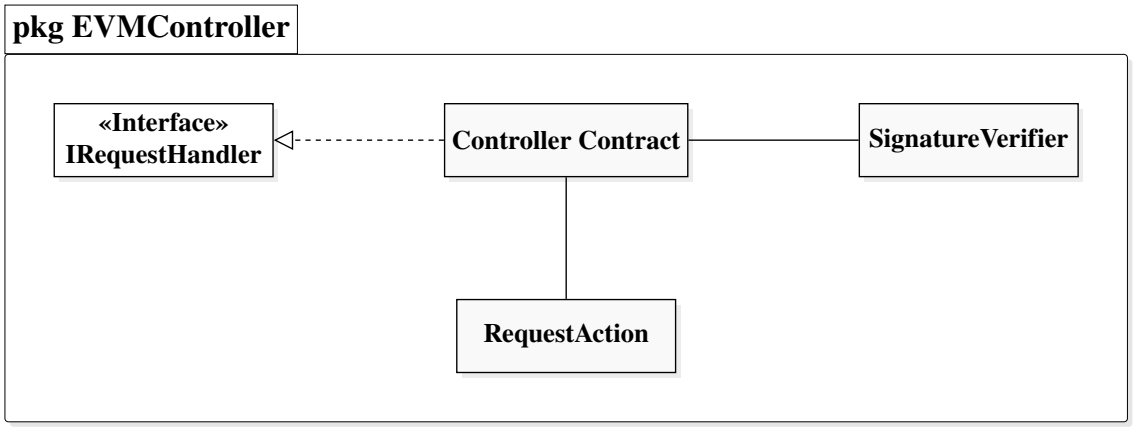
- **EventListener:** This component is responsible for monitoring blockchain networks. It implements the IRpcProvider interface to maintain agnostic connections to various EVM chains (Ethereum, BSC, Arbitrum). Its primary role is to detect RequestCreated logs and normalize them into a standard event format.

- **EventQueue:** Acting as a buffer, this class decouples the ingestion layer (Listener) from the processing layer (Orchestrator). It ensures that during high network traffic, events are not lost but queued for sequential processing.
- **TxOrchestrator:** This is the core logic unit of the middleware. It consumes events from the queue, validates the data integrity, and constructs the corresponding cross-chain transaction payloads.
- **KeyManager:** A security-critical class responsible for managing the Guardian's private keys.
  - Composition: The TxOrchestrator has a composition relationship with KeyManager, indicating that the orchestrator cannot function (cannot sign transactions) without the secure signing module.
- **SolanaRelayer:** This class handles the low-level networking required to broadcast signed transactions to the Solana cluster and confirm their finality.
  - Aggregation: The TxOrchestrator aggregates the SolanaRelayer, utilizing it as a service to dispatch the final outcome of its logic.

c, EVM Controller Package Design

The EVM Controller Package is designed to manage user interactions on the EVM chains, ensuring that requests are properly formatted, signed, and validated before being relayed to the Solana Hub. This package adheres to a layered design within the EVM environment, separating concerns related to request handling, data validation, and asset management.

Figure 0.4 illustrates the class structure within this package.



Hình 0.4: Detailed Design of EVM Controller Package

Design Explanation:

- **Controller Contract:** This is the core contract deployed on the EVM chain.

It manages user requests and orchestrates the cross-chain interaction flow.

  - Interface Implementation: The contract implements the IRequestHandler interface, defining the standard methods for processing external requests.
  - Usage Relationships: It has direct associations with RequestAction and SignatureVerifier classes. The Controller Contract utilizes RequestAction to structure and validate incoming request data and relies on SignatureVerifier to perform cryptographic checks on user signatures.

- **RequestAction:** A data structure class that encapsulates all the necessary parameters for a cross-chain request (e.g., nonce, action type, amount, destination chain ID). It is used by the Controller Contract to hold and process incoming requests.
- **SignatureVerifier:** This utility class is responsible for validating the authenticity of a user's signature. It takes the message M and the signature σ as input and verifies if they were generated by the owner of the intended EVM address. This is a critical security component that prevents unauthorized actions.

0.2 Detailed design

0.2.1 User interface design

Phần này có độ dài từ hai đến ba trang. Sinh viên đặc tả thông tin về màn hình mà ứng dụng của mình hướng tới, bao gồm độ phân giải màn hình, kích thước màn hình, số lượng màu sắc hỗ trợ, v.v. Tiếp đến, sinh viên đưa ra các thống nhất/chuẩn hóa của mình khi thiết kế giao diện như thiết kế nút, điều khiển, vị trí hiển thị thông điệp phản hồi, phối màu, v.v. Sau cùng sinh viên đưa ra một số hình ảnh minh họa thiết kế giao diện cho các chức năng quan trọng nhất. Lưu ý, sinh viên không nhầm lẫn giao diện thiết kế với giao diện của sản phẩm sau cùng.

0.2.2 Layer design

Phần này có độ dài từ ba đến bốn trang. Sinh viên trình bày thiết kế chi tiết các thuộc tính và phương thức cho một số lớp chủ đạo/quan trọng nhất của ứng dụng (từ 2-4 lớp). Thiết kế chi tiết cho các lớp khác, nếu muốn trình bày, sinh viên đưa vào phần phụ lục.

Để minh họa thiết kế lớp, sinh viên thiết kế luồng truyền thông điệp giữa các đối tượng tham gia cho 2 đến 3 use case quan trọng nào đó bằng biểu đồ trình tự (hoặc biểu đồ giao tiếp).

0.2.3 Database design

Phần này có độ dài từ hai đến bốn trang. Sinh viên thiết kế, vẽ và giải thích biểu đồ thực thể liên kết (E-R diagram). Từ đó, sinh viên thiết kế cơ sở dữ liệu tùy theo hệ quản trị cơ sở dữ liệu mà mình sử dụng (SQL, NoSQL, Firebase, v.v.)

0.3 Application Building

0.3.1 Libraries and Tools

Sinh viên liệt kê các công cụ, ngôn ngữ lập trình, API, thư viện, IDE, công cụ kiểm thử, v.v. mà mình sử dụng để phát triển ứng dụng. Mỗi công cụ phải được chỉ rõ phiên bản sử dụng. SV nên kẻ bảng mô tả tương tự như Bảng **??**. Nếu có nhiều nội dung trình bày, sinh viên cần xoay ngang bảng.

Mục đích	**Công cụ**	**Địa chỉ URL**
IDE lập trình	Eclipse Oxygen a64 bit	http://www.eclipse.org/
v.v.	v.v.	v.v.

Bảng 1: Danh sách thư viện và công cụ sử dụng

0.3.2 Achievement

Sinh viên trước tiên mô tả kết quả đạt được của mình là gì, ví dụ như các sản phẩm được đóng gói là gì, bao gồm những thành phần nào, ý nghĩa, vai trò?

Sinh viên cần thống kê các thông tin về ứng dụng của mình như: số dòng code, số lớp, số gói, dung lượng toàn bộ mã nguồn, dung lượng của từng sản phẩm đóng gói, v.v. Tương tự như phần liệt kê về công cụ sử dụng, sinh viên cũng nên dùng bảng để mô tả phần thông tin thống kê này.

0.3.3 Illustration of main functions

Sinh viên lựa chọn và đưa ra màn hình cho các chức năng chính, quan trọng, và thú vị nhất. Mỗi giao diện cần phải có lời giải thích ngắn gọn. Khi giải thích, sinh viên có thể kết hợp với các chú thích ở trong hình ảnh giao diện.

0.4 Testing

Phần này có độ dài từ hai đến ba trang. Sinh viên thiết kế các trường hợp kiểm thử cho hai đến ba chức năng quan trọng nhất. Sinh viên cần chỉ rõ các kỹ thuật kiểm thử đã sử dụng. Chi tiết các trường hợp kiểm thử khác, nếu muốn trình bày, sinh viên đưa vào phần phụ lục. Sinh viên sau cùng tổng kết về số lượng các trường hợp kiểm thử và kết quả kiểm thử. Sinh viên cần phân tích lý do nếu kết quả kiểm thử không đạt.

0.5 Deployment

Sinh viên trình bày mô hình và/hoặc cách thức triển khai thử nghiệm/thực tế. Ứng dụng của sinh viên được triển khai trên server/thiết bị gì, cấu hình như thế nào. Kết quả triển khai thử nghiệm nếu có (số lượng người dùng, số lượng truy cập, thời gian phản hồi, phản hồi người dùng, khả năng chịu tải, các thống kê, v.v.)