VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Dinh Minh Hai

A SUPPORT TOOL TO SPECIFY AND VERIFY TEMPORAL PROPERTIES IN OCL

BACHELOR'S THESIS

Major: Computer Science

HA NOI – 2025

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Dinh Minh Hai

A SUPPORT TOOL TO SPECIFY AND VERIFY TEMPORAL PROPERTIES IN OCL

BACHELOR'S THESIS
Major: Computer Science

Supervisor: Assoc. Prof. Dang Duc Hanh

HA NOI – 2025

ABSTRACT

Abstract:

Keywords:

DECLARATION

I hereby declare that I composed this thesis, ***"A Support Tool to Specify and Verify Temporal Properties in OCL"***, under the supervision of Assoc. Prof. Dang Duc Hanh. This work reflects my own effort and serious commitment to research. I have incorporated and adapted select open-source code and modeling resources to align with the research objectives, and all external materials used have been properly cited. I take full responsibility for the content and integrity of this thesis.

Ha Noi, 07th April 2025

Student

Dinh Minh Hai

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Assoc. Prof. Dang Duc Hanh, for his invaluable guidance and unwavering support throughout the research and writing of this thesis. His expertise and dedication have been instrumental in shaping this work.

I am also grateful to the alumni and current members of the research group for their insightful discussions and constructive feedback, which greatly enriched my research.

Furthermore, I extend my thanks to the faculty members of the University of Engineering and Technology for their passionate teaching and for equipping me with the essential knowledge and skills that form the foundation of this thesis.

Lastly, I offer my gratitude to my family for their constant care, support, and encouragement. Their belief in me provided the motivation and stability I needed to pursue and complete this thesis.

Although I have endeavored to conduct this research to the highest standard, I recognize that limitations in my knowledge and experience may have led to unintentional shortcomings. I sincerely welcome comments and suggestions from professors and peers to enhance this work further.

To all who have supported me on this journey, I am profoundly grateful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Glossary of Abbreviations and Terms

Acronyms	**Definition**
MDE	Model Driven Engineering
UML	Unified Modeling Language
OCL	Object Constraint Language
ERC20	Ethereum Request for Comment-20
DEX	Decentralized Exchange
SPL	Solana Program Library
SDK	Software development kit
DOM	Document Object Model

INTRODUCTION

Model-Driven Engineering (MDE) approaches system development by emphasizing models over source code. Through modeling languages such as UML and OCL, developers construct models that abstract the system complexity and serve as primary development artifacts. These models can then be transformed into executable code, documentation, and test cases using automated model transformation techniques. Techniques like validation and verification help ensure the models' quality, with structural aspects, represented by class and object diagrams, being particularly important.

The Unified Modeling Language (UML) offers a standard way to create diagrams that show a system's structure and behavior, while the Object Constraint Language (OCL) adds precise rules to these models. OCL uses simple logic to define conditions that must always be true or requirements for operations. However, it lacks the expressive power to model temporal properties that involve different states of the model or depend on event occurrences. This limitation stems from the absence of built-in constructs for handling time and events in OCL.

In this work, we aim to extend OCL to support temporal properties and events,

Để đạt được các mục tiêu trên, khóa luận sẽ áp dụng các phương pháp nghiên cứu sau:

Cấu trúc khóa luận gồm 4 phần như sau:

- **Chương 1**:
- **Chương 2**:
- **Chương 3**:

- **Kết luận**:

Chương 1
KIẾN THỨC NỀN TẢNG

1.1 Giới thiệu

Chương này vào việc nghiên cứu kiến thức cơ bản và toàn diện về công nghệ chuỗi khối, một công nghệ nền tảng cho dự án LaunchCrypt. Mục đích chính của chương này là giúp độc giả hiểu rõ về cơ chế hoạt động, đặc điểm, và các loại chuỗi khối khác nhau, cũng như các kiến thức cơ bản về hợp đồng thông minh. Kiến thức này sẽ tạo cơ sở vững chắc để hiểu sâu hơn về các khái niệm phức tạp hơn trong lĩnh vực tài chính phi tập trung (DeFi) và các ứng dụng chuỗi khối khác được đề cập trong các phần tiếp theo của khóa luận. Đồng thời, phần này cũng giúp làm rõ lý do tại sao chuỗi khối là công nghệ then chốt cho việc phát triển các ứng dụng phi tập trung như LaunchCrypt. Chương sẽ tập trung vào năm mục: **Mục 1.2** trình bày tổng quan về công nghệ chuỗi khối, **Mục 1.3** trình bày tổng quan kiến thức về tài chính phi tập trung (DeFi) , **Mục 1.4** trình bày tổng quan về token và kiến trúc của token trên các chuỗi khối khác nhau, **Mục 1.5** trình bày tổng quan về thị trường tạo lập tự động (AMM) - phương thức giao dịch chính được sử dụng trong sản phẩm, **Mục 1.6** trình bày các thư viện và các ngôn ngữ lập trình được sử dụng trong khóa luận.

1.2 Tổng quan về công nghệ chuỗi khối

1.2.1 Công nghệ chuỗi khối

Công nghệ chuỗi khối là một cơ chế lưu trữ dữ liệu phân tán dựa trên các khối thông tin được liên kết với nhau bằng phương thức mã hóa thông

tin. Các dữ liệu được lưu trên chuỗi có tính đồng bộ và nhất quán theo thời gian. Hệ thống này dựa vào các giao thức đồng thuận để thay đổi thông tin bằng cách thêm các khối mới vào trong chuỗi. Hiện tại có rất nhiều giao thức đồng thuận đang được áp dụng từ việc tận dụng độ khó của các thuật toán mã hóa đến đếm sự đồng thuận của các máy xác nhận trong hệ thống, tất cả đều tập trung vào mục đích đảm bảo tính minh bạch và bất biến của các thông tin đang được lưu trữ.

Bên cạnh đó, công nghệ chuỗi khối nhắm tới việc xây dựng một giao thức trao đổi thông tin phi tập trung khi mà không có bất cứ một thực thể duy nhất nào có quyền quyết định tới các thay đổi trong một chuỗi, cũng như dữ liệu sẽ được phân phối về các nút trong mạng để đảm bảo yêu cầu về truy xuất và xác thực. Bằng việc áp dụng công nghệ chuỗi khối, ta có thể giải quyết được vấn đề xác thực giao dịch giữa những cá nhân hay tổ chức với nhau. Hiện tại, với các giao dịch thông thường, chúng ta cần phải có một bên trung gian mà các bên tham gia cùng tin cậy để làm chứng. Việc này có thể dẫn tới các rủi ro về bảo mật khi dữ liệu của bên thứ ba bị thay đổi một cách không mong muốn. Hay tồi tệ hơn là chính những bên trung gian xác thực này bắt tay với một vài cá nhân hay tổ chức nhằm thao túng giao dịch, gây bất lợi cho các bên tham gia còn lại. Các lợi ích chính của việc áp dụng công nghệ chuỗi khối có thể kể tới như cung cấp tính bảo mật, giảm thiểu chi phí cho các giao dịch số. Bằng cơ chế phi tập trung và đồng thuận thì việc giả mạo và thao túng các giao dịch này gần như là bất khả thi. Có thể thấy rằng, công nghệ chuỗi khối trong thời gian gần đây đang được áp dụng mạnh mẽ trong các giao dịch tài chính. Nhưng, sự áp dụng của công nghệ này không chỉ dừng ở đó mà nó còn có thể được ứng dụng trong nhiều ngành nghề khác như xác minh bản quyền, xác minh tính tin cậy nguồn gốc của sản phẩm, vv.

1.2.2 Các mô hình chuỗi khối

Trong bối cảnh phát triển nhanh chóng của công nghệ chuỗi khối, việc hiểu rõ về các mô hình chuỗi khối khác nhau là cần thiết để đánh giá và lựa chọn giải pháp phù hợp cho các ứng dụng tài chính phi tập trung. Ba mô hình chính của công nghệ chuỗi khối: công khai, riêng tư và liên hợp - mỗi mô hình đều có những đặc điểm, ưu điểm và hạn chế riêng, phản ánh sự đa dạng trong cách tiếp cận với công nghệ sổ cái phân tán này.

Chuỗi khối công khai, như Bitcoin và Ethereum, là những mạng lưới mở và phi tập trung hoàn toàn. Chúng cho phép bất kỳ ai cũng có thể tham gia vào quá trình xác thực giao dịch và duy trì mạng lưới. Tính minh bạch và khả năng chống can thiệp cao là những ưu điểm nổi bật của mô hình này. Tuy nhiên, chuỗi khối công khai thường phải đối mặt với những thách thức về khả năng mở rộng và hiệu suất, đặc biệt là khi số lượng giao dịch tăng cao. Cơ chế đồng thuận như Proof of Work (PoW) trong Bitcoin, mặc dù đảm bảo an ninh mạng, nhưng lại tiêu tốn nhiều năng lượng.

Ngược lại, chuỗi khối riêng tư được thiết kế để phục vụ cho nhu cầu cụ thể của một tổ chức hoặc nhóm tổ chức. Trong mô hình này, quyền tham gia và xác thực giao dịch được kiểm soát chặt chẽ. Hyperledger Fabric là một ví dụ điển hình cho loại chuỗi khối này. Ưu điểm của chuỗi khối riêng tư là khả năng xử lý giao dịch nhanh hơn và hiệu quả năng lượng cao hơn. Tuy nhiên, tính phi tập trung và minh bạch - những đặc tính cốt lõi của công nghệ chuỗi khối - có thể bị giảm đi trong mô hình này.

Đứng giữa hai mô hình trên là mô hình chuỗi khối liên hợp, một giải pháp trung gian kết hợp ưu điểm của cả chuỗi khối công khai và riêng tư. Trong mô hình này, một nhóm các tổ chức cùng quản lý mạng lưới, tạo ra một hệ thống bán tập trung. R3 Corda trong lĩnh vực tài chính là một ví dụ tiêu biểu. Chuỗi khối liên hợp cung cấp một sự cân bằng giữa tính minh bạch và khả năng kiểm soát, đồng thời duy trì hiệu suất cao hơn so với chuỗi

khối công khai. Tuy nhiên, mô hình này cũng đối mặt với thách thức trong việc điều phối giữa các thành viên và giải quyết xung đột lợi ích.

Trong bối cảnh của dự án LaunchCrypt, sau khi cân nhắc kỹ lưỡng các mô hình chuỗi khối hiện có, khóa luận quyết định lựa chọn mô hình chuỗi khối công khai làm nền tảng cho hệ thống. Quyết định này dựa trên nhiều yếu tố quan trọng phù hợp với mục tiêu và tầm nhìn của dự án. Thứ nhất, tính minh bạch và phi tập trung của chuỗi khối công khai hoàn toàn phù hợp với tinh thần cốt lõi của tài chính phi tập trung (DeFi), tạo ra một môi trường mở và công bằng cho tất cả người dùng. Điều này đặc biệt quan trọng đối với LaunchCrypt, một nền tảng tập trung vào việc tiếp cận với mọi người dùng. Thứ hai, khả năng chống can thiệp và tính bất biến của dữ liệu trong chuỗi khối công khai sẽ đảm bảo tính toàn vẹn của các giao dịch và thông tin token, tạo niềm tin cho người dùng - một yếu tố then chốt trong việc xây dựng một hệ sinh thái tài chính mạnh mẽ. Cuối cùng, việc chọn chuỗi khối công khai cũng phù hợp với mục tiêu hỗ trợ đa chuỗi của LaunchCrypt, cho phép tích hợp dễ dàng với các chuỗi khối công khai lớn như Avalanche, Solana và Aptos. Điều này mở rộng phạm vi tiếp cận của nền tảng và tăng tính linh hoạt cho người dùng.

1.2.3 Hợp đồng thông minh

Ethereum là một mạng phân tán được xây dựng dựa trên công nghệ chuỗi khối bởi Vitalik Buterin vào năm 2014 với mục tiêu xây dựng các ứng dụng phân quyền phi tập trung. Mạng Ethereum ngoài việc làm một cơ sở dữ liệu, nó còn được sử dụng như một máy tính khổng lồ mà mọi người đều có thể truy cập bằng cách lập trình và tương tác với các hợp đồng thông minh. Hợp đồng truyền thống là tập hợp các điều khoản của một mối quan hệ và các ràng buộc này được đảm bảo bởi sự đồng thuận của các bên tham gia và được bảo trợ bởi một bên trung gian có quyền lực cao hơn (thường là

các chính phủ). Còn hợp đồng thông minh được đảm bảo bằng các đoạn mã, chúng có thể coi là các chương trình chạy được trên chuỗi khối. Các giao dịch dựa trên hợp đồng thông minh bắt buộc phải tuân thủ theo các điều khoản đã được lập trình sẵn để có thể được thực thi. Cách thức hoạt động của hợp đồng thông minh là các đoạn mã sẽ được biên dịch và ký bởi khóa của người tạo ra chúng, sau đó là triển khai trên chuỗi khối công khai. Một khi đã triển khai xong, bất cứ ai tham gia mạng lưới đều có thể sử dụng chương trình này và tạo ra các giao dịch nếu chúng phù hợp với các điều kiện đã đặt ra. Tuy nhiên, các hợp đồng thông minh sẽ tồn tại bất biến cho nên nếu có bất kỳ sai sót gì thì chỉ có thể tạo ra một hợp đồng mới thay cho cái cũ. Lợi ích của việc sử dụng hợp đồng thông minh sẽ không yêu cầu xác minh danh tính của các bên tham gia hay một bên trung gian đứng ra làm bên đại diện. Điều này giúp tăng tốc độ và giảm chi phí giao dịch, cũng như xây dựng một môi trường giao dịch không yêu cầu sự tin tưởng lẫn nhau. Ngoài ra, hợp đồng thông minh còn có tính tùy chỉnh cao bởi vì chúng là những đoạn mã và có thể triển khai dễ dàng bởi bất kỳ cá nhân nào tham gia mạng lưới, cung cấp đa dạng các cách thức và giải pháp cho nhu cầu của người sử dụng.

Trong bối cảnh của dự án LaunchCrypt, hợp đồng thông minh đóng vai trò then chốt trong việc thực hiện mục tiêu phi tập trung quá trình tạo và giao dịch token. LaunchCrypt sẽ triển khai một hệ thống hợp đồng thông minh đa chức năng trên các chuỗi khối công khai như Avalanche, Solana và Aptos, nhằm tự động hóa và đảm bảo tính minh bạch cho toàn bộ quy trình. Đầu tiên, hợp đồng thông minh sẽ được sử dụng để tạo ra các token mới một cách dễ dàng và an toàn, cho phép người dùng không có kiến thức chuyên sâu về lập trình vẫn có thể định nghĩa các tham số cơ bản như tên token, ký hiệu và logo của token. Tiếp theo, LaunchCrypt sẽ triển khai các hợp đồn thông minh quản lý thanh khoản, tự động tạo và duy trì các cặp giao dịch cho các token mới, đảm bảo tính linh hoạt và hiệu quả trong quá trình giao dịch. Cuối cùng, hợp đồng thông minh cũng được sử dùng để quản lý quá

trình "tốt nghiệp"của token, tự động chuyển token lên các sàn giao dịch phi tập trung lớn hơn khi đạt đủ điều kiện về thanh khoản và khối lượng giao dịch. Bằng cách tận dụng tính bất biến và minh bạch của hợp đồng thông minh, LaunchCrypt không chỉ đảm bảo tính công bằng và an toàn cho người dùng, mà còn tạo ra một hệ sinh thái tự động và hiệu quả, giúp tăng niềm tin của người dùng vào nền tảng.

1.3 Tổng quan về tài chính phi tập trung (DeFi)

Là một dự án tập trung vào việc giải quyết các vấn đề trong lĩnh vực tài chính phi tập trung, việc hiểu và nắm bắt được bản chất của tài chính phi tập trung là điều vô cùng cần thiết và quan trọng. Nó không chỉ giúp định hướng phương thức phát triển của LaunchCrypt mà còn đảm bảo rằng các giải pháp được đề xuất sẽ phù hợp và hiệu quả trong bối cảnh DeFi đang không ngừng vươn lên. Phần này sẽ làm rõ cách DeFi đang thay đổi cảnh quan tài chính truyền thống, cũng như các cơ hội và thách thức mà nó mang lại. Đặc biệt, chúng ta sẽ tập trung vào vai trò của DeFi trong việc tạo ra các thị trường tài chính mở, minh bạch và không cần trung gian.

1.3.1 Giới thiệu về tài chính phi tập trung

Tài chính phi tập trung (DeFi) đã nổi lên như một cuộc cách mạng trong lĩnh vực tài chính, đánh dấu sự chuyển đổi từ hệ thống tài chính truyền thống tập trung (CeFi) sang một mô hình mới, phi tập trung và mở. DeFi có thể được định nghĩa là một hệ sinh thái các ứng dụng tài chính được xây dựng trên nền tảng chuỗi khối, chủ yếu là Ethereum, nhằm tái tạo và mở rộng các dịch vụ tài chính truyền thống mà không cần đến trung gian. Nguyên tắc cốt lõi của DeFi bao gồm tính minh bạch, khả năng tương tác, và không cần sự tin cậy (trustless), được thực hiện thông qua việc sử dụng các hợp đồng thông minh và công nghệ chuỗi khối. Điều này cho phép tạo ra các ứng dụng

tài chính tự động, mở và có thể kiểm chứng, từ đó giảm thiểu sự can thiệp của con người và tăng cường tính công bằng trong hệ thống tài chính.

1.3.2 Giới thiệu về sàn giao dịch phi tập trung (DEX)

Sàn giao dịch phi tập trung (Decentralized Exchanges - DEX) đã nổi lên như một trong những ứng dụng quan trọng nhất của DeFi, cung cấp một phương thức giao dịch tài sản kỹ thuật số mà không cần đến trung gian. Khác với sàn giao dịch tập trung truyền thống, DEX hoạt động dựa trên hợp đồng thông minh trên các nền tảng chuỗi khối, cho phép người dùng duy trì quyền kiểm soát tài sản của họ trong suốt quá trình giao dịch. Cơ chế hoạt động cơ bản của DEX dựa trên việc kết nối trực tiếp giữa người mua và người bán thông qua các giao thức được mã hóa, loại bỏ nhu cầu về một bên trung gian đáng tin cậy. So với sàn giao dịch tập trung, DEX mang lại nhiều ưu điểm đáng kể. Thứ nhất, DEX cung cấp tính minh bạch cao hơn, với mọi giao dịch đều được ghi lại trên chuỗi khối công khai. Thứ hai, người dùng duy trì quyền kiểm soát đối với khóa riêng tư, tức là quyền quyết soát đối với chính tài sản của họ, từ đó giảm thiểu rủi ro mất mát do hack hoặc lỗi của sàn giao dịch. Cuối cùng, DEX thường cung cấp danh sách token đa dạng hơn, bao gồm cả các token mới và ít phổ biến. Tuy nhiên, DEX cũng phải đối mặt với những thách thức như khả năng mở rộng hạn chế, chi phí gas cao (đặc biệt trên mạng Ethereum), và trải nghiệm người dùng có thể phức tạp hơn đối với người mới.

1.4 Tổng quan về token và kiến trúc của token trên các chuỗi khối khác nhau

Để đề xuất giải pháp về việc tạo và giao dịch token trên LaunchCrypt, việc hiểu rõ bản chất và cấu trúc của token trên các nền tảng chuỗi khối khác nhau là điều cần phải nắm rõ. Phần này tập trung về việc nghiên cứu

các kiến thức nền tảng về token trên các chuỗi khối khác nhau, chỉ rõ cấu trúc của các loại token, từ đó làm tiền đề khi triển khai module tạo token và giao dịch trong suốt khóa luận.

1.4.1 Tổng quan về token

Token trong bối cảnh công nghệ chuỗi khối là một dạng tài sản kỹ thuật số được tạo ra và quản lý trên một nền tảng chuỗi khối. Về bản chất, token là một đại diện kỹ thuật số cho một quyền lợi hoặc giá trị cụ thể, có thể là tài sản, tiện ích, hoặc quyền biểu quyết trong một hệ thống. Khác với cryptocurrency gốc của một chuỗi (như Bitcoin trên mạng Bitcoin hoặc Ether trên Ethereum), token thường được tạo ra và quản lý thông qua hợp đồng thông minh trên một chuỗi khối hiện có. Các đặc điểm cơ bản của token bao gồm khả năng chuyển nhượng, tính chia nhỏ, giá trị đại diện, và tính xác thực. Token có thể được chuyển từ một địa chỉ này sang địa chỉ khác một cách dễ dàng và nhanh chóng. Nhiều loại token có thể được chia thành các đơn vị nhỏ hơn, cho phép giao dịch với số lượng linh hoạt. Mỗi token đại diện cho một giá trị cụ thể trong hệ thống mà nó hoạt động, và mọi giao dịch liên quan đến token đều được xác thực và ghi lại trên chuỗi khối, đảm bảo tính minh bạch và an toàn.

1.4.2 Kiến trúc token trên Ethereum

ERC-20 (Ethereum Request for Comment 20) [**ERC20**] là tiêu chuẩn token phổ biến nhất trên chuỗi khối Ethereum. ERC-20 đã trở thành nền tảng cho hầu hết các fungible token trên Ethereum. Tiêu chuẩn này định nghĩa một tập hợp các quy tắc mà một token trên Ethereum phải tuân theo, đảm bảo tính tương thích và nhất quán trong hệ sinh thái Ethereum. ERC-20 định nghĩa một interface bao gồm 9 hàm bắt buộc và 2 sự kiện. Cấu trúc cơ bản như sau:

ERC20: *Fungible token standard function in Ethereum*

```
1  function name() public view returns (string)
2  function symbol() public view returns (string)
3  function decimals() public view returns (uint8)
4  function totalSupply() public view returns (uint256)
5  function balanceOf(address owner) public view returns
   (uint256 balance)
6  function transfer(address to, uint256 value) public returns
   (bool success)
7  function transferFrom(address from, address to, uint256 value)
   public returns (bool success)
8  function approve(address spender, uint256 value) public
   returns
   (bool success)
9  function allowance(address owner, address spender) public
   view
   returns (uint256 remaining)
```

ERC20: *Fungible token standard event in Ethereum*

```
1  event Transfer(address indexed from, address indexed to,
   uint256 value)
2  event Approval(address indexed owner, address indexed
   spender, uint256 value)
```

Mỗi hàm và sự kiện trong cấu trúc này có một vai trò cụ thể:

- **name**: Trả về tên của token được hiển thị trên chuỗi (Ví dụ: Ethers).
- **symbol**: Trả về ký hiệu rút gọn của token được hiển thị trên chuỗi (Ví dụ: ETH).

- **decimals**: Trả về số chữ số thập phân được sử dụng để biểu diễn một đơn vị nhỏ nhất của token (Ví dụ: 18).
- **totalSupply**: Trả về tổng cung hiện tại của token trên toàn bộ chuỗi.
- **balanceOf**: Trả về số dư token của một địa chỉ cụ thể.
- **transfer**: Thực hiện chuyển một số lượng token từ địa chỉ người gửi đến một địa chỉ khác.
- **transferFrom**: Thực hiện chuyển token từ một địa chỉ này sang địa chỉ khác, sử dụng cơ chế allowance.
- **allowance**: Kiểm tra số lượng token mà một địa chỉ được phép chi tiêu thay cho chủ sở hữu.
- **approve**: Cho phép một địa chỉ chi tiêu một số lượng token thay mặt chủ sở hữu.

Sự kiện **Transfer** và **Approval** được phát ra khi có sự chuyển token hoặc phê duyệt chi tiêu, giúp các ứng dụng bên ngoài theo dõi hoạt động của token.

Ngoài ra, để kiểm soát nguồn cung và kiểm soát lạm phát, các token tuân theo tiêu chuẩn ERC20 thường xây dựng các hàm nội bộ sau:

ERC20: *Fungible token standard internal function in Ethereum*	
1	function **mint**(address **account**, uint256 **value**) internal
2	function **burn**(address **account**, uint256 **value**) internal

Trong đó, hàm **mint** thường được sử dụng để đúc 1 lượng token ban đầu cho người khởi tạo token, **burn** là hàm được sử dụng để "đốt"một lượng token ở một địa chỉ cụ thể, giúp hạn chế lạm phát và điều chỉnh tổng cung của token theo xu hướng giảm đi.

1.4.3 Kiến trúc token trên Solana

Solana, một chuỗi khối hiệu suất cao, có cách tiếp cận khác biệt so với Ethereum trong việc triển khai và quản lý token. Thay vì sử dụng smart contracts như Ethereum, Solana sử dụng một chương trình có sẵn gọi là Token Program để quản lý việc tạo và giao dịch token. Phần này sẽ đề cập đến những kiến thức cơ bản về cách token được triển khai trên Solana. Chúng được gọi là SPL (Solana program library) [**SPL**]. Dưới đây là 3 thành phần cơ bản giúp cho token hoạt động được trên Solana.

- **Token Program**: Chứa tất cả logic để tương tác với các token trên Solana (kể cả fungible và non-fungible token).
- **Mint Account**: Đại diện cho một loại token cụ thể và lưu trữ metadata toàn cầu về token, chẳng hạn như tổng nguồn cung và quyền đúc (địa chỉ được ủy quyền để tạo token mới).
- **Token Account**: Tài khoản đại diện cho sự sở hữu 1 loại token ứng với 1 user.

Token Program chứa một số chức năng chính, bao gồm:

- **InitializeMint**: Tạo ra 1 **Mint Account** để đại diện cho một token mới trên Solana
- **InitializeAccount**: Tạo ra 1 **Token Account** mới.
- **MintTo**: Tạo ra một lượng mới của một loại token cụ thể và thêm chúng vào một token account. Điều này làm tăng nguồn cung token và chỉ có thể được thực hiện bởi địa chỉ có thẩm quyền.
- **Transfer**: Chuyển một lượng token từ token account này sang token account khác.

Token trên Solana được xác định duy nhất bằng địa chỉ của Mint Account do Token Program sở hữu. Tài khoản này là một bộ đếm toàn cầu cho một loại token cụ thể và lưu trữ dữ liệu như:

- **Supply**: Tổng lượng cung hiện tại của token.
- **Decimals**: Số chữ số thập phân được sử dụng để biểu diễn một đơn vị nhỏ nhất của token.
- **Mint authority**: Địa chỉ được cấp quyền cho việc tạo ra token mới, điều này làm tăng tổng cung của token.
- **Freeze authority**: Địa chỉ được cấp quyền cho việc đóng băng token (dừng các hoạt động chuyển token qua lại giữa các token account).

Sau đây là thông tin lưu trữ của Mint Account được triển khai trong Solana:

Solana Mint Account

```
1 pub struct Mint {
2     pub mint_authority: COption<Pubkey>
3     pub supply: u64
4     pub decimals: u8
5     pub is_initialized: bool
6     pub freeze_authority: COption<Pubkey>
7 }
```

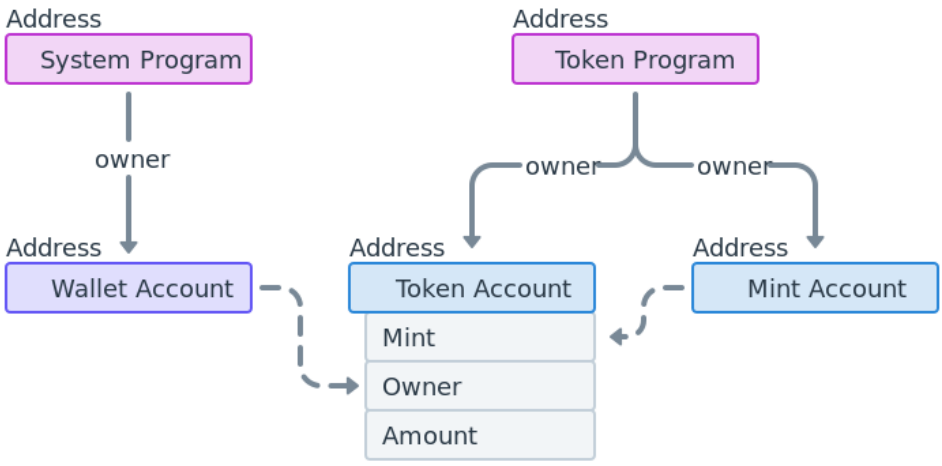
Để theo dõi quyền sở hữu riêng lẻ của từng đơn vị token cụ thể, Token Program phải tạo ra một loại tài khoản dữ liệu khác. Trong cấu trúc token của Solana, loại tài khoản này được gọi là Token Account. Dữ liệu được lưu trữ trên Token Account bao gồm:

- **Mint**: Loại token mà Token Account này đang nắm giữ.

Hình 1.1: Cấu trúc của Mint Account trên solana. [**SPL**]

- **Owner**: Tài khoản được ủy quyền để thực hiện các giao dịch trên Token Account.
- **Amount**: Số lượng token mà Token Account này đang nắm giữ.

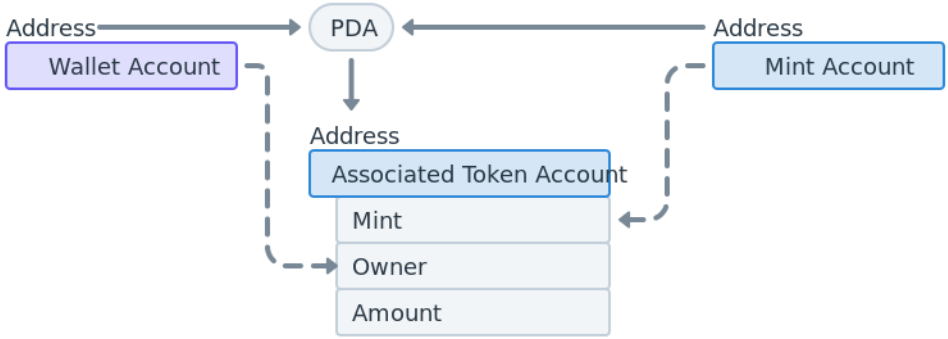
Để một ví sở hữu đơn vị của một token nhất định, nó cần tạo một Token Account cho loại Token (Mint Account) tương ứng và chỉ định ví đó là chủ sở hữu của Token Account. Một ví có thể tạo nhiều Token Account cho cùng một loại token, nhưng mỗi Token Account chỉ có thể được sở hữu bởi một ví và nắm giữ các đơn vị của một loại token.



Hình 1.2: Account relationship trên solana. [**SPL**]

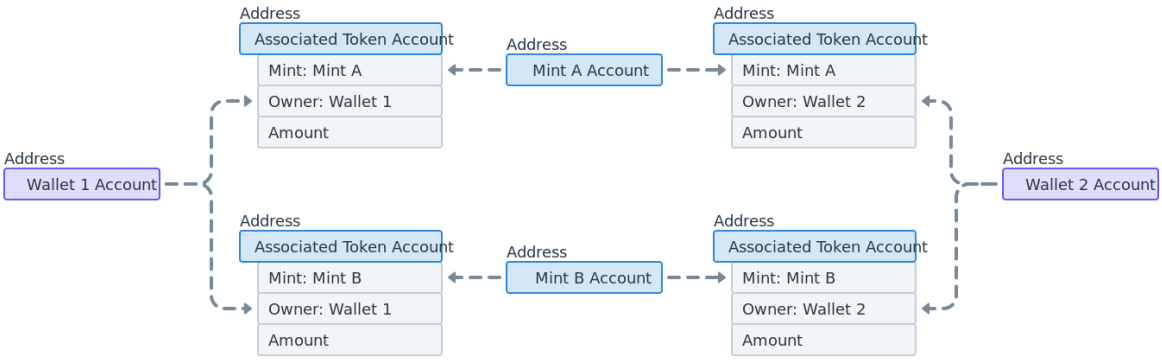
Tuy nhiên, để đơn giản hóa quá trình định vị địa chỉ Token Account cho một owner và một Mint Account cụ thể, Solana đã đưa ra giải pháp sử dụng Associated Token Accounts.

Associated Token Accounts là Token Account có địa chỉ được xác định rõ ràng bằng cách sử dụng địa chỉ của owner và địa chỉ của Mint Account. Có thể coi rằng Associated Token Accounts là Token Account "mặc định"cho một owner và một Mint Account cụ thể. Điều quan trọng phải hiểu rằng Associated Token Accounts không phải là một Token Account khác, đó chỉ là một loại Token Account có địa chỉ cụ thể được xác định từ trước.



Hình 1.3: Associated Token Account trên solana. [**SPL**]

Cuối cùng, ta có quan hệ giữa Wallet account, Associated Token Account và mint account như sau:



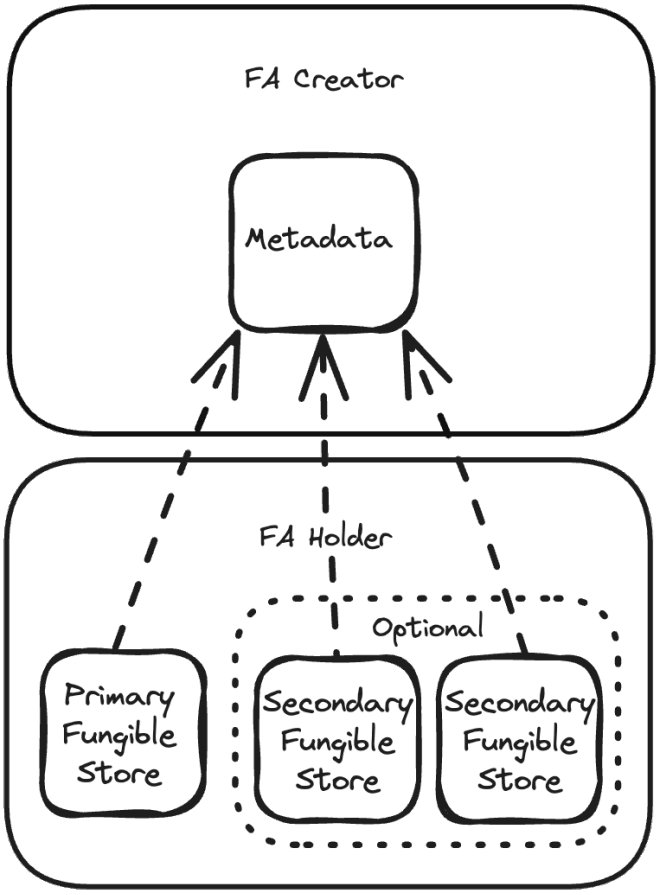
Hình 1.4: Quan hệ giữa các loại tài khoản trên solana. [**SPL**]

1.4.4 Kiến trúc token trên Aptos

Tiêu chuẩn token trên Aptos [**FA**] (còn được gọi là "**Fungible Asset**" hoặc "**FA**") cung cấp một cách tiêu chuẩn, an toàn về type để xác định token trong hệ sinh thái Move. Tiêu chuẩn này cho phép chuyển và tùy chỉnh các token cho mọi trường hợp sử dụng. Trong Aptos, với tiêu chuẩn "**Fungible Asset**", token được biểu diễn dưới dạng một object chứa data. Tiêu chuẩn này cung cấp các chức năng như tạo, đúc, chuyển và đốt (burn) các token. Để thực hiện điều này, tiêu chuẩn "**Fungible Asset**" sử dụng hai Move object như sau:

- **Object<Metadata>**: Đối tượng này thể hiện thông tin chi tiết về chính Fungible Asset, bao gồm thông tin như name, symbol và decimals.
- **Object<Fungible Store>**: Đối tượng này lưu trữ số lượng đơn vị Fungible Asset được sở hữu bởi tài khoản tương ứng. Fungible Asset có thể hoán đổi với bất kỳ Fungible Asset có cùng siêu dữ liệu (metadata). Một tài khoản có thể sở hữu nhiều FungibleStore cho một Fungible Asset duy nhất, nhưng điều này chỉ áp dụng cho các trường hợp sử dụng nâng cao.

Sơ đồ bên dưới thể hiện mối quan hệ giữa các object này. Đối tượng **metadata** được sở hữu bởi người tạo **FA**, sau đó được tham chiếu trong **FungibleStores** của chủ sở hữu **FA** để cho biết **FA** nào đang được theo dõi:



Hình 1.5: Mối quan hệ giữa Metadata và Fungible Store trong Aptos. [**FA**]

Như vậy, khi người dùng muốn chuyển **Fungible Asset** từ địa chỉ này sang địa chỉ khác, về bản chất, Aptos sẽ cập nhật lại thông tin về số lượng **Fungible Asset** còn lại trong các **Fungible Store** ứng với mỗi người dùng. Về các chức năng còn lại của token, Aptos cung cấp các **reference** (ref) được tạo ngay khi **Fungible Asset** được tạo ra. Một số ref phổ biến được sử dụng trong tiêu chuẩn **Fungible Asset** như sau:

- **ConstructorRef**: Tham chiếu cho phép FA được tùy chỉnh ngay sau khi chúng được tạo ra. Thông thường, constructorRef được sử dụng để tạo ra các ref khác.
- **MintRef**: Cung cấp khả năng tạo ra các đơn vị FA mới.

- **TransferRef**: Cung cấp khả năng đóng băng các tài khoản khỏi việc chuyển FA hoặc bỏ qua việc đóng băng hiện có. (Điều này có thể quan trọng khi cố gắng tuân thủ một số quy định).
- **BurnRef**: Cung cấp khả năng xóa các đơn vị FA.

Lưu ý: Tất cả các tham chiếu phải được tạo khi đối tượng được tạo vì đó là lần duy nhất có thể tạo ConstructorRef của đối tượng.

1.5 Automated Market Maker (AMM)

Một trong những đổi mới quan trọng nhất trong lĩnh vực DEX là sự ra đời của Automated Market Maker (AMM). AMM là một loại DEX đặc biệt sử dụng các thuật toán để tự động định giá và cung cấp thanh khoản cho các cặp giao dịch. Thay vì dựa vào sổ lệnh truyền thống (Order book), AMM sử dụng các pool thanh khoản, nơi người dùng có thể đóng góp tài sản và nhận lại token thanh khoản đại diện cho phần đóng góp của họ. Công thức phổ biến nhất cho AMM là **X*Y=K (1)**, trong đó X và Y là số lượng của hai tài sản trong pool, và k là một hằng số.

1.5.1 Công thức trao đổi token trên AMM

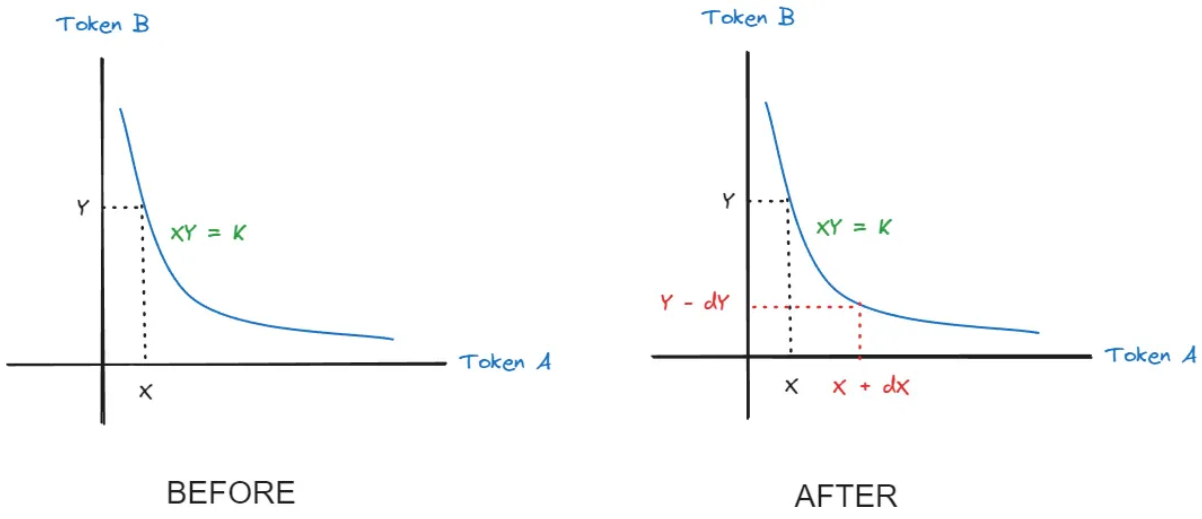
Khi một giao dịch xảy ra, số lượng còn lại của 2 token (từ nay gọi là **tokenA** và **tokenB**) trong pool sẽ thay đổi. Tuy nhiên, tích số lượng còn lại của 2 token trong pool sẽ không đổi, luôn luôn bằng k. Bây giờ, chúng ta cùng xét một ví dụ, giả sử ta sẽ bán một lượng tokenA để nhận lại 1 lượng tokenB. Ta có:

$$dX = \text{Số lượng tokenA bán}$$

$$dY = \text{Số lượng tokenB nhận}$$

Sau quá trình này, theo công thức X*Y=K, ta có:

$$(X + dX)(Y - dY) = K$$



Hình 1.6: Trạng thái của AMM trước và sau 1 lệnh bán dX. [**AMM**]

Từ đây, ta có công thức tính số lượng token nhận được (**dY**) dựa vào số lượng token bán (**dX**) như sau:

$$XY = K$$

$$(X + dX)(Y - dY) = K$$

$$Y - dY = \frac{K}{X + dX}$$

$$dY = Y - \frac{XY}{X + dX}$$

$$dY = \frac{YX + YdX - XY}{X + dX}$$

$$dY = \frac{YdX}{X + dX} \quad \textbf{(2)}$$

Như vậy, công thức (**2**) là công thức để tính lượng token **dY** nhận được khi ta muốn trao đổi một lượng token **dX**

1.5.2 Công thức tính lượng token share khi cung cấp thanh khoản

Khi một người dùng muốn đóng góp thanh khoản vào trong pool, giả sử họ muốn thêm vào một lương là **dX** và **dY**. Lúc này, để duy trì tỷ giá giữa hai token ở trong pool, ta có công thức ràng buộc giữa **dX** và **dY** như sau:

$$XY = K$$

$$(X + dX)(Y + dY) = K'$$

$$\frac{X}{Y} = \frac{X + dX}{Y + dY}$$

$$XdY = YdX$$

$$\frac{X}{Y} = \frac{dX}{dY}$$

$$dY = \frac{YdX}{X} \quad \textbf{(3)}$$

Lúc này, AMM sẽ trả về cho người dùng một lượng token "share". Khi người dùng muốn rút thanh khoản ra khỏi pool, họ có thể sử dụng lượng token "share" trước đó để thực hiện hành động này. Ta định nghĩa giá trị tổng thanh khoản trong pool bằng một hàm như sau:

$$f(X, Y) = \sqrt{(XY)}$$

Với định nghĩa này, ta có:

$$L0 = f(X, Y)$$

$$L1 = f(X + dX, Y + dY)$$

Ta định nghĩa T là tổng lượng token "share" đã được tạo ra, s là lượng token

"share" mới sẽ được đúc cho người dùng. Lúc này, theo quy tắc tổng số lượng token "share" sẽ tăng tỷ lệ thuận với sự gia tăng thanh khoản, ta có công thức tính số lượng token "share" s như sau:

$$\frac{L1}{L0} = \frac{T+s}{T}$$

$$L1 * T = L0 * (T+s)$$

$$s = \frac{(L1-L0) * T}{L0} \quad \textbf{(4)}$$

Như vậy (**4**) là công thức để tính lượng token "share" mà người dùng nhận được khi cung cấp thanh khoản. Cuối cùng, ta sẽ xây dựng công thức khi người dùng muốn sử dụng token "share" để rút thanh khoản ra khỏi pool.

1.5.3 Công thức tính lượng token nhận được khi rút thanh khoản

Khi một người dùng muốn dừng cung cấp thanh khoản, họ sẽ trả vào pool một lượng token "share" s. Như phần 1.5.2, ta định nghĩa T là tổng lượng token "share" đã được tạo ra, giả sử dX và dY là lượng token mà người dùng này nhận được sau khi ngừng cung cấp thanh khoản, ta có công thức tính dX và dY như sau:

$$dX = \frac{s}{T} * X$$

$$dY = \frac{s}{T} * Y$$

Để chứng minh công thức trên, ta định nghĩa v là giá trị tổng thanh khoản được rút ra, L là tổng giá trị thanh khoản ở trong pool, như vậy, ta có:

$$v = f(dX, dY) = \sqrt{(dX dY)}$$

$$L = f(X, Y) = \sqrt{XY}$$

Để duy trì tỷ lệ cân bằng giữa tổng giá trị thanh khoản và tổng lượng token "share" được tạo ra, ta cần điều kiện như sau:

$$\frac{v}{L} = \frac{s}{T} \quad \textbf{(5)}$$

Để duy trì tỷ giá giữa hai token ở trong pool sau khi rút thanh khoản, ta đã chứng minh ở mục 1.5.2 rằng

$$dY = \frac{Y}{X} * dX$$

Với công thức này, tiếp tục biến đổi điều kiện (**5**), ta có:

$$\frac{\sqrt{dXdY}}{\sqrt{XY}} = \frac{s}{T}$$

$$\frac{\sqrt{dX * \frac{Y}{X} * dX}}{\sqrt{XY}} = \frac{s}{T}$$

$$\frac{dX}{X} = \frac{s}{T}$$

$$dX = \frac{s}{T} * X$$

Như vậy, thông qua các mục vừa tìm hiểu, ta đã nắm được nguyên lý hoạt động cơ bản của một AMM, các công thức trong trường hợp trao đổi token, thêm thanh khoản và rút thanh khoản. Mô hình AMM sẽ được áp dụng xuyên suốt trong quá trình mua/bán và trao đổi token trên hệ thống LaunchCrypt.

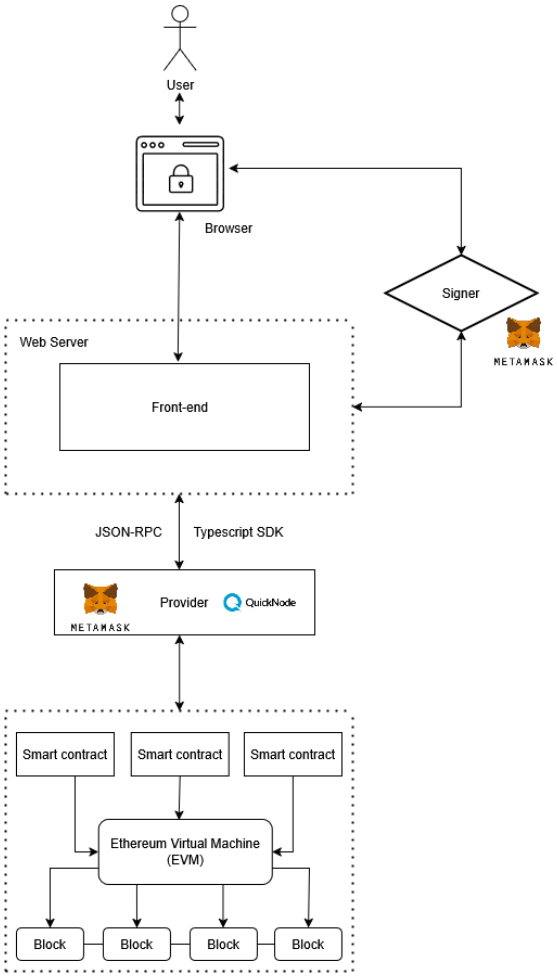
1.6 Tổng quan về các thư viện và ngôn ngữ lập trình được sử dụng trong khóa luận

1.6.1 Ngôn ngữ lập trình Typescript, thư viện Reactjs và Nestjs

Trong quá trình phát triển LaunchCrypt, việc lựa chọn công nghệ phù hợp đóng vai trò quan trọng trong việc đảm bảo hiệu suất, bảo mật và khả

năng mở rộng của dự án. Một trong những công nghệ chính được sử dụng là ngôn ngữ lập trình TypeScript kết hợp với thư viện ReactJS cho phần frontend và NestJS cho phần backend. Mỗi công nghệ này đều mang lại những lợi thế đáng kể, góp phần đẩy nhanh quá trình phát triển sản phẩm và tạo nên một cấu trúc phần mềm vức chắc.

TypeScript là một superset của JavaScript, bổ sung kiểu tĩnh và các tính năng hướng đối tượng. Trong bối cảnh hệ thống của LaunchCrypt, Typescript thể hiện khả năng hỗ trợ mạnh mẽ cho việc phát triển ứng dụng trên chuỗi khối khi các chuỗi khối này đều cung cấp SDK cho TypeScript, tạo điều kiện thuận lợi cho việc tương tác với các mạng lưới như Avalanche, Solana, và Aptos. Ngoài ra, Typescript cũng giúp lập trình viên phát hiện lỗi sớm trong quá trình phát triển, điều này là đặc biệt quan trọng khi làm việc với các hợp đồng thông minh, khi các giao dịch trên chuỗi khối không có tính chất thu hồi.



Hình 1.7: Sử dụng Typescript SDK để tương tác với chuỗi khối.

Đối với phần frontend, ReactJS được chọn làm thư viện chính để xây dựng giao diện người dùng. ReactJS, được phát triển bởi Facebook, là một trong những thư viện JavaScript phổ biến nhất cho việc xây dựng giao diện người dùng động. Lý do chính để chọn ReactJS cho LaunchCrypt bao gồm: Thứ nhất, hiệu suất cao của ReactJS, nhờ vào cơ chế Virtual DOM, giúp tối ưu hóa việc render và cập nhật UI, điều này rất quan trọng khi hiển thị dữ liệu thời gian thực như giá token và trạng thái giao dịch. Thứ hai, cộng đồng lớn và hệ sinh thái phong phú của ReactJS cung cấp nhiều thư viện và công cụ hỗ trợ, giúp đẩy nhanh quá trình phát triển. Thứ ba, khả năng tái sử dụng component của ReactJS giúp tạo ra một codebase dễ bảo trì và mở rộng. Cuối cùng, ReactJS kết hợp tốt với TypeScript, cho phép xây dựng các

components với kiểu dữ liệu rõ ràng, giảm thiểu lỗi và cải thiện trải nghiệm phát triển.

Về phía backend, NestJS được lựa chọn làm thư viện chính. NestJS là một framework Node.js tiến bộ, được phát triển từ ExpressJs và được xây dựng với TypeScript. Có nhiều lý do khiến NestJS trở thành lựa chọn lý tưởng cho việc xây dựng backend của LaunchCrypt: Đầu tiên, NestJS cung cấp một cấu trúc rõ ràng và module hóa, điều này rất quan trọng khi xây dựng một ứng dụng phức tạp như LaunchCrypt, giúp quản lý code dễ dàng hơn và cải thiện khả năng mở rộng. Thứ hai, NestJS sử dụng Typescript, là một ngôn ngữ lập trình được hỗ trợ rất tốt bởi những SDK được xây dựng sẵn trên các nền tảng chuỗi khối. Thứ ba, NestJS cung cấp nhiều decorators và utilities có sẵn, giúp đơn giản hóa việc xây dựng RESTful APIs và WebSocket gateways, rất hữu ích cho việc xử lý các yêu cầu thời gian thực trong ứng dụng DeFi. Cuối cùng, NestJS có khả năng tích hợp tốt với nhiều thư viện và công cụ Node.js khác, cho phép linh hoạt trong việc lựa chọn các công nghệ bổ sung cần thiết. Việc kết hợp TypeScript, ReactJS và NestJS tạo nên một bộ công nghệ mạnh mẽ và hiệu quả cho LaunchCrypt. TypeScript đảm bảo tính nhất quán và an toàn kiểu dữ liệu xuyên suốt toàn bộ dự án. ReactJS cung cấp một nền tảng frontend linh hoạt và hiệu suất cao, trong khi NestJS mang lại một backend có cấu trúc tốt và dễ mở rộng. Việc lựa chọn ngôn ngữ lập trình cũng như các thư viện này giúp LaunchCrypt bảo đảm được các tiêu chí về bảo mật và mở rộng, trong khi vẫn giữ một cấu trúc phần mềm vững chắc và tốc độ phát triển nhanh.

1.6.2 Ngôn ngữ lập trình Solidity, Rust và Move

Trong quá trình phát triển LaunchCrypt, một nền tảng đa chuỗi cho việc tạo và giao dịch token, ba ngôn ngữ lập trình hợp đồng thông minh chính được sử dụng là Solidity, Rust và Move, mỗi ngôn ngữ đóng vai trò quan

trọng trong việc tương tác với các chuỗi khối khác nhau. Solidity, ngôn ngữ chủ đạo cho Avalanche và nhiều chuỗi khối tương thích máy ảo Ethereum, được chọn vì khả năng phát triển hợp đồng thông minh mạnh mẽ và hệ sinh thái rộng lớn, cho phép LaunchCrypt tận dụng các thư viện có sẵn và công cụ phong phú trong lĩnh vực DeFi. Rust, với hiệu suất cao và tính an toàn vượt trội, được sử dụng để phát triển hợp đồng thông minh trên Solana, cung cấp khả năng xử lý giao dịch nhanh chóng và chi phí thấp, đặc biệt quan trọng cho các ứng dụng DeFi yêu cầu thông lượng cao. Move, ngôn ngữ được thiết kế đặc biệt cho chuỗi khối Aptos, mang đến một mô hình lập trình độc đáo tập trung vào bảo mật và linh hoạt trong quản lý tài sản kỹ thuật số, giúp LaunchCrypt triển khai các tính năng cần thiết và đảm bảo tính bảo mật trên các chuỗi khối mới nổi này. Quan trong hơn, các ngôn ngữ lập trình này đã hỗ trợ các tiêu chuẩn token được định nghĩa trong mục 1.4, giúp việc tạo và giao dịch token trở nên đơn giản và linh hoạt hơn. Việc kết hợp cả ba ngôn ngữ này trong LaunchCrypt không chỉ cho phép dự án tận dụng ưu điểm riêng của từng chuỗi khối, mà còn đảm bảo khả năng mở rộng trong tương lai, khi các hệ sinh thái chuỗi khối tiếp tục phát triển và đa dạng hóa.

1.7 Tổng kết chương

Như vậy, qua chương đầu tiên, ta có thể nắm rõ được các khái niệm cơ bản về công nghệ chuỗi khối và cơ chế hoạt động của nó, hiểu được bản chất của tài chính phi tập trung (DeFi) và vai trò của nó trong hệ sinh thái tài chính hiện đại. Chương này cũng giới thiệu chi tiết về token, cấu trúc và đặc điểm của chúng trên các chuỗi khối khác nhau, cũng như cơ chế hoạt động của thị trường tạo lập tự động (AMM) - nền tảng cho các giao dịch trong DeFi. Bên cạnh đó, chương đầu tiên cũng cung cấp tổng quan về các ngôn ngữ lập trình và thư viện quan trọng như Solidity, Rust, Move, TypeScript, ReactJS và NestJS, tạo nền tảng vững chắc cho việc phát triển dự án LaunchCrypt

trong các chương tiếp theo.

Chương 2
PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

2.1 Giới thiệu

Chương này trình bày về thiết kế của toàn bộ hệ thống LaunchCrypt. Chương sẽ tập trung vào 5 mục: **Mục 2.2** là nêu tổng quan chức năng bao gồm các biểu đồ usecase, **mục 2.3** là đặc tả chi tiết từng chức năng và luồng hoạt động của chúng, **mục 2.4** là đặc tả yêu cầu phi chức năng, **mục 2.5** là thiết kế kiến trúc, và cuối cùng, **mục 2.6** sẽ đề cập về thiết kế chi tiết của hệ thống.

2.2 Đặc tả yêu cầu

Phần này tập trung vào việc trình bày tổng quan về các chức năng chính của hệ thống LaunchCrypt thông qua biểu đồ ca sử dụng. Mục đích chính của phần này là giúp người đọc hiểu rõ về phạm vi, khả năng và các tương tác chính giữa người dùng và hệ thống. Kiến thức này sẽ tạo cơ sở vững chắc để hiểu sâu hơn về cách LaunchCrypt hoạt động và tương tác với người dùng, đồng thời cung cấp bối cảnh cho việc đặc tả chi tiết các chức năng trong phần tiếp theo.

2.2.1 Mô tả bài toán

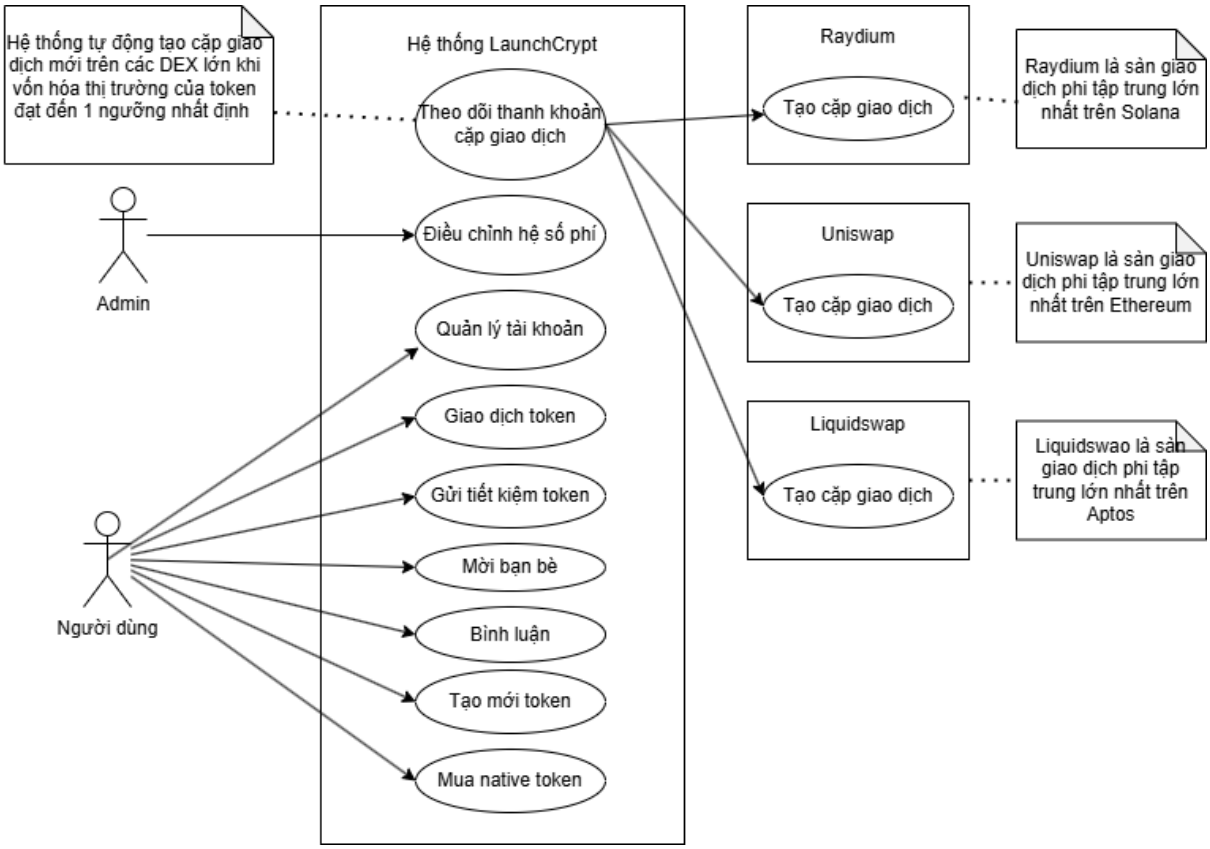
Trong bối cảnh tài chính phi tập trung (DeFi) đang phát triển mạnh mẽ, nhu cầu về một nền tảng toàn diện cho phép người dùng dễ dàng tạo, quản lý và giao dịch token trên nhiều chuỗi khối khác nhau đang ngày càng tăng cao. Hiện tại, việc phát hành và quản lý token trên các blockchain như

Avalanche, Solana hay Aptos vẫn đang đối mặt với nhiều thách thức về mặt kỹ thuật và hạ tầng. Các rào cản này bao gồm yêu cầu cao về kiến thức lập trình chuyên sâu, sự phức tạp trong quá trình triển khai hợp đồng thông minh (smart contract), cũng như khó khăn trong việc tương tác giữa các chuỗi khối khác nhau. Đặc biệt, đối với những doanh nghiệp nhỏ hoặc các cá nhân không có chuyên môn kỹ thuật, việc tham gia vào hệ sinh thái token trở nên gần như bất khả thi, dẫn đến sự thiếu cân bằng và hạn chế trong sự phát triển của hệ sinh thái Web3 nói chung.

Xuất phát từ thực trạng trên, đề tài này đề xuất phát triển một nền tảng mang tên LaunchCrypt. Đây là một giải pháp toàn diện nhằm đơn giản hóa và tối ưu hóa toàn bộ quy trình liên quan đến việc tạo, phân phối và giao dịch token trên nhiều chuỗi khối khác nhau. LaunchCrypt hướng đến việc loại bỏ các rào cản kỹ thuật hiện tại, cho phép cả người dùng chuyên nghiệp lẫn người mới bắt đầu đều có thể dễ dàng tham gia vào hệ sinh thái token. Nền tảng này không chỉ cung cấp công cụ để tạo token mà còn xây dựng một hệ sinh thái hoàn chỉnh, bao gồm các tính năng giao dịch token thông qua các cặp giao dịch được tạo tự động, khả năng gửi tiết kiệm token để sinh lợi nhuận, cũng như các tính năng tương tác cộng đồng nhằm tăng cường trải nghiệm người dùng và tạo ra một môi trường đầu tư lành mạnh, minh bạch.

Mục tiêu chính của LaunchCrypt là trở thành cầu nối giữa công nghệ chuỗi khối phức tạp và nhu cầu ứng dụng thực tiễn của người dùng, từ đó thúc đẩy quá trình phổ cập và áp dụng rộng rãi công nghệ chuỗi khối trong các lĩnh vực khác nhau. Bằng cách cung cấp một nền tảng dễ sử dụng, an toàn và hiệu quả, LaunchCrypt không chỉ giải quyết các vấn đề hiện tại trong hệ sinh thái token mà còn mở ra những cơ hội mới cho sự phát triển và đổi mới trong lĩnh vực tài chính phi tập trung, góp phần vào sự phát triển bền vững, an toàn của nền kinh tế số trong tương lai.

2.2.2 Mô hình ca sử dụng



Hình 2.1: Biểu đồ ca sử dụng tổng quát.

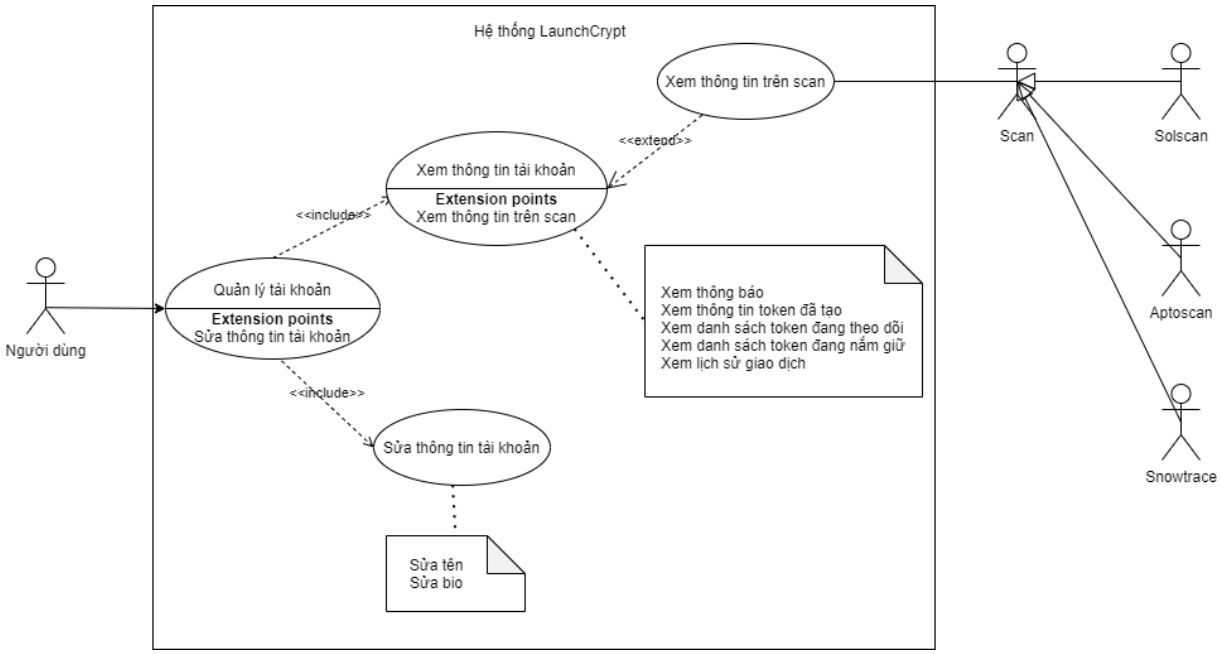
Các tác nhân tham gia trong biểu đồ:

1. **Người quản lý** (admin): Nắm vai trò kiểm soát hệ thống, quản lý hệ số phí tạo cũng như giao dịch token.

2. **Người dùng**: Người có nhu cầu sử dụng dịch vụ của hệ thống LaunchCrypt.

3. **Raydium/Liquidswap/Uniswap**: Các sàn giao dịch phi tập trung lớn trên các chuỗi khác nhau, là nơi thanh khoản sẽ được đẩy lên cho quá trình "tốt nghiệp" token.

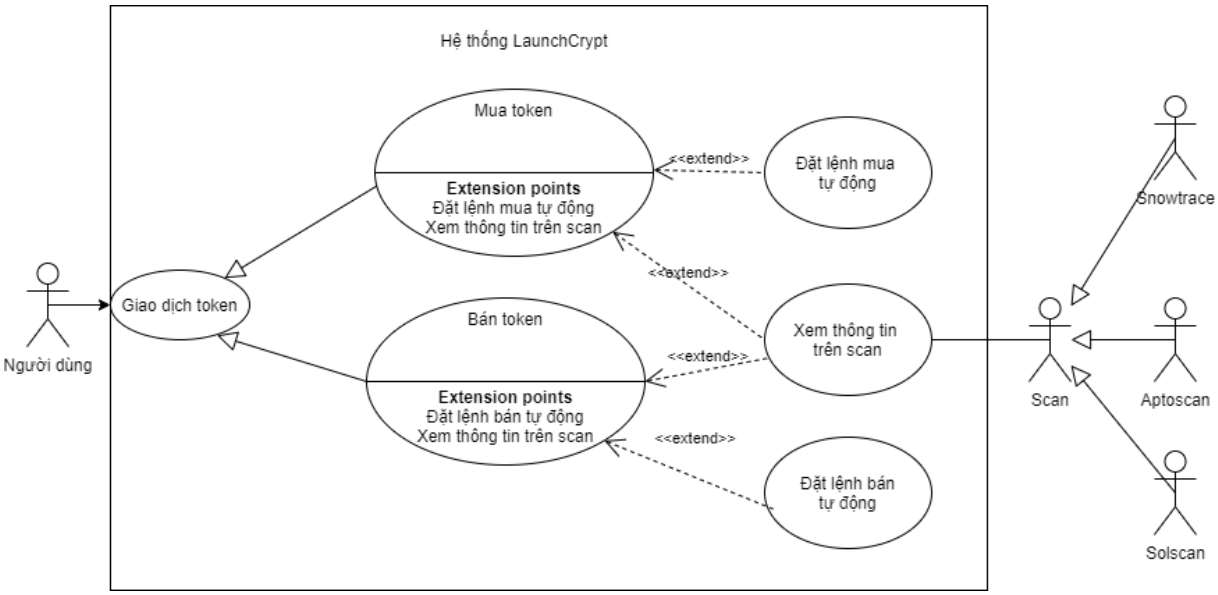
4. **CoinBase**: Là sàn giao dịch tập trung được dùng phổ biến, được sử dụng khi người dùng có nhu cầu mua native token của các chuỗi khối layer 1.



Hình 2.2: Biểu đồ phân rã ca sử dụng quản lý tài khoản.

Tác nhân: Người dùng

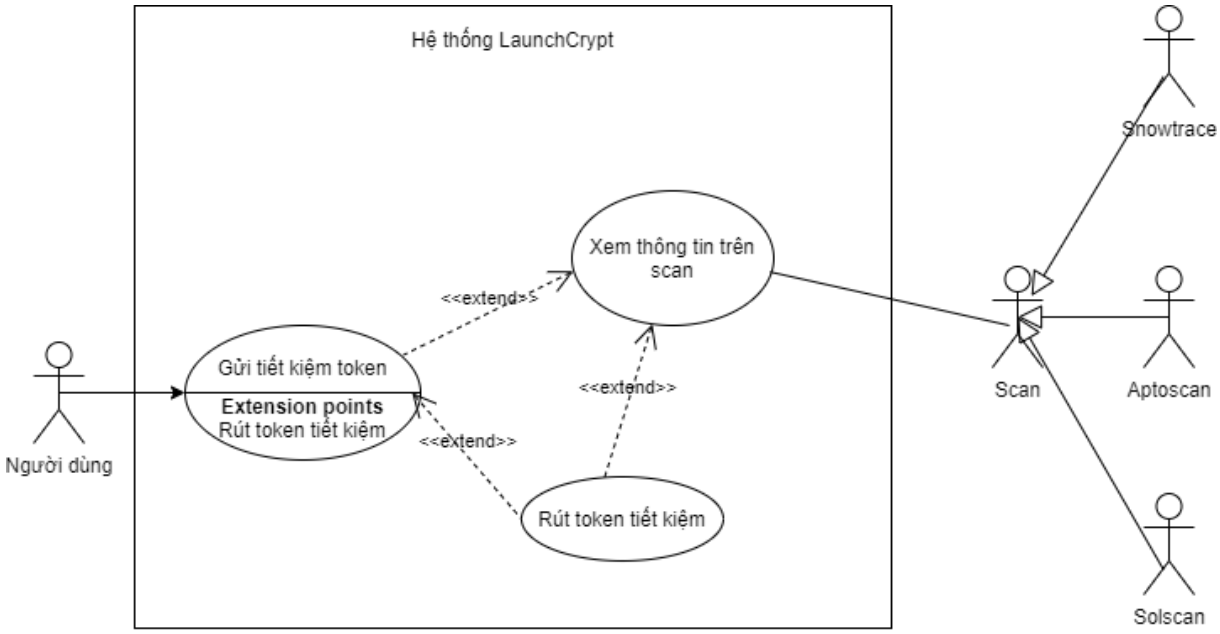
Mô tả: Người dùng xem thông tin cá nhân và chỉnh sửa thông tin bao gồm tên và tiểu sử (bio). Ngoài ra, người dùng có thể xem thông tin về ví của mình trên scan.



Hình 2.3: Biểu đồ phân rã ca sử dụng giao dịch token.

Tác nhân: Người dùng

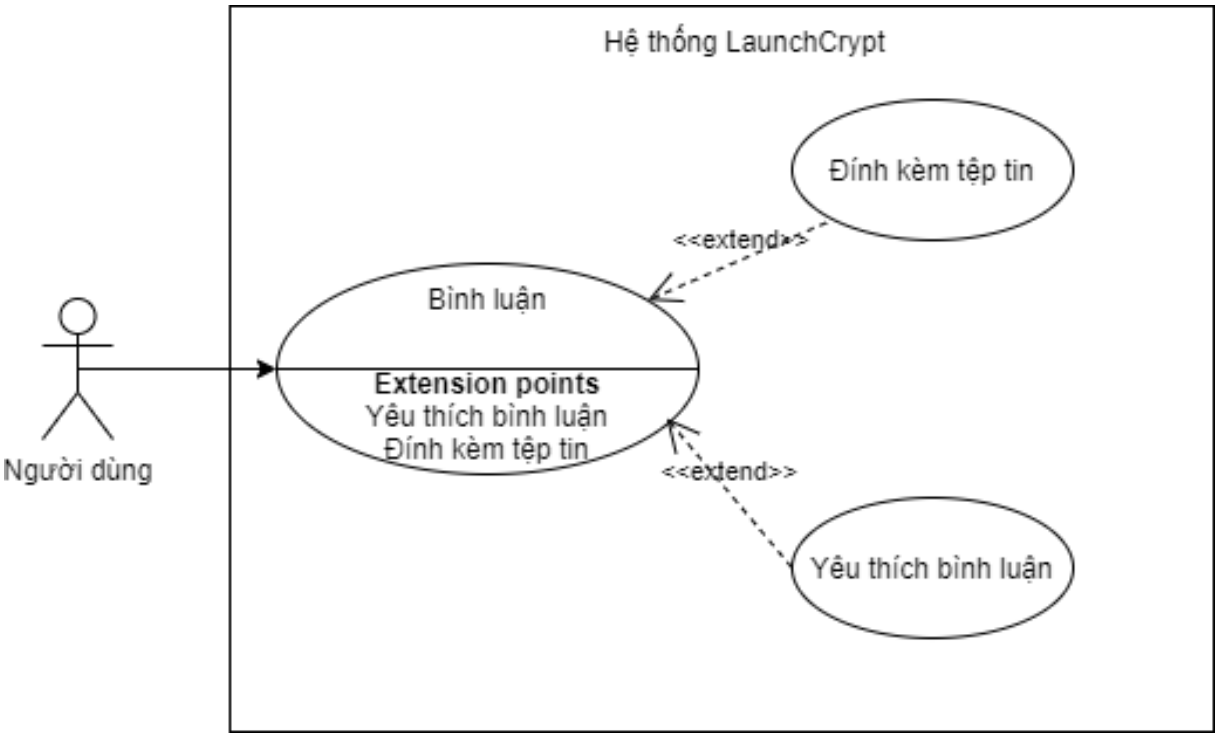
Mô tả: Người dùng giao dịch, mua bán, trao đổi giữa các token được tạo trên hệ thống và native token trên nền tảng LaunchCrypt. Ngoài ra, người dùng có thể đặt các lệnh mua, bán tự động hoặc xem thông tin đầy đủ về giao dịch trên trang scan.



Hình 2.4: Biểu đồ phân rã ca sử dụng gửi tiết kiệm token.

Tác nhân: Người dùng

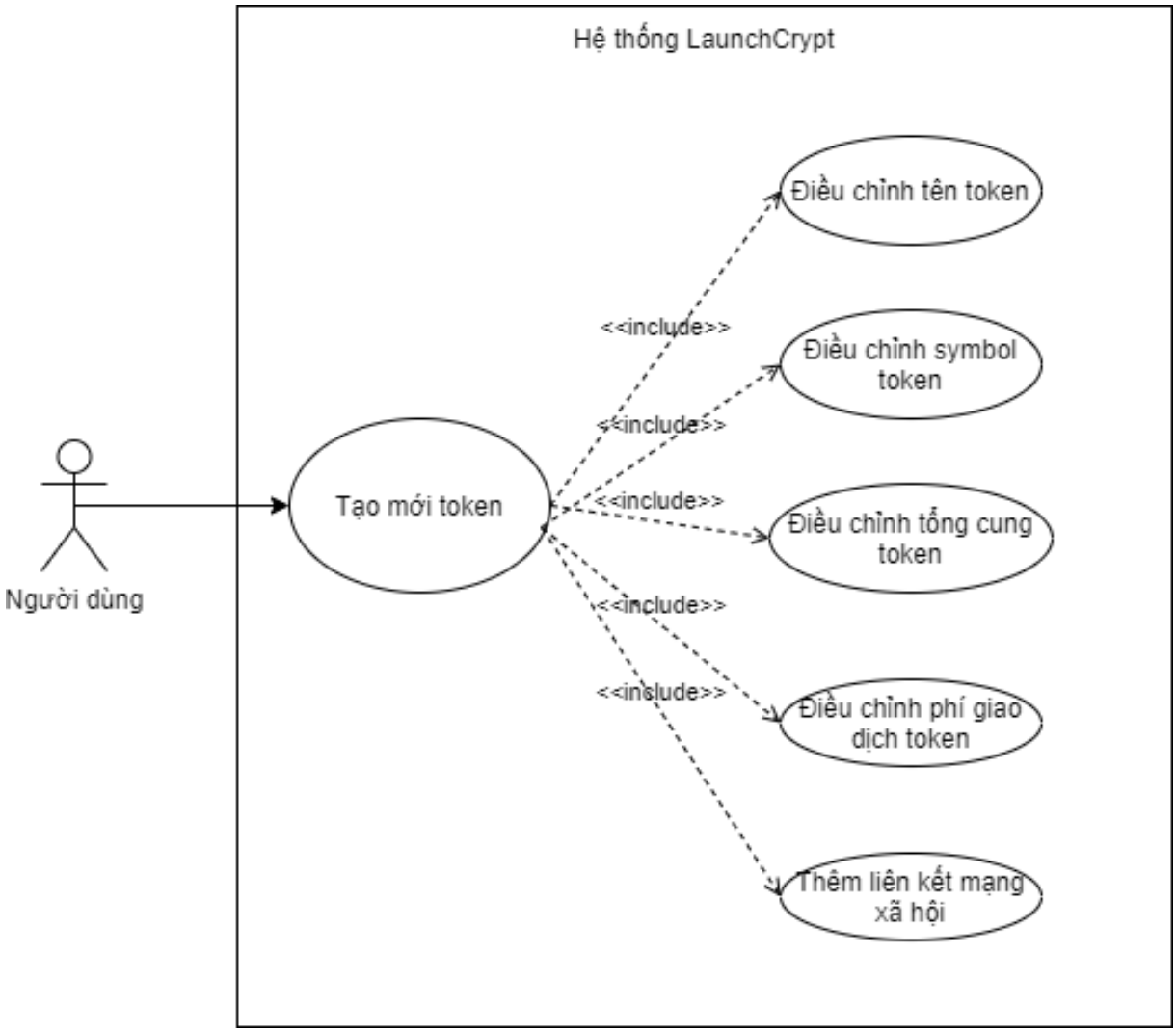
Mô tả: Người dùng gửi tiết kiệm để nhận về 1 khoản lãi suất theo tỷ lệ đã quy định trên nền tảng LaunchCrypt. Người dùng có thể rút tiền gửi tiết kiệm khi đạt đủ mốc thời gian và xem thông tin chi tiết về các giao dịch trên trang scan.



Hình 2.5: Biểu đồ phân rã ca sử dụng bình luận.

Tác nhân: Người dùng

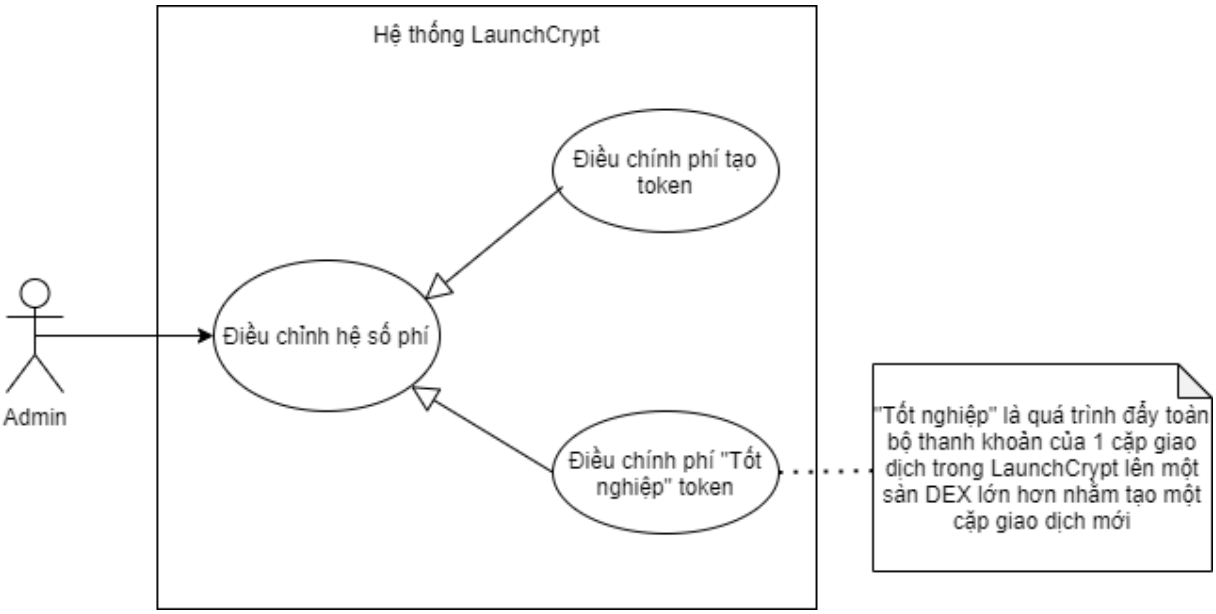
Mô tả: Khi người dùng xem thông tin giao dịch của 1 token, họ có thể để lại bình luận hoặc trao đổi về thông tin token đó với những người dùng khác. Người dùng cũng có thể yêu thích các bình luận, trả lời các bình luận của người dùng khác hoặc đính kèm tệp tin trong bình luận của mình.



Hình 2.6: Biểu đồ phân rã ca sử dụng tạo mới token.

Tác nhân: Người dùng

Mô tả: Người dùng tạo mới một token và điều chỉnh thông số của token theo ý muốn. Sau khi token được tạo thành công, hệ thống LaunchCrypt sẽ tự động triển khai một hợp động thông minh đại diện cho cặp giao dịch giữa token này và native token.



Hình 2.7: Biểu đồ phân rã ca sử dụng điều chỉnh hệ số phí.

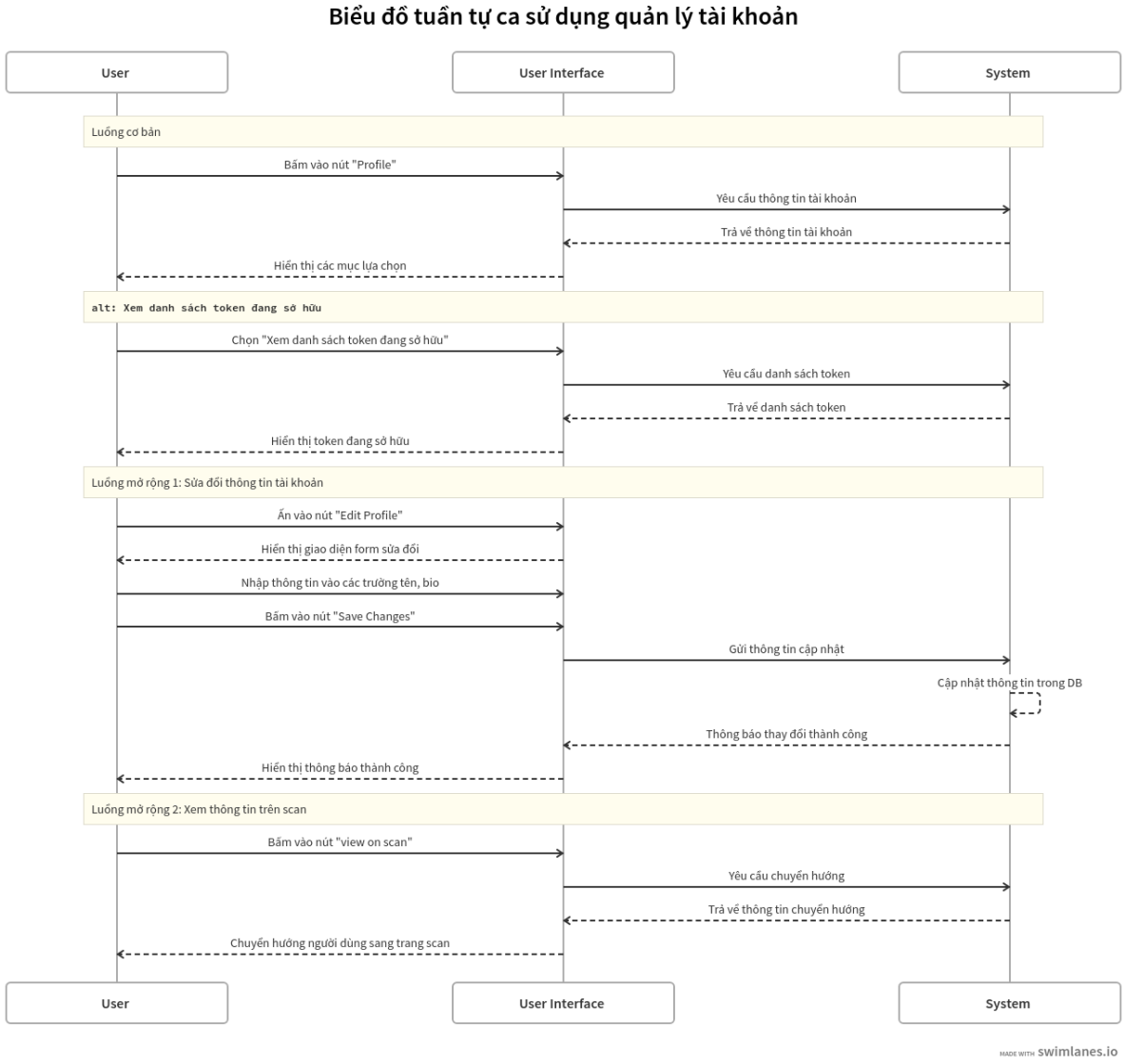
Tác nhân: Admin

Mô tả: Admin điều chỉnh lượng phí mà người dùng cần trả trong quá trình tạo và "tốt nghiệp"token.

2.2.3 Đặc tả ca sử dụng

Use case	Quản lý tài khoản
Tác nhân	Người dùng
Tiền điều kiện	Người dùng đã đăng nhập thành công.
Hậu điều kiện	Người dùng xem/cập nhật thành công thông tin tài khoản của mình.
Luồng cơ bản	1) Người dùng lựa chọn "Profile". 2) Hệ thống hiển thị giao diện quản lý tài khoản với các tùy chọn. 3) Người dùng lựa chọn các mục được hiển thị trên giao diện, bao gồm: • Xem thông báo. • Xem danh sách token đang sở hữu. • Xem thông tin token đã tạo. • Xem danh sách token đang theo dõi. • Xem danh sách user đang theo dõi. • Xem danh sách token đang nắm giữ. • Xem lịch sử giao dịch. 4) Hệ thống hiển thị thông tin tương ứng với mục mà người dùng lựa chọn.
Luồng mở rộng 1: Sửa đổi thông tin tài khoản.	3.1) Người dùng lựa chọn "Edit Profile" 4.1) Hệ thống hiển thị giao diện form sửa đổi thông tin tài khoản 5.1) Người dùng nhập thông tin vào các trường tên, bio. 6.1) Người dùng chọn "Save Changes" 7.1) Hệ thống hiển thị thông báo thay đổi thay công.
Luồng mở rộng 2: Xem thông tin trên scan	4.2) Người dùng chọn "view on scan". 5.2) Hệ thống chuyển hướng người dùng sang trang scan, hiển thị thông tin về tài khoản, giao dịch, ...

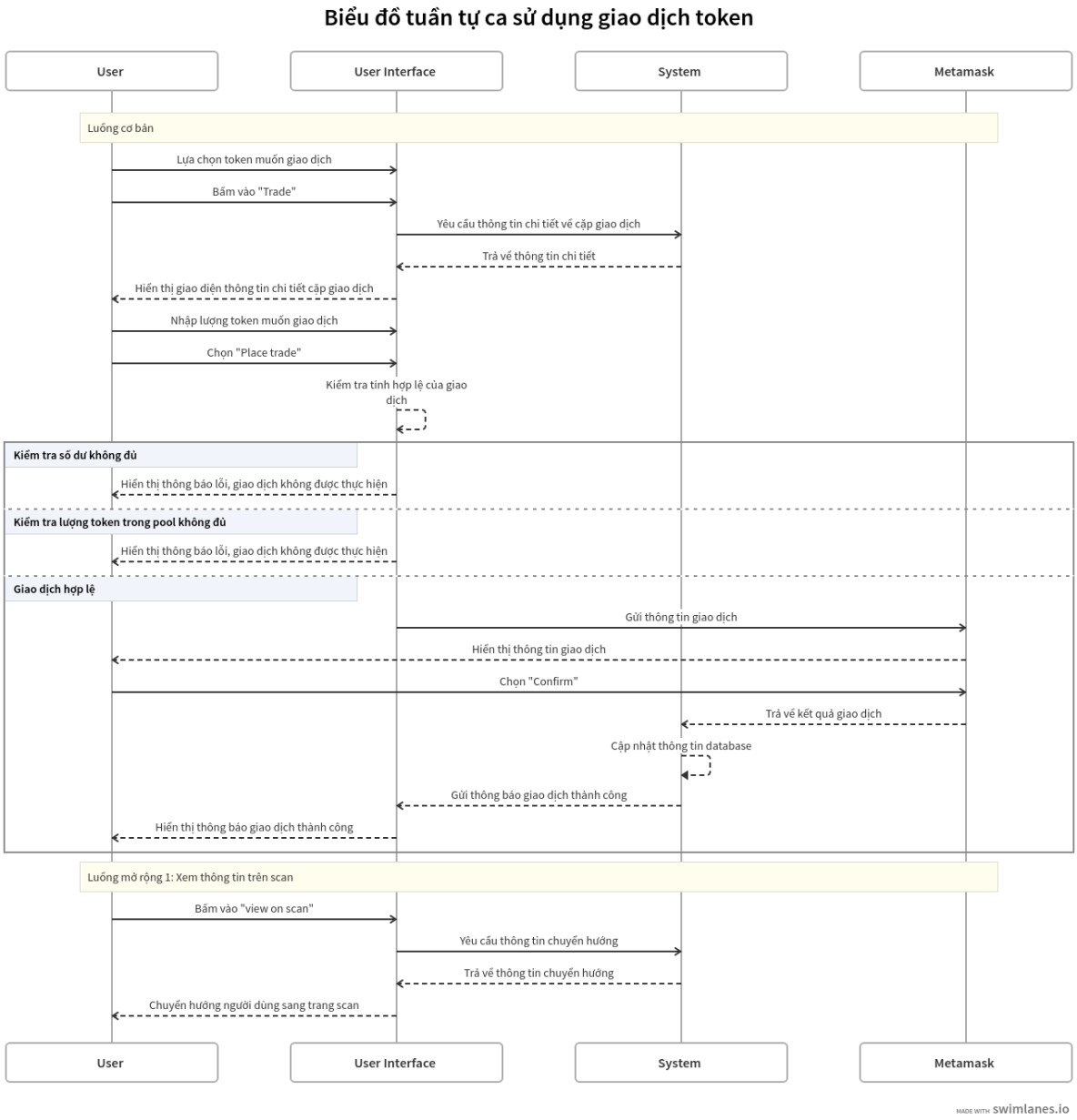
Bảng 2.1: Đặc tả ca sử dụng quản lý tài khoản.



Hình 2.8: Biểu đồ tuần tự ca sử dụng quản lý tài khoản.

Use case	Giao dịch token
Tác nhân	Người dùng
Tiền điều kiện	Người dùng đăng nhập thành công vào hệ thống.
Hậu điều kiện	Người dùng giao dịch token thành công.
Luồng cơ bản	1) Người dùng tìm kiếm thông tin token, hoặc xem thông tin token hiển thị sẵn trên hệ thống. 2) Người dùng lựa chọn token muốn giao dịch, sau đó chọn "Trade". 3) Hệ thống hiển thị giao diện thông tin chi tiết về token. 4) Ở mục “Buy/Sell” trên giao diện, người dùng nhập lượng token muốn giao dịch. 5) Người dùng chọn “Place trade” 6) Hệ thống hiển thị giao diện xác nhận bao gồm thông tin về transaction, số token người dùng cần chi trả và số token người dùng nhận được 7) Người dùng chọn “Confirm”. 8) Hệ thống hiển thị thông báo giao dịch thành công.
Luồng mở rộng 1: Xem thông tin trên scan	8.1) Người dùng chọn “view on scan” 9.1) Hệ thống hiển thị thông tin giao dịch trên trang scan tương ứng.
Luồng thay thế 1: Người dùng không đủ token trong tài khoản	6.2) Hệ thống hiển thị thông báo lỗi, giao dịch không được thực hiện
Luồng thay thế 2: Pool giao dịch không còn đủ token	6.3) Hệ thống hiển thị thông báo lỗi, giao dịch không được thực hiện

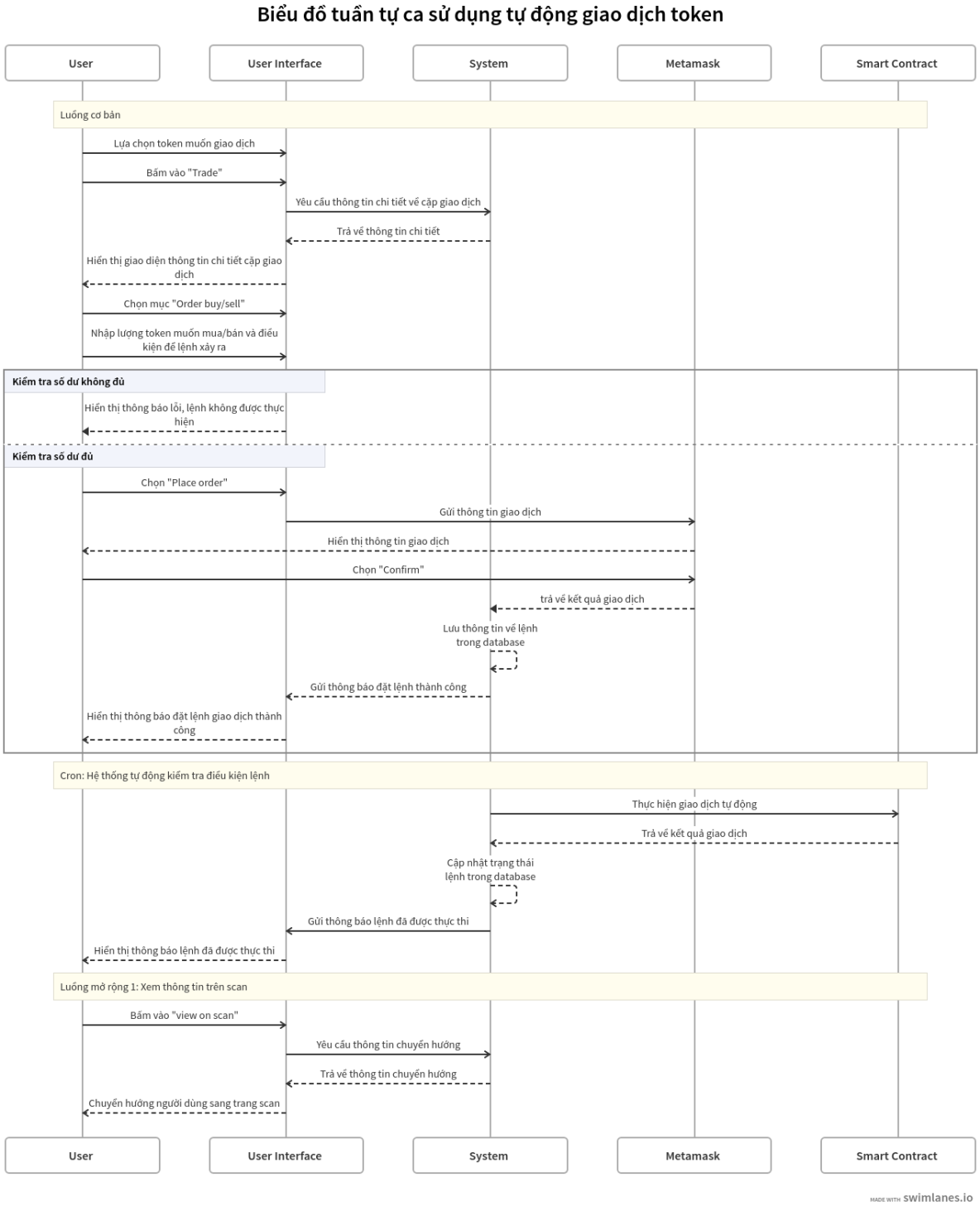
Bảng 2.2: Đặc tả ca sử dụng giao dịch token



Hình 2.9: Biểu đồ tuần tự ca sử dụng giao dịch token.

Use case	Tự động giao dịch token
Tác nhân	Người dùng
Tiền điều kiện	Người dùng đăng nhập thành công vào hệ thống.
Hậu điều kiện	Người dùng đặt lệnh giao dịch token thành công.
Luồng cơ bản	1) Người dùng tìm kiếm thông tin token, hoặc xem thông tin token hiển thị sẵn trên hệ thống. 2) Người dùng lựa chọn token muốn đặt lệnh giao dịch. 3) Hệ thống hiển thị giao diện thông tin chi tiết về token. 4) Ở mục "Order buy/sell" trên giao diện, người dùng nhập lượng token muốn giao dịch, lượng native token tối đa cho lệnh (trường hợp lệnh mua) cũng như lượng thanh khoản cần đạt để lệnh được thực thi. 5) Người dùng chọn "Place order" 6) Hệ thống hiển thị giao diện xác nhận bao gồm thông tin về transaction, số token người dùng cần chi trả nếu lệnh được thực thi. 7) Người dùng chọn "Confirm". 8) Hệ thống hiển thị thông báo đặt lệnh giao dịch thành công.
Luồng mở rộng 1: Xem thông tin trên scan	8.1) Người dùng chọn "view on scan" 9.1) Hệ thống hiển thị thông tin giao dịch trên trang scan tương ứng.
Luồng thay thế 1: Người dùng không đủ token để chi trả cho lệnh	6.3) Lệnh bị hủy, hệ thống hiển thị thông báo đặt lệnh không thành công.

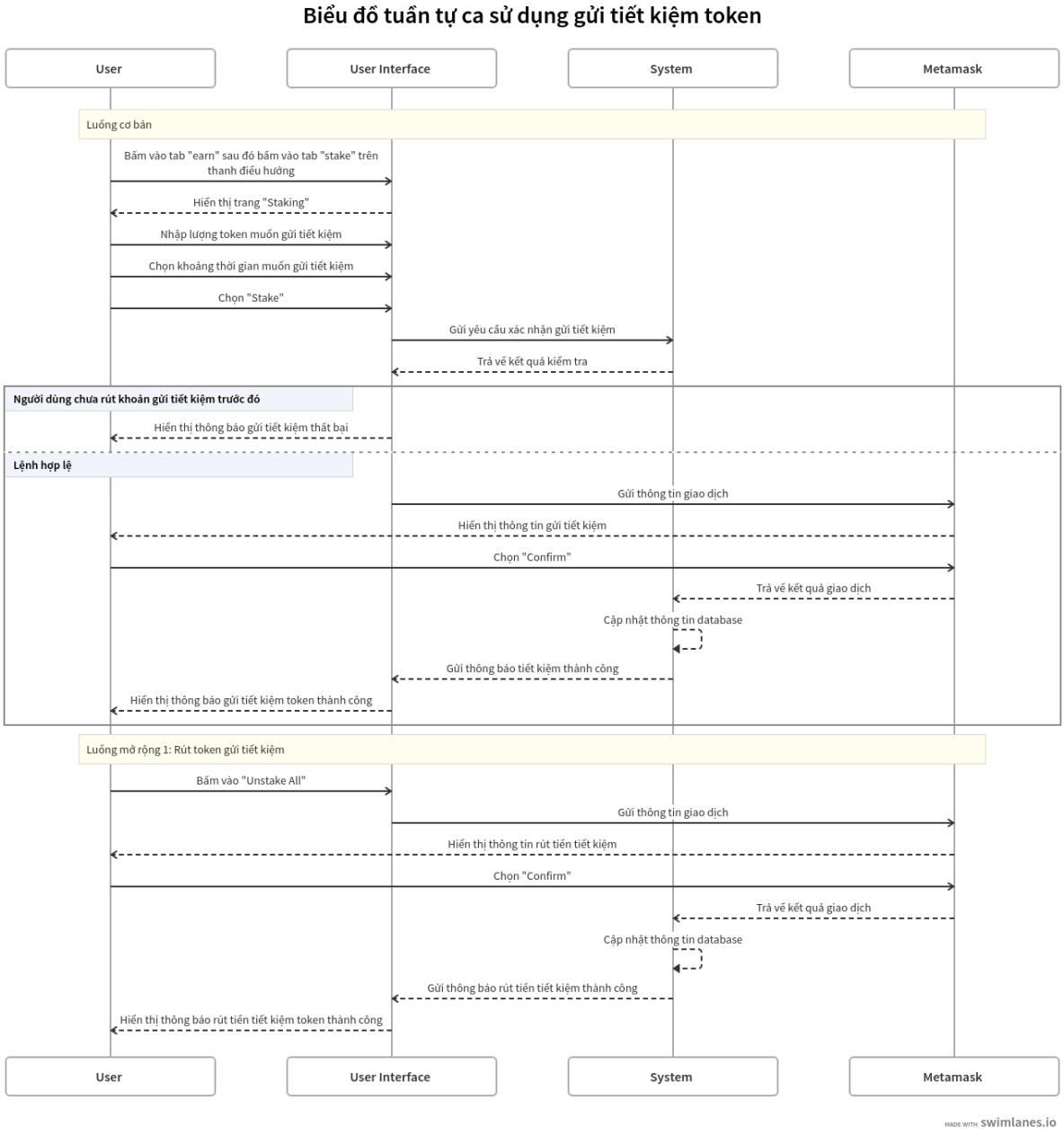
Bảng 2.3: Đặc tả ca sử dụng tự động đặt lệnh giao dịch token



Hình 2.10: Biểu đồ tuần tự ca sử dụng tự động giao dịch token.

Use case	Gửi tiết kiệm token
Tác nhân	Người dùng
Tiền điều kiện	Người dùng đăng nhập thành công vào hệ thống.
Hậu điều kiện	Người dùng gửi tiết kiệm thành công.
Luồng cơ bản	1) Người dùng chọn "earn"trên thanh điều hướng, sau đó chọn "stake". 2) Người dùng nhập lượng token muốn gửi tiết kiệm. 3) Người dùng chọn khoảng thời gian muốn gửi tiết kiệm . 4) Người dùng chọn "Stake". 5) Hệ thống hiển thị giao diện xác nhận bao gồm thông tin về transaction, số token người dùng cần chi trả nếu lệnh được thực thi. 6) Người dùng chọn “Confirm”. 7) Hệ thống hiển thị thông báo gửi tiết kiệm token thành công.
Luồng mở rộng 1: Rút token gửi tiết kiệm	8.1) Người dùng chọn “Unstake All”. 9.1) Hệ thống hiển thị giao diện xác nhận bao gồm thông tin về transaction, số token người dùng sẽ nhận được sau khi rút tiền tiết kiệm 10.1) Hệ thống hiển thị thông tin giao dịch trên trang scan tương ứng. Trong trường hợp người dùng chưa gửi tiết kiệm trước đó, hệ thống sẽ báo lỗi.
Luồng thay thế 1: Người dùng gửi tiết kiệm khi chưa rút khoản tiền tiết kiệm cũ trước đó	5.2) Lệnh bị hủy, hệ thống hiển thị thông báo gửi tiết kiệm không thành công.

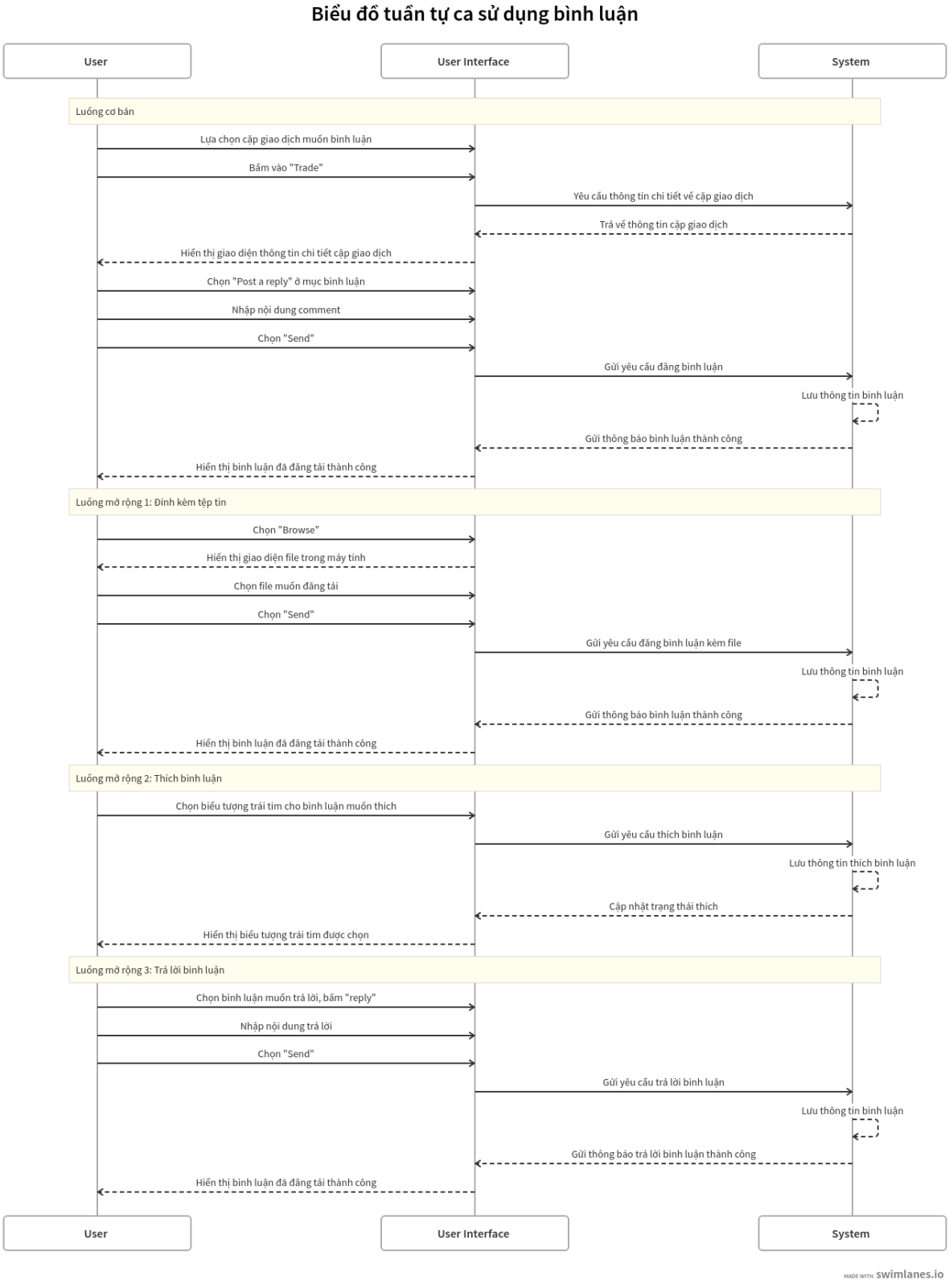
Bảng 2.4: Đặc tả ca sử dụng gửi tiết kiệm token



Hình 2.11: Biểu đồ tuần tự ca sử dụng gửi tiết kiệm token.

Use case	Bình luận
Tác nhân	Người dùng
Tiền điều kiện	Người dùng đăng nhập thành công vào hệ thống
Hậu điều kiện	Bình luận được đăng tải thành công
Luồng cơ bản	1) Người dùng tìm kiếm thông tin token, hoặc xem thông tin token hiển thị sẵn trên hệ thống. 2) Người dùng lựa chọn token muốn xem thông tin. 3) Hệ thống hiển thị giao diện thông tin chi tiết về token. 4) Ở mục bình luận, người dùng chọn "Post a reply" 5) Hệ thống hiển thị hộp thông tin bao gồm nội dung comment, file đính kèm. 6) Người dùng nhập nội dung comment. 7) Người dùng chọn "Send"
Luồng mở rộng 1: Đính kèm tệp tin	6.1) Người dùng chọn "Browse" 7.1) Hệ thống hiển thị giao diện file trong máy tính của người dùng. 8.1) Người dùng chọn file muốn đăng tải. 9.1) Người dùng chọn "Post a reply"
Luồng mở rộng 2: Thích bình luận	4.2) Người dùng chọn biểu tượng trái tim cho bình luận mà mình thích.
Luồng mở rộng 3: Trả lời bình luận	4.3) Người dùng chọn bình luận mình muốn trả lời, chọn "reply"

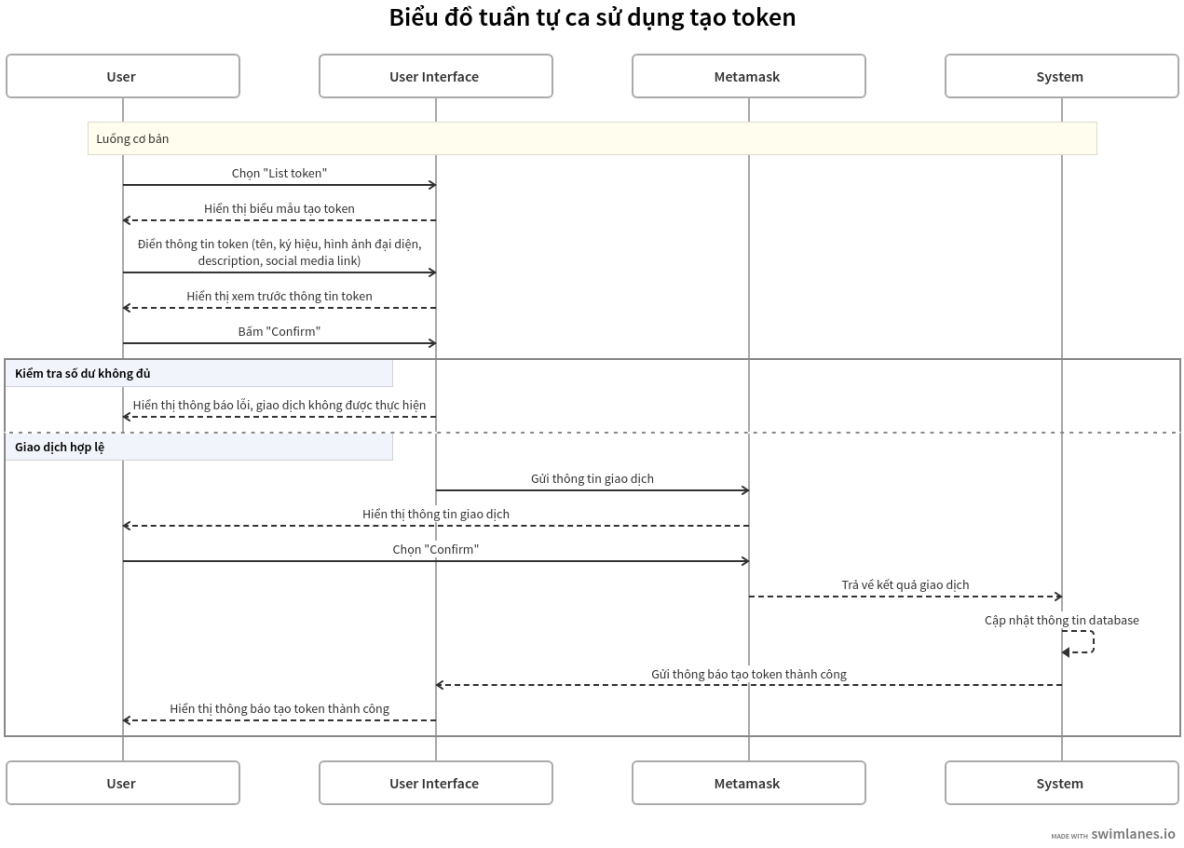
Bảng 2.5: Đặc tả ca sử dụng bình luận



Hình 2.12: Biểu đồ tuần tự ca sử dụng bình luận.

Use case	Tạo token
Tác nhân	Người dùng
Tiền điều kiện	Người dùng thành công đăng nhập vào hệ thống
Hậu điều kiện	Token được tạo thành công trên hệ thống
Luồng cơ bản	1) Người dùng chọn "List token" 2) Hệ thống hiển thị biểu mẫu bao gồm thông tin tên, ký hiệu, hình ảnh đại diện, description, social media link của token. 3) Người dùng điền đầy đủ thông tin token vào trong biểu mẫu, các trường thông tin bắt buộc bao gồm tên, ký hiệu và ảnh đại diện của token 4) Người dùng chọn "Confirm" 5) Hệ thống hiển thị giao diện xác nhận bao gồm thông tin về transaction, 6) Người dùng chọn "Confirm". 7) Hệ thống hiển thị thông báo tạo token thành công.
Luồng thay thế 1: Người dùng không đủ token để chi trả cho phí tạo giao dịch	5.1) Hệ thống hiển thị thông báo lỗi, token không được tạo thành công

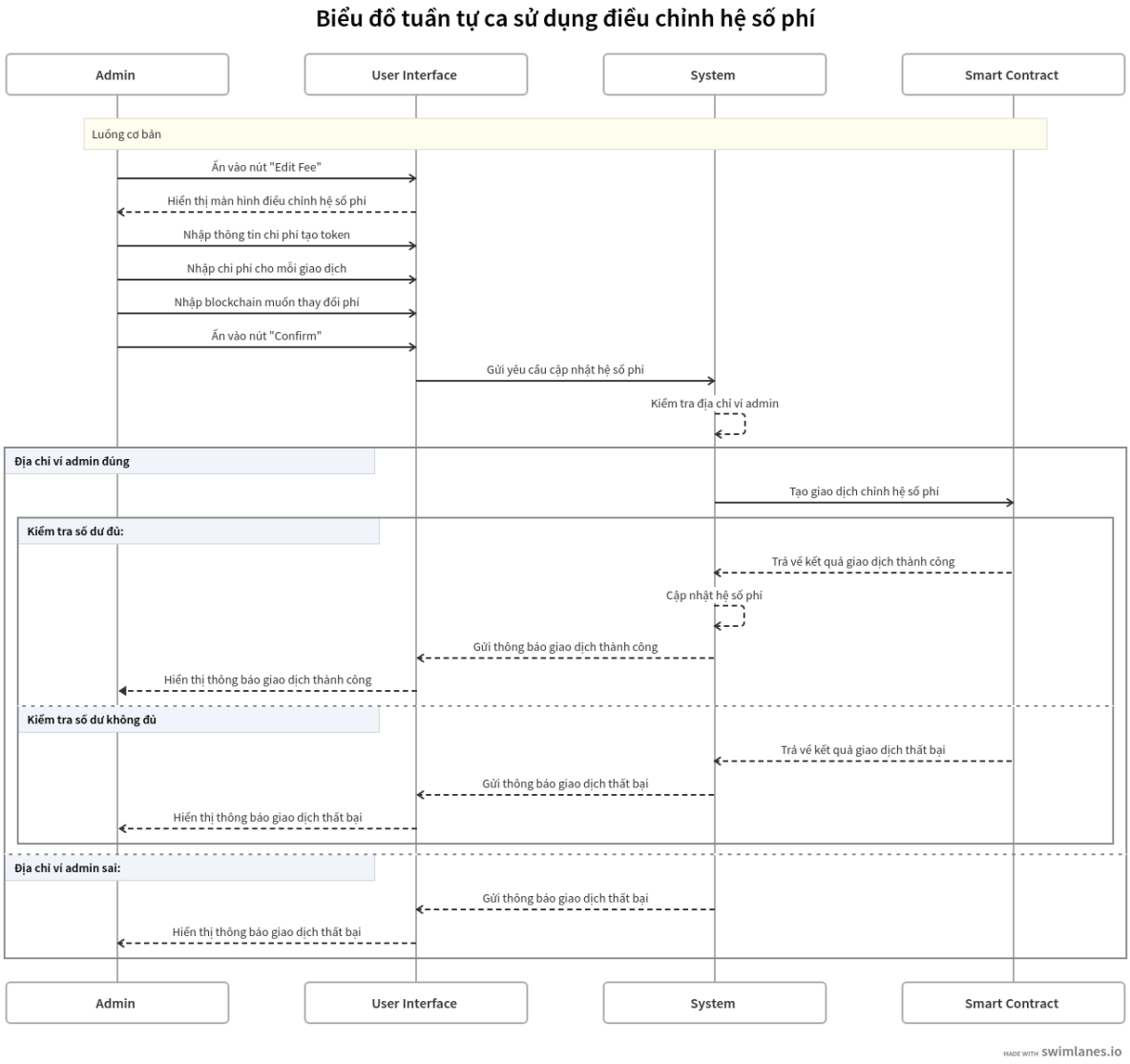
Bảng 2.6: Đặc tả ca sử dụng tạo token



Hình 2.13: Biểu đồ tuần tự ca sử dụng tạo token.

Use case	Điều chỉnh hệ số phí.
Tác nhân	Admin của hệ thống.
Tiền điều kiện	Admin thành công đăng nhập vào hệ thống.
Hậu điều kiện	Hệ số phí (bao gồm chi phí tạo token, chi phí cho mỗi giao dịch) thay đổi theo ý muốn của admin.
Luồng cơ bản	1) Admin ấn vào nút "Edit Fee" 2) Hệ thống hiển thị giao diện màn hình điều chỉnh hệ số phí. 3) Admin nhập đầy đủ thông tin các trường bao gồm chi phí tạo token, chi phí cho mỗi giao dịch, chuỗi khối muốn thay đổi phí. Giá trị ban đầu của hai trường này sẽ là giá trị đang được sử dụng. 4) Sau khi hoàn tất bước (3), Admin ấn vào nút "Confirm". 5) Hệ thống hiển thị thông báo thay đổi hệ số phí thành công.
Luồng thay thế 1: Không đủ token trong ví của admin	4.1) Ví của admin không còn đủ token để trả phí gas 5.1) Hệ thống hiển thị thông báo thay đổi hệ số phí thất bại.

Bảng 2.7: Đặc tả ca sử dụng điều chỉnh hệ số phí.



Hình 2.14: Biểu đồ tuần tự ca sử dụng điều chỉnh hệ số phí.

2.2.4 Yêu cầu phi chức năng

STT	Yêu cầu	Mô tả
1	Độ trễ thấp	Yêu cầu phần mềm có độ trễ thấp (<2 giây) để phục vụ cho nhu cầu giao dịch, tránh tình trạng giao dịch bị tắc hoặc bị trượt giá thường xuyên..
2	Tính sẵn sàng	Phần mềm cần có khả năng phục vụ các nhu cầu trong mọi thời điểm.
3	Dữ liệu nhất quán	Đồng bộ dữ liệu được lưu trong phần mềm và các dữ liệu lưu trữ onchain.
4	Tính bảo mật	Chi người có quyền mới được cập nhật cách hoạt động của hợp đồng thông minh. Ví admin phải là ví multisignature.
5	Khả năng bảo trì	Mã nguồn được cấu trúc theo mô hình phát triển hướng nghiệp vụ và kiến trúc sạch (Clean architecture)

Bảng 2.8: Yêu cầu phi chức năng của hệ thống

2.3 Thiết kế kiến trúc

2.3.1 Tổng quan kiến trúc

Hệ thống được xây dựng theo mô hình 3-layer architecture, một mô hình kiến trúc phổ biến trong phát triển phần mềm, cho phép tách biệt các thành phần logic và dễ dàng bảo trì, mở rộng. Kiến trúc này bao gồm ba lớp chính:

Presentation Layer (Controller Layer):

- Xử lý các HTTP requests từ client
- Định tuyến (routing) request đến các xử lý tương ứng
- Validate dữ liệu đầu vào
- Trả về response cho client

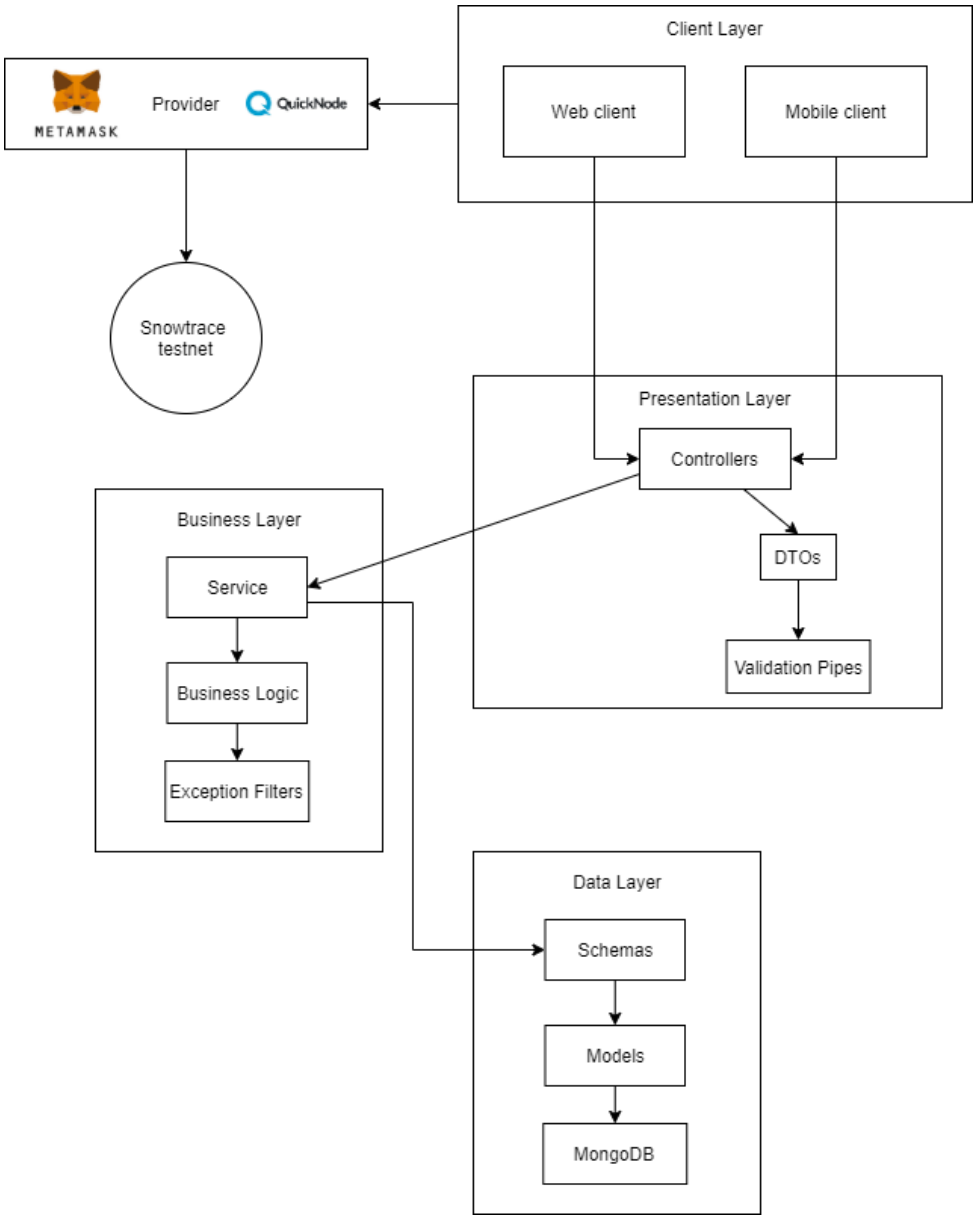
Business Layer (Service Layer):

- Chứa toàn bộ business logic của ứng dụng

- Xử lý nghiệp vụ và các quy tắc kinh doanh
- Kết nối giữa Controller Layer và Data Layer
- Xử lý các trường hợp ngoại lệ

Data Layer (Schema Layer):

- Tương tác trực tiếp với cơ sở dữ liệu
- Định nghĩa cấu trúc dữ liệu
- Xử lý các thao tác CRUD với database



Hình 2.15: Sơ đồ kiến trúc tổng quan của hệ thống.

2.3.2 Điểm nổi bật của kiến trúc

Tính module hóa cao

- Mỗi tính năng được tổ chức thành một module riêng biệt
- Dễ dàng phát triển và bảo trì từng module độc lập mà không cần phải triển khai toàn bộ hệ thống
- Khả năng tái sử dụng code cao
- Khả năng cô lập lỗi, lỗi ở một dịch vụ sẽ không làm ngừng toàn bộ hệ thống

Dependency Injection

- Giảm thiểu boilerplate code, từ đó làm cho mã nguồn ngắn gọn và dễ bảo trì hơn
- Dễ dàng thay đổi implementation
- Thuận lợi cho việc testing

Type Safety

- Sử dụng TypeScript để đảm bảo type safety
- Giảm thiểu lỗi runtime
- Tăng khả năng maintain code

2.3.3 Chi tiết các layer

Controller Layer

Controller Layer đóng vai trò như một lớp giao tiếp giữa client và hệ thống, xử lý các HTTP requests và định tuyến chúng đến các xử lý tương ứng. Layer này được cài đặt thông qua các class được đánh dấu với decorator `@Controller`.

Cấu trúc của Controller

```
@Controller('trading-pairs')
export class TradingPairsController {
    constructor(private tradingPairService: TradingPairsService) {}

    @Get()
    getAllTradingPairs(queryAllDto: QueryAllDto) {
        return this.tradingPairService.getAllTradingPairs(queryAllDto);
    }

    @Post()
    createTradingPair(@Body() createTradingPairDto: CreateTradingPairDto) {
        return this.tradingPairService.createTradingPair(
            createTradingPairDto);
    }
}
```

Chức năng chính của Controller Layer

- **Request Handling:** Xử lý các HTTP requests thông qua các decorators như `@Get()`, `@Post()`, `@Put()`, `@Delete()`
- **Parameter Extraction:** Trích xuất dữ liệu từ request thông qua các decorators `@Body()`, `@Query()`, `@Param()`
- **Data Validation:** Validate dữ liệu đầu vào thông qua các DTO (Data Transfer Objects)
- **Route Management:** Quản lý các endpoints của API

Validation và DTOs

```
export class CreateTradingPairDto {
    @IsNotEmpty()
    creator: string;

    @IsNotEmpty()
    @ValidateNested({ each: true })
    @Type(() => Token)
    tokenA: Token;

    @IsNotEmpty()
    @ValidateNested({ each: true })
    @Type(() => Token)
    tokenB: Token;
}
```

Data Transfer Objects (DTOs) được sử dụng để định nghĩa cấu trúc dữ liệu truyền giữa client và server:

Service Layer

Service Layer chứa business logic của ứng dụng và đóng vai trò trung gian giữa Controller Layer và Data Layer. Layer này được cài đặt thông qua các class được đánh dấu với decorator `@Injectable()`.

Cấu trúc của Service

```
@Injectable()
export class TradingPairsService {
   constructor(
       @InjectModel(TradingPair.name)
       private tradingPairModel: Model<TradingPair>
   ) {}

   async getAllTradingPairs(queryAllDto: QueryAllDto): Promise<TradingPair
       []> {
       const { page = 1, limit = 20, sortField, sortOrder = 'asc' } =
           queryAllDto;
       const skip = (page - 1) * limit;
       const sort = sortField ?
           { [sortField]: sortOrder === 'asc' ? 1 : -1 } : {};

       return await this.tradingPairModel
           .find()
           .skip(skip)
           .limit(limit)
           .sort(sort)
           .exec();
   }
}
```

Chức năng chính của Service Layer

- **Business Logic:** Xử lý các logic nghiệp vụ phức tạp
- **Data Transformation:** Chuyển đổi dữ liệu giữa các layer
- **Error Handling:** Xử lý và bắt các lỗi phát sinh
- **Transaction Management:** Quản lý các giao dịch với database

Data Layer

Data Layer là lớp tương tác trực tiếp với cơ sở dữ liệu, được cài đặt thông qua các Schema và Model của Mongoose.

Cấu trúc của Schema

```
@Schema({
    timestamps: true,
    toJSON: {
        virtuals: true,
        transform: (_, ret) => {
            delete ret._id;
            delete ret.__v;
            return ret;
        }
    }
})
export class TradingPair {
    @Prop({ required: true })
    creator: string;

    @Prop({ required: true, type: TokenSchema })
    tokenA: Token;

    @Prop({ required: true, type: TokenSchema })
    tokenB: Token;
}
```

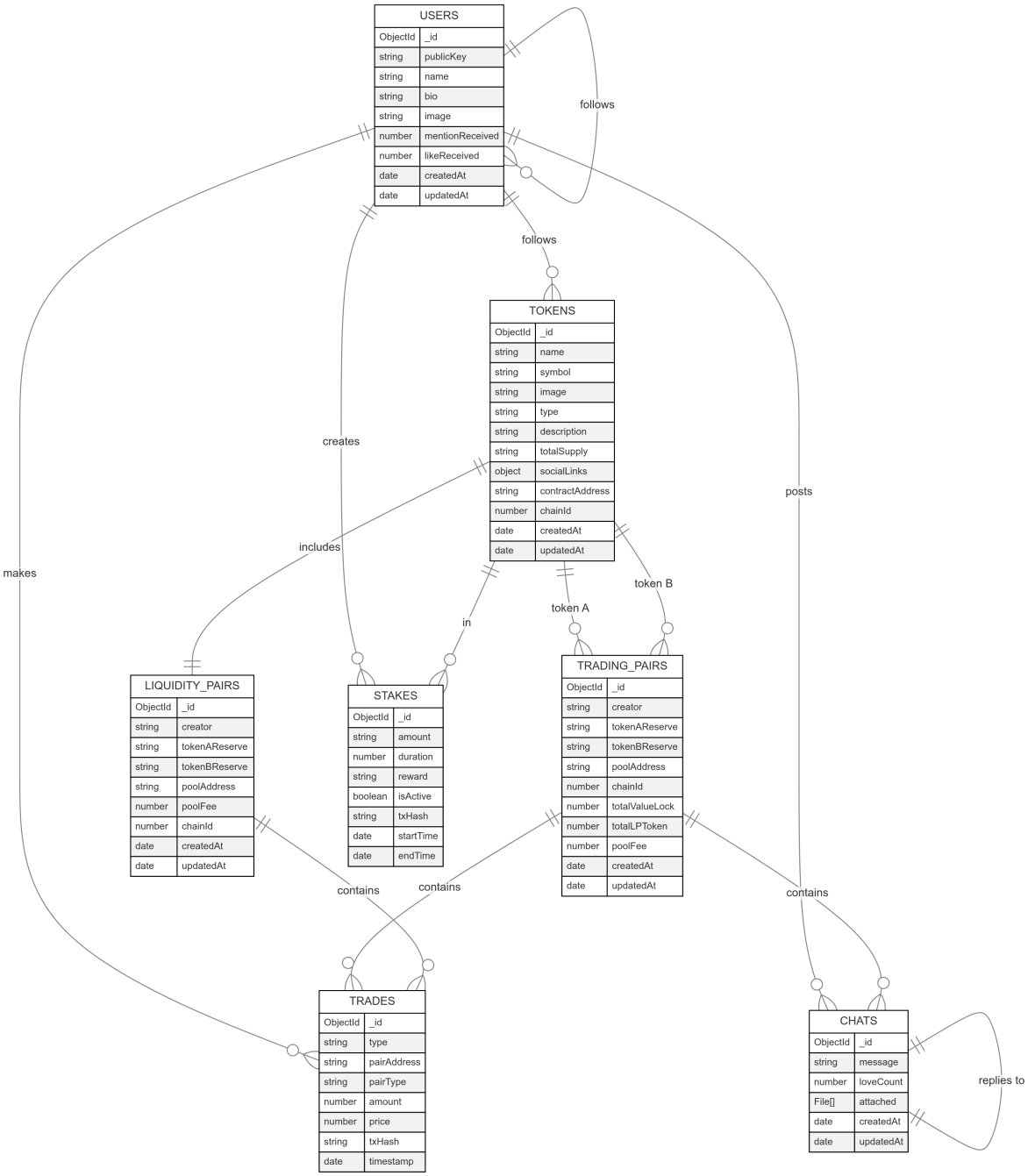
Chức năng chính của Data Layer

- **Data Definition:** Định nghĩa cấu trúc dữ liệu thông qua Schema
- **Data Access:** Cung cấp các phương thức truy cập dữ liệu
- **Data Validation:** Validate dữ liệu ở mức Schema
- **Query Building:** Xây dựng và thực thi các truy vấn database

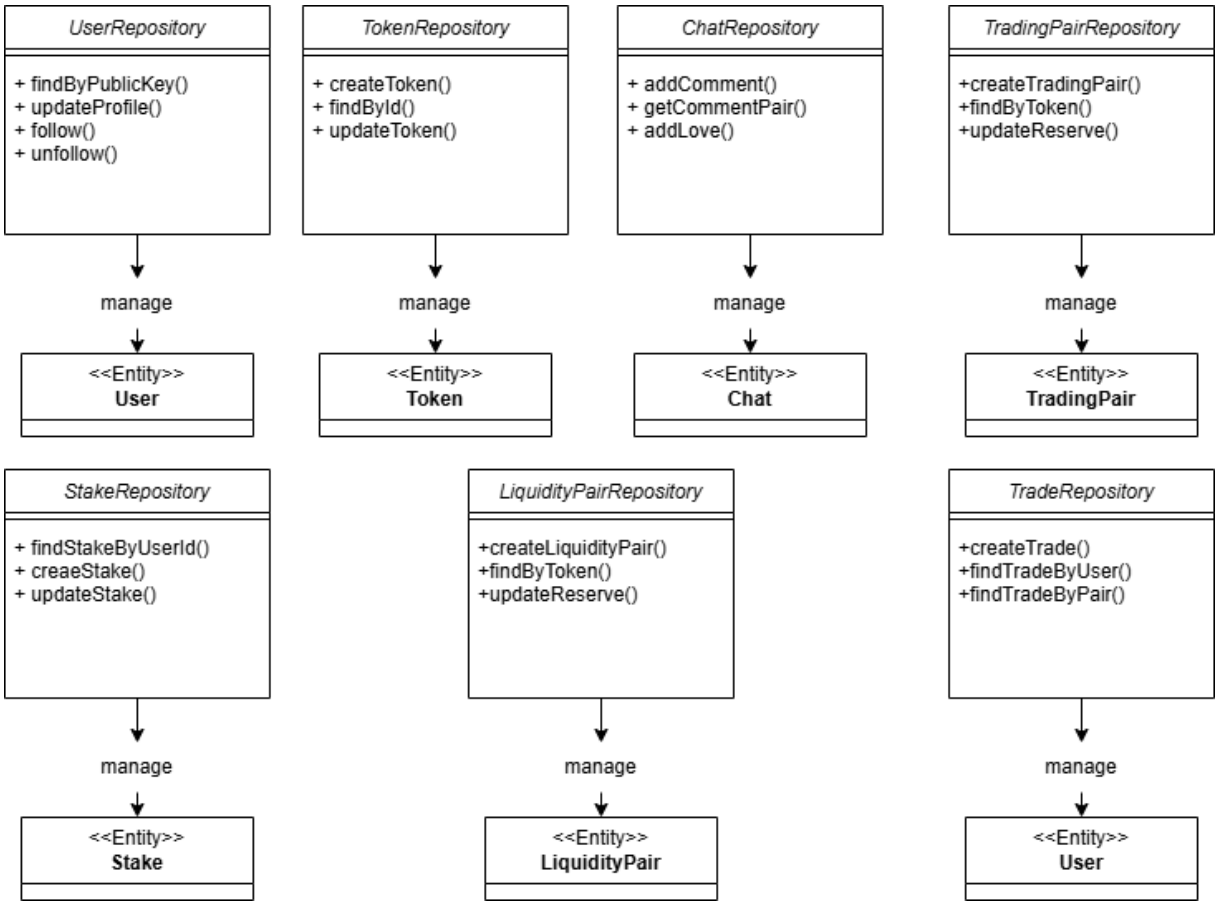
2.4 Thiết kế chi tiết

2.4.1 Thiết kế cơ sở dữ liệu

Từ quá trình phân tích ca sử dụng được đề cập ở mục 2.2.2, ta xác định biểu đồ thực thể liên kết và các lớp persistence như sau:



Hình 2.16: Biểu đồ thực thể liên kết.



Hình 2.17: Biểu đồ lớp persistence.

Từ đây, ta có thiết kế các bảng trong cơ sở dữ liệu:

Bảng Users

STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	publicKey	string	
3	name	string	
4	bio	string	
5	follower	objectid[]	reference to Users
6	following	objectid[]	reference to Users
7	tokenFollow	objectid[]	reference to Tokens
8	mentionReceived	number	
9	likeReceived	number	
10	createdAt	date	
11	updatedAt	date	

Bảng 2.9: Bảng Users

Bảng Tokens

STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	name	string	
3	symbol	string	
4	image	string	
5	type	string	
6	description	string	
7	totalSupply	string	
8	socialLinks	object	
9	contractAddress	string	
10	chainId	number	
11	createdAt	date	
12	updatedAt	date	

Bảng 2.10: Bảng Tokens

Bảng Chats

STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	creator	string	
3	parent	ObjectId	reference to Chats
4	liquidityPair	ObjectId	reference to LiquidityPairs
5	message	string	
6	loveCount	number	
7	attached	File[]	
8	createdAt	date	
9	updatedAt	date	

Bảng 2.11: Bảng Chats

Bảng Trading_Pairs

STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	creator	string	
3	TokenA	ObjectId	reference to Tokens
4	TokenB	ObjectId	reference to Tokens
5	tokenAReserve	string	
6	tokenBReserve	string	
7	poolAddress	string	
8	chainId	number	
9	totalValueLock	number	
10	totalLPToken	number	
11	poolFee	number	
12	createdAt	date	
13	updatedAt	date	

Bảng 2.12: Bảng Trading_Pairs

Bảng Liquidity_Pairs

STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	creator	string	
3	TokenA	ObjectId	reference to Tokens
4	tokenAReserve	string	
5	tokenBReserve	string	
6	poolAddress	string	
7	poolFee	number	
8	chainId	number	
9	createdAt	date	
10	updatedAt	date	

Bảng 2.13: Bảng Liquidity_Pairs

Bảng Stakes

STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	staker	string	
3	amount	string	
4	duration	number	
5	reward	string	
6	isActive	boolean	
7	txHash	string	
8	startTime	date	
9	endTime	date	

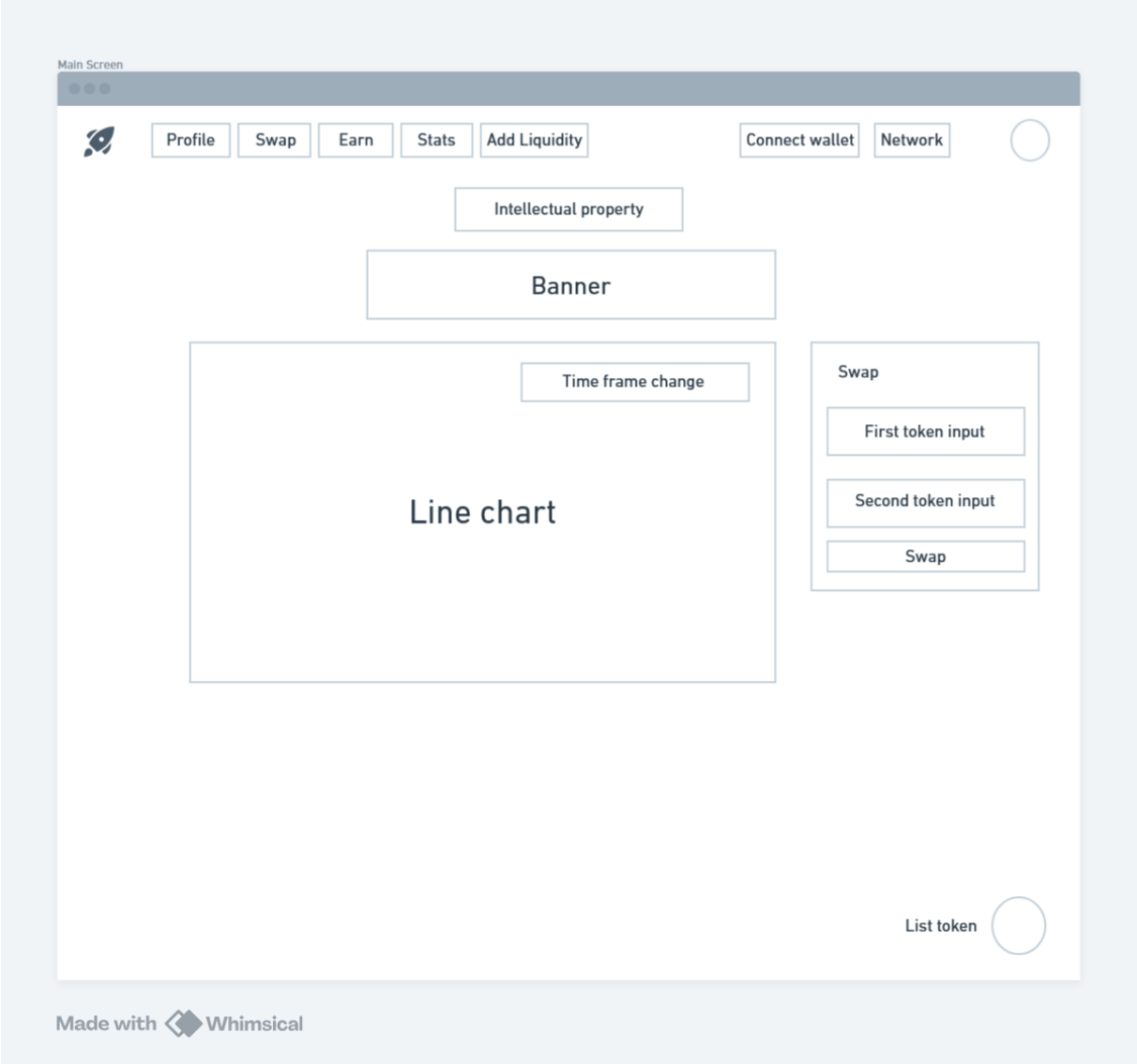
Bảng 2.14: Bảng Stakes

Bảng Trades

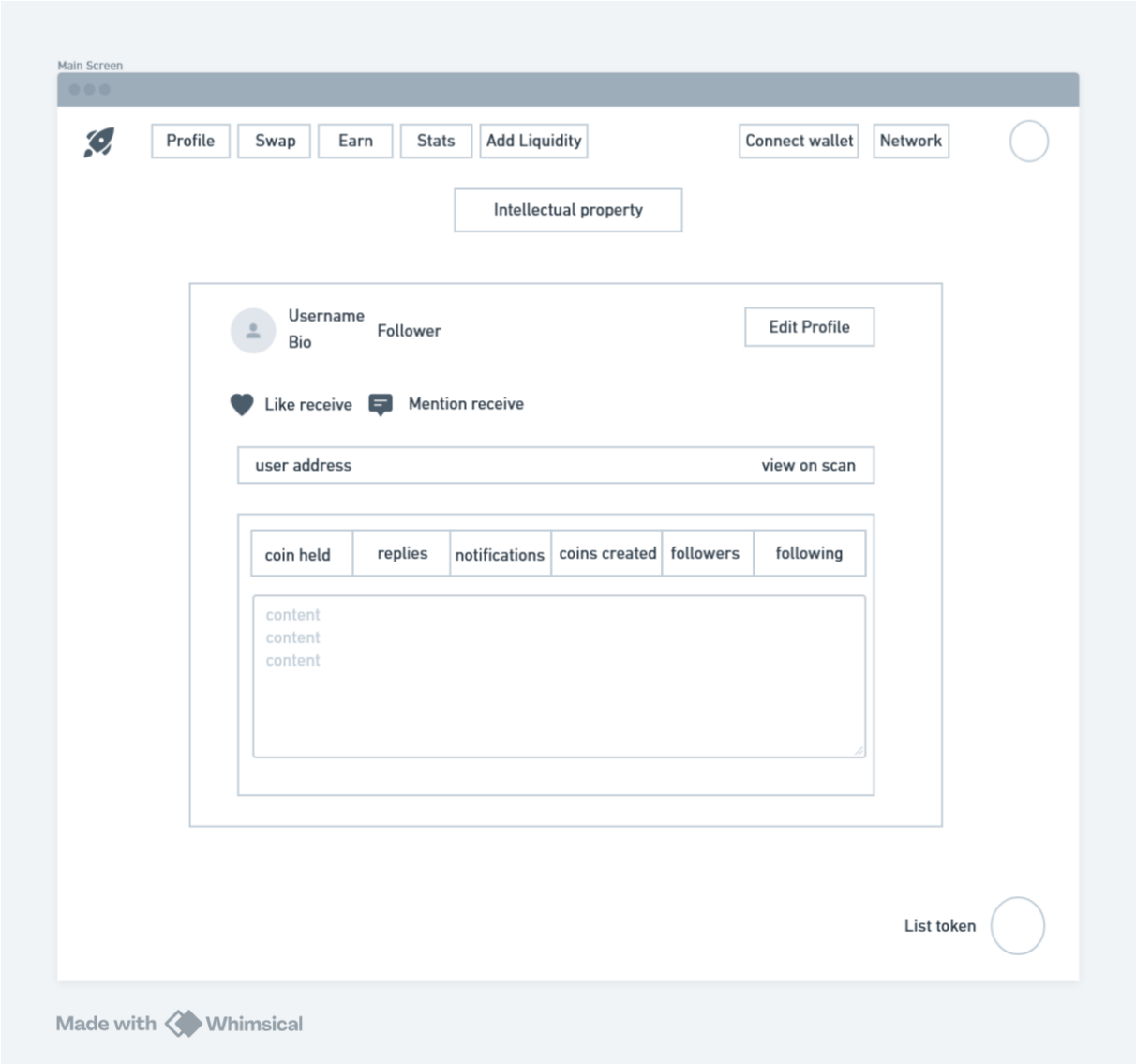
STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	type	string	
3	pairId	ObjectId	reference to LiquidityPairs || TradingPairs
4	tokenId	ObjectId	reference to Tokens
5	pairAddress	string	
6	pairType	string	
7	amount	number	
8	price	number	
9	txHash	string	
10	timestamp	date	

Bảng 2.15: Bảng Trades

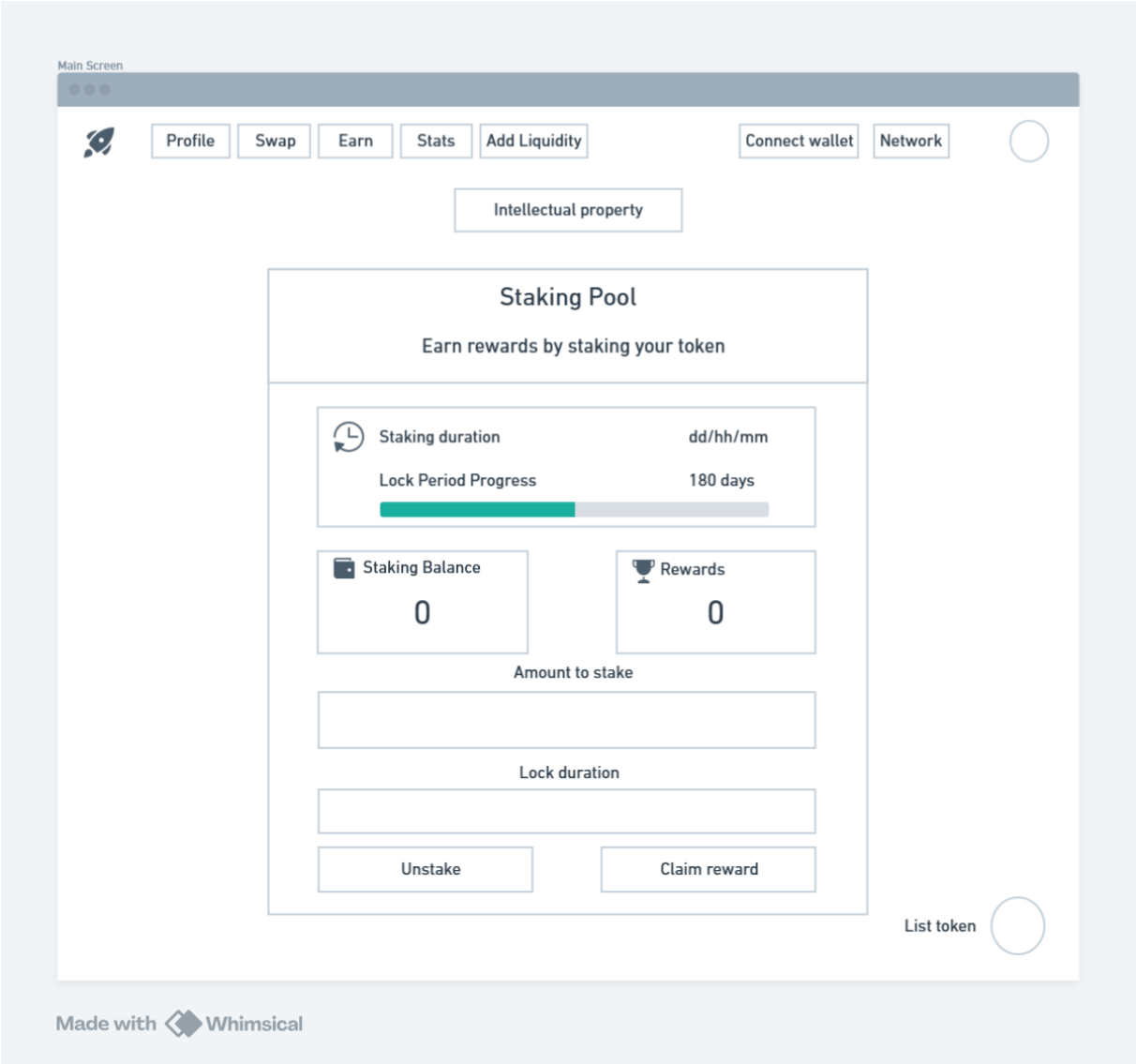
2.4.2 Thiết kế giao diện



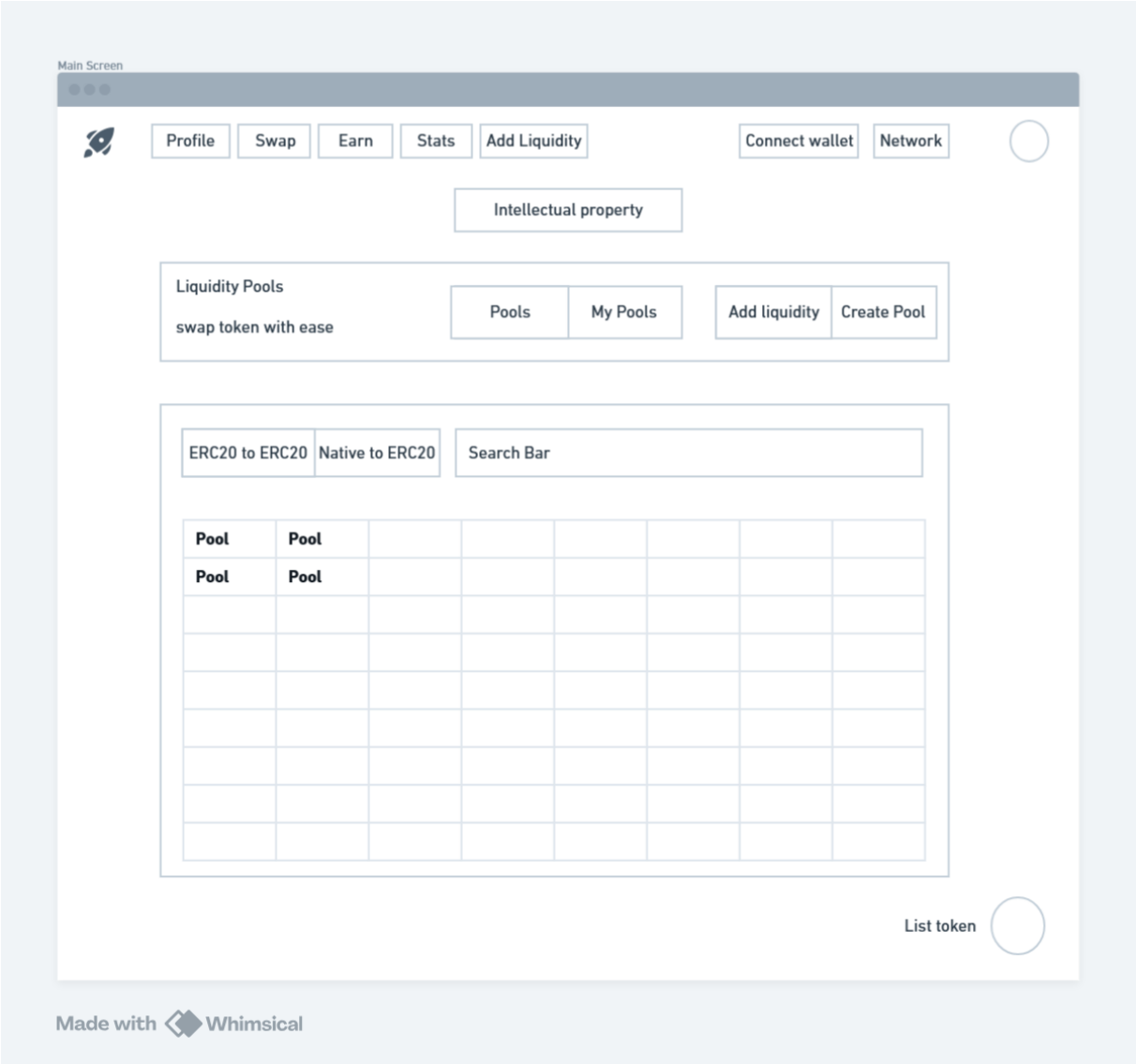
Hình 2.18: Thiết kế trang chủ.



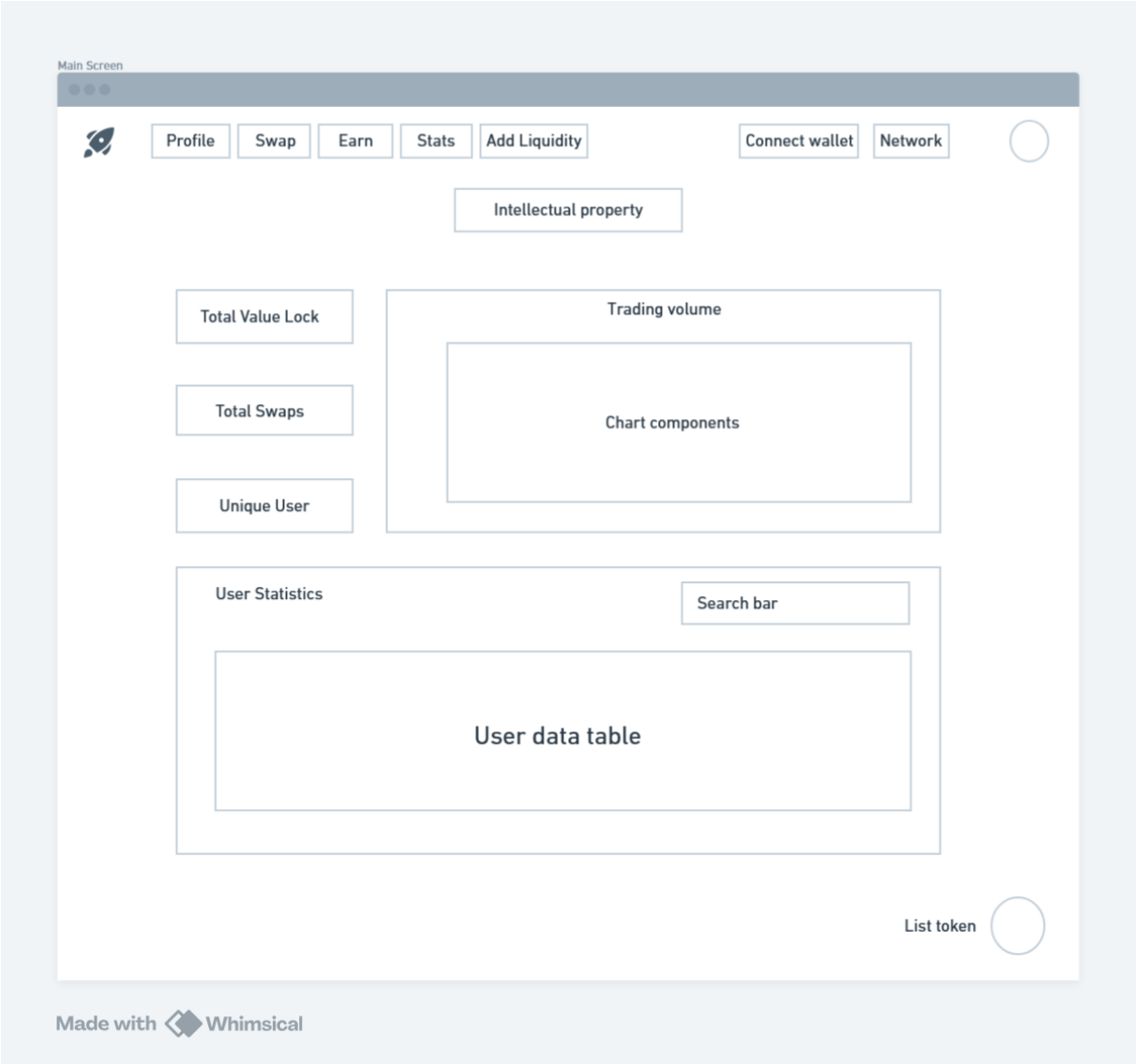
Hình 2.19: Thiết kế trang thông tin người dùng.



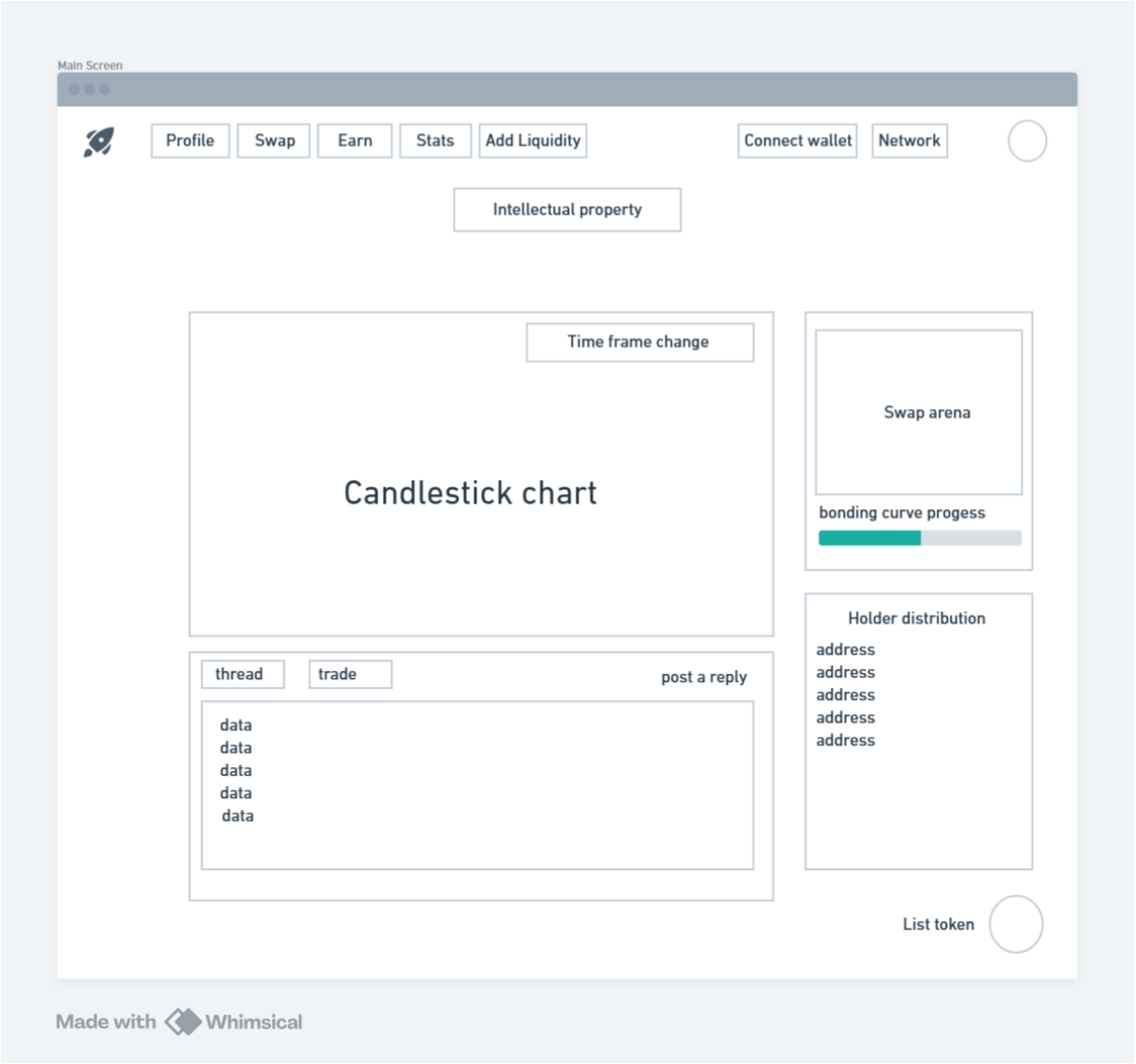
Hình 2.20: Thiết kế trang gửi tiết kiệm token.



Hình 2.21: Thiết kế trang hiển thị các cặp giao dịch.



Hình 2.22: Thiết kế trang thống kê thông số.



Hình 2.23: Thiết kế trang giao dịch token.

Chương 3

Cài đặt và thực nghiệm

Trong phần này, khóa luận tập trung vào việc trình bày quá trình phát triển cũng như các kết quả thu được từ việc kiểm thử và đánh giá các chức năng chính của nghiên cứu. Những hình ảnh minh họa được đưa vào để cung cấp một cái nhìn trực quan và cụ thể về các tính năng và hiệu suất của hệ thống được phát triển. Cùng với đó, các phương pháp kiểm thử sẽ được trình bày để minh họa quá trình đánh giá hiệu suất và độ tin cậy của hệ thống. Kết quả thu được sẽ được phân tích và đánh giá một cách cụ thể và có tính khách quan, từ đó đưa ra những nhận xét và kết luận chi tiết về hiệu suất và khả năng áp dụng của hệ thống trong thực tế. Điều này sẽ giúp hiểu rõ hơn về cách mà khóa luận này đóng góp vào lĩnh vực tương ứng cũng như tiềm năng phát triển trong tương lai.

3.1 Xây dựng ứng dụng

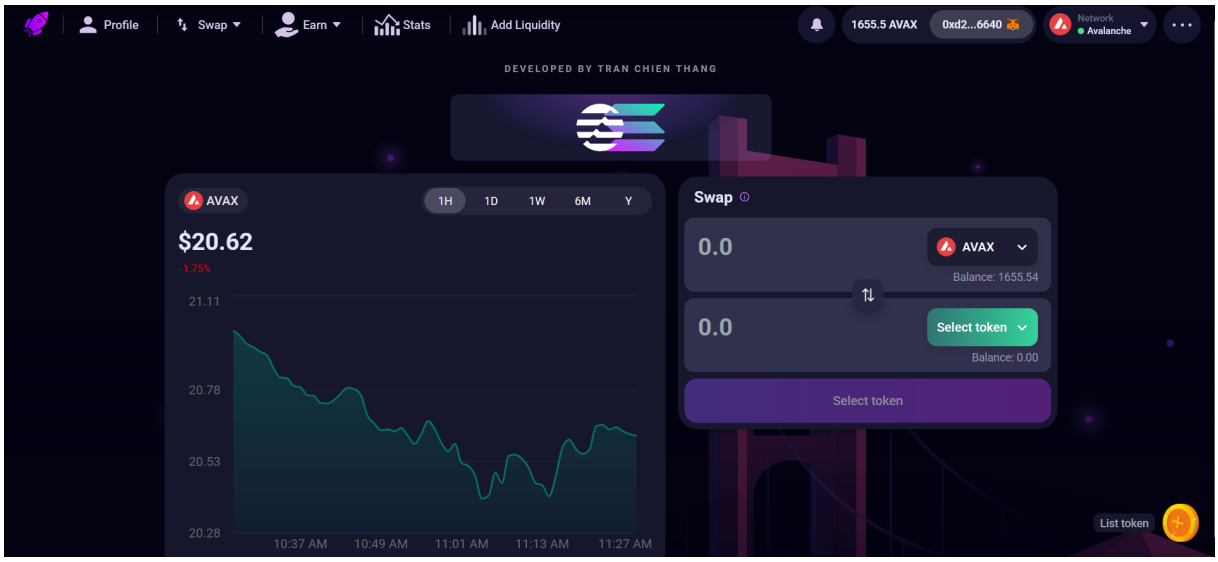
3.1.1 Thư viện và công cụ sử dụng

STT	Mục đích	Công cụ	Địa chỉ URL
1	Chỉnh sửa mã nguồn	Visual Studio Code	https://code.visualstudio.com
2	Cơ sở dữ liệu	MongoDB	https://www.mongodb.com
3	Kiểm thử API	Postman	https://www.postman.com
4	Lưu trữ mã nguồn	Github	https://github.com
5	Vẽ bảng biểu	Drawio, Mermaid, Swimlanes	https://app.diagrams.net https://mermaid.js.org https://swimlanes.io
6	Thiết kế wireframe	Whimsical	https://whimsical.com
7	Biên dịch Smart Contract	Foundry	https://book.getfoundry.sh
8	Kiểm thử Smart Contract	Foundry, Tenderly	https://book.getfoundry.sh https://tenderly.com

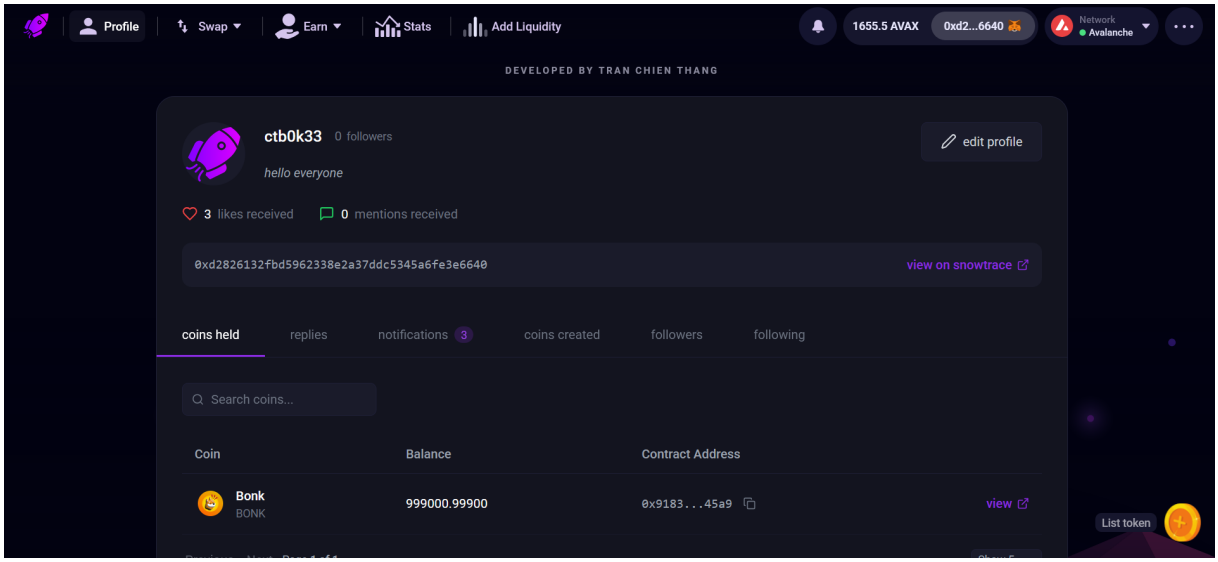
Bảng 3.1: Bảng tổng hợp các công cụ được sử dụng trong đề tài

3.1.2 Kết quả thực nghiệm

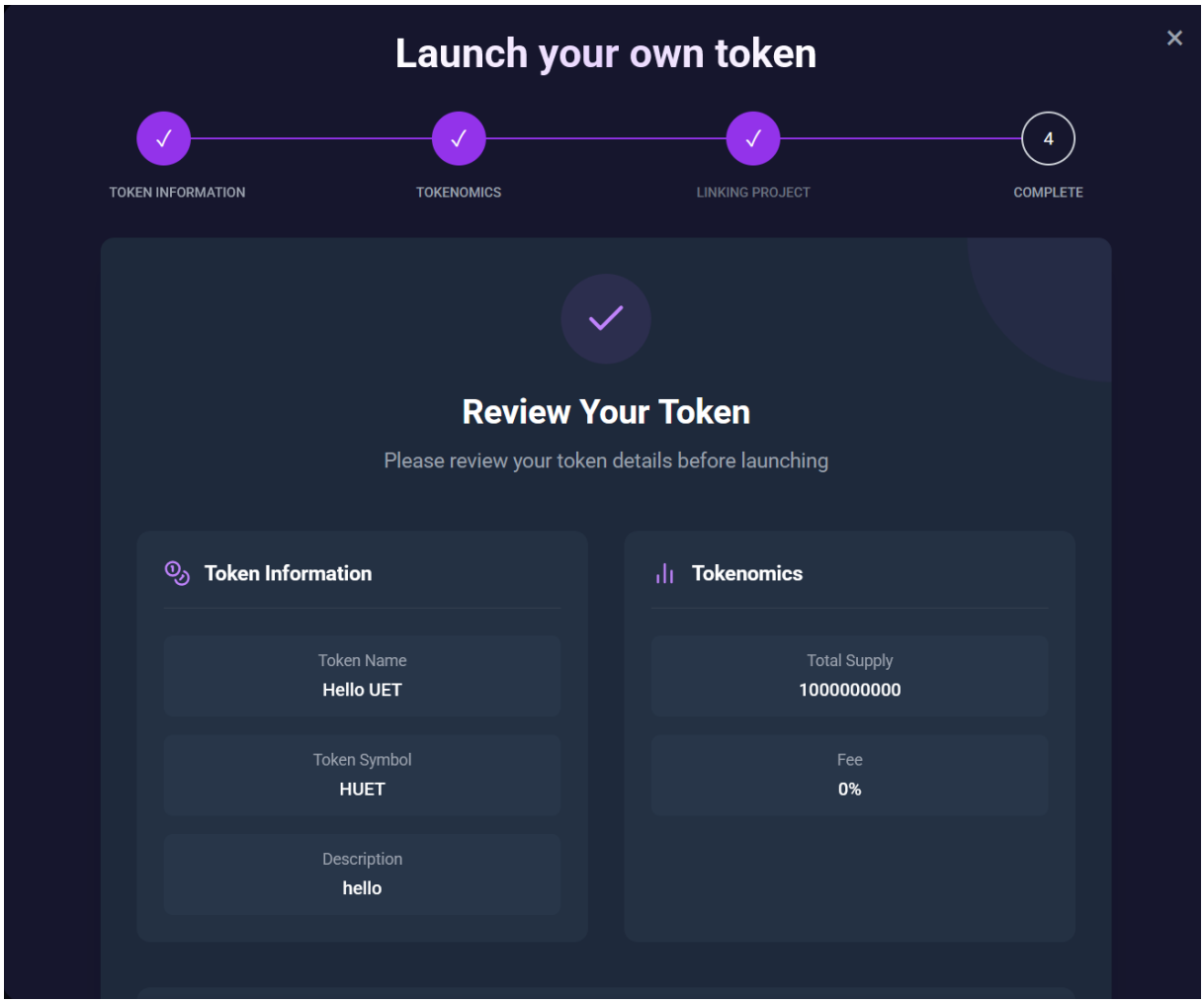
Sau đây là các hình ảnh của thực tế của ứng dụng web đã được phát triển được chạy trên trình duyệt Firefox (Windows 11). Với các thiết bị khác, giao diện sẽ co giãn phù hợp với kích thước màn hình nhưng lượng thông tin không thay đổi nhiều



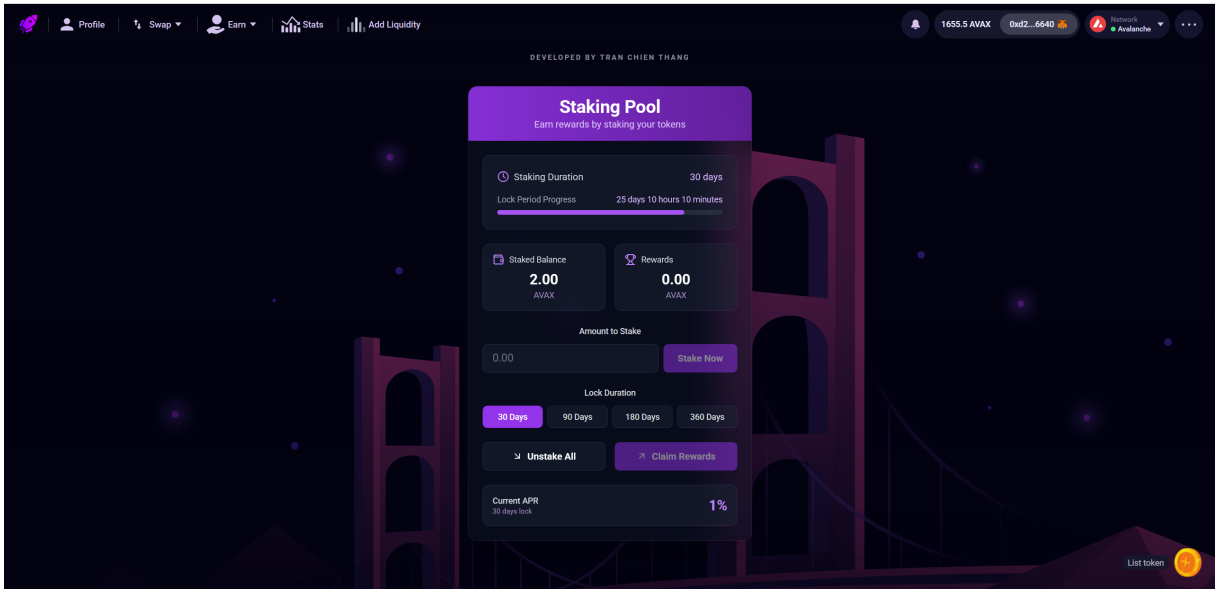
Hình 3.1: Giao diện màn hình chính của ứng dụng.



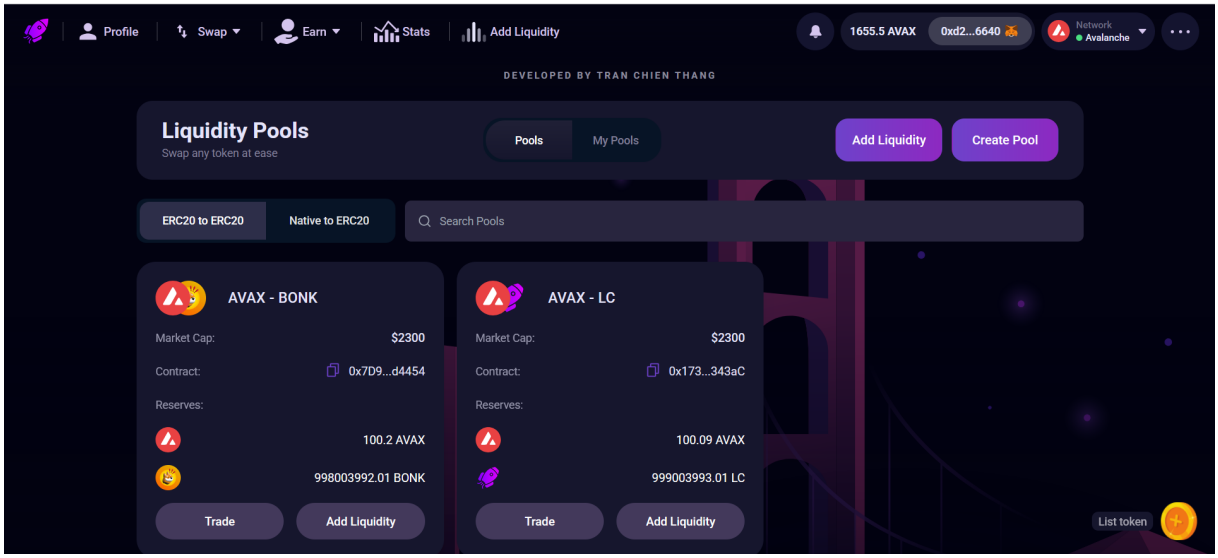
Hình 3.2: Giao diện chức năng quản lý thông tin cá nhân.



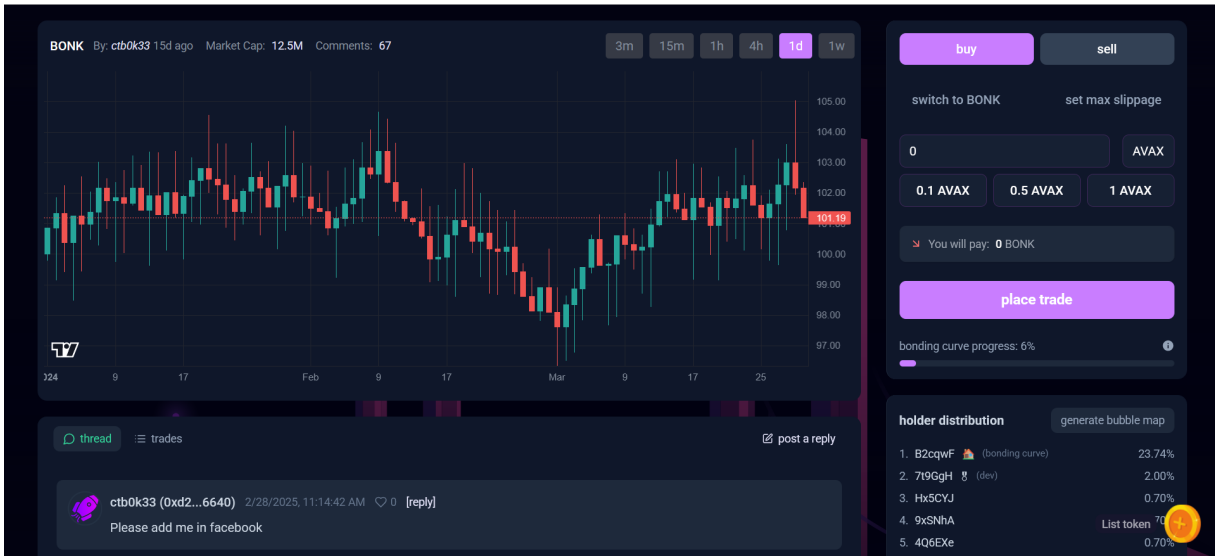
Hình 3.3: Giao diện chức năng tạo token mới.



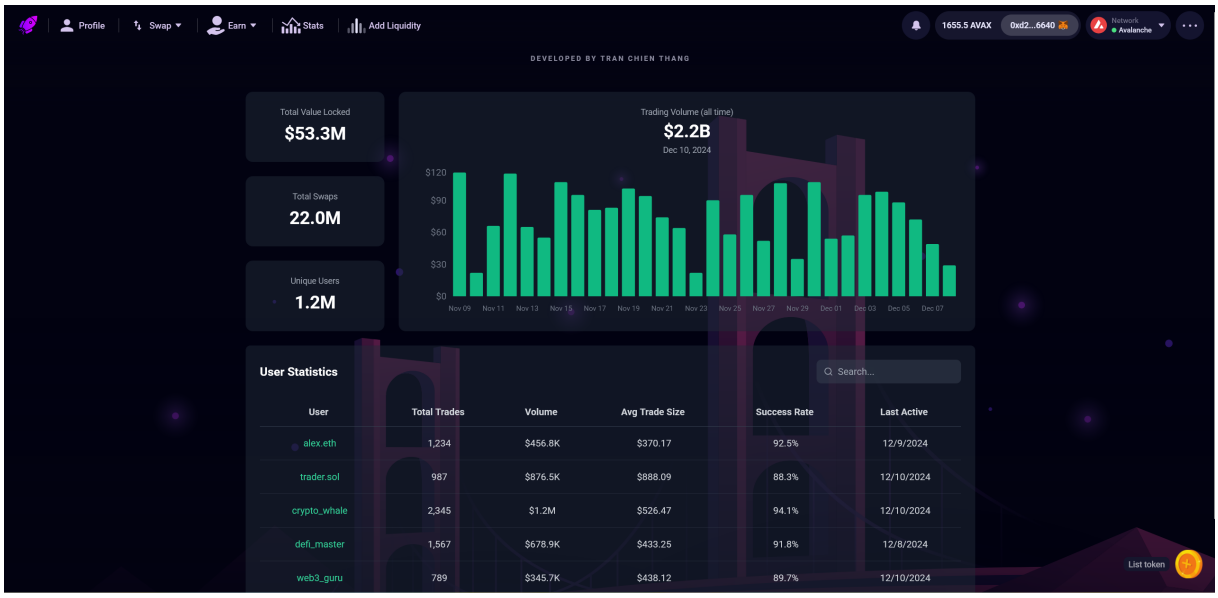
Hình 3.4: Giao diện chức năng gửi tiết kiệm token.



Hình 3.5: Giao diện chức năng thao tác với các cặp giao dịch.



Hình 3.6: Giao diện màn chức năng giao dịch token.



Hình 3.7: Giao diện màn hình thống kê thông số của ứng dụng.

3.2 Kiểm thử

Khóa luận sẽ sử dụng các bộ test được sinh bởi hai kỹ thuật kiểm thử hộp đen là phân hoạch tương đương và phân tích giá trị biên để thực hiện kiểm thử các chức năng. Tại mỗi ca kiểm thử, khóa luận sẽ kiểm tra trạng thái trả về của API, kết quả trả về từ hợp đồng thông minh. Thời gian phản hồi dưới 2 giây để thỏa mãn yêu cầu phi chức năng tương ứng. Để thống nhất với quá trình phát triển, khóa luận áp dụng kỹ thuật kiểm thử TDD (Test Driven Development). Theo đó, khóa luận sẽ thiết kế các API, viết các bộ test tương ứng cho hợp đồng thông minh và thực hiện việc phát triển à kiểm thử song song. Quá trình kiểm thử sẽ được diễn ra tự động bằng cách sử dụng ứng dụng POSTMAN cũng như thư viện foundry và tenderly trong quá trình phát triển.

Tính năng	**Ngữ cảnh**	**Kết quả mong muốn**
Đăng nhập	1) Kết nối ví metamask	1) Giao diện thành công hiển thị địa chỉ ví và số dư của người dùng 2) Người dùng có thể sử dụng tất cả các tính năng còn lại của ứng dụng
Quản lý tài khoản	1) Người dùng xem thông tin tài khoản cá nhân.	1) Hệ thống hiển thị chính xác thông tin các token người dùng đang sử hữu 2) Hệ thống hiển thị chính xác tên, bio, ảnh đại diện của người dùng
Cập nhật thông tin cá nhân	1) Người dùng cập nhật thông tin bio, ảnh đại diện	1) Hệ thống thành công cập nhật thông tin cá nhân

Tạo token	1) Người dùng bấm vào "List token" 2) Người dùng điền đầy đủ thông tin về token 3) Người dùng chọn "confirm"và xác nhận giao dịch trên metamask	1) Token được tạo thành công trên chuỗi khối 2) Cặp giao dịch mới được tạo thành công trên chuỗi khối 3) Token và cặp giao dịch mới hiển thị trên hệ thống với thông số chính xác
Tạo token (số dư không đủ)	Tương tự như ca kiểm thử "Tạo token"	1) Hệ thống hiển thị thông báo tạo token thất bại
Giao dịch token	1) Người dùng bấm vào "Swap" 2) Người dùng chọn cặp giao dịch mong muốn 3) Người dùng chọn thực hiện giao dịch mua hoặc bán token	1) Giao dịch thành công được ghi lại trên chuỗi khối 2) Số dư trong ví của người dùng cập nhật thành công 3) Thông tin về cặp giao dịch như số lượng token còn lại trong pool, giá token thay đổi chính xác. 4) Thông tin giao dịch được ghi lại và hiển thị chính xác trên hệ thống

Giao dịch token (số dư không đủ)	Tương tự như ca kiểm thử "Giao dịch token"	1) Giao diện hiển thị giao dịch thất bại
Giao dịch token (Không đủ thanh khoản trong pool)	Tương tự như ca kiểm thử "Giao dịch token"	1) Giao diện hiển thị thông báo "Not enough reserve in pool"
Gửi tiết kiệm token	1) Người dùng bấm vào "Earn", xong đó bấm vào "stake" 2) Người dùng nhập số lượng token muốn gửi tiết kiệm và thời gian gửi tiết kiệm 3) Người dùng chọn "stake"và xác nhận giao dịch	1) Giao dịch thành công được ghi lại trên chuỗi khối 2) Số dư trong ví của người dùng cập nhật thành công 3) Thông tin về stake của user hiển thị trên màn hình chính xác. 4) Thông tin stake được ghi lại chính xác trên hệ thống
Gửi tiết kiệm token (chưa rút khoản gửi tiết kiệm cũ)	Tương tự như ca kiểm thử "Gửi tiết kiệm token"	1) Giao diện hiển thị thông báo gửi tiết kiệm thất bại
Gửi tiết kiệm token (số dư không đủ)	Tương tự như ca kiểm thử "Gửi tiết kiệm token"	1) Giao diện hiển thị thông báo gửi tiết kiệm thất bại

Bình luận	1) Người dùng bấm vào "Swap" 2) Người dùng chọn cặp giao dịch mong muốn 3) Người dùng để lại bình luận	1) Giao diện hiển thị bình luận thành công
Yêu thích bình luận	1) Người dùng bấm vào "Swap" 2) Người dùng chọn cặp giao dịch mong muốn 3) Người dùng thích 1 bình luận của người dùng khác	1) Giao diện hiển thị ghi nhận thích bình luận thành công

Bảng 3.2: Các trường hợp kiểm thử.

Tính năng kiểm thử	**Kết quả**	**Chú thích**
Đăng nhập	Đạt	Kiểm thử UI và API
Quản lý tài khoản	Đạt	Kiểm thử UI và API
Tạo token	Đạt	Kiểm thử UI, API và Smart contract
Giao dịch token	Đạt	Kiểm thử UI, API và Smart contract
Gửi tiết kiệm token	Đạt	Kiểm thử UI, API và Smart contract
Rút tiết kiệm token	Đạt	Kiểm thử UI, API và Smart contract
Bình luận	Đạt	Kiểm thử UI và API

Bảng 3.3: Kết quả kiểm thử chức năng

3.3 Đánh giá hiệu năng và phương hướng cải thiện của hệ thống

Mặc dù các thử nghiệm đều chứng minh rằng hệ thống được phát triển thỏa mãn và đáp ứng được tối thiểu các yêu cầu về chức năng và phi chức năng

đã đặt ra. Tuy nhiên, trong môi trường thực tế, sẽ cần nhiều sự cải thiện để hệ thống có thể hoạt động mượt mà và ổn định. Điều này vô cùng quan trọng vì khóa luận tập trung phát triển một nền tảng cho phép người dùng giao dịch giữa các loại token. Điều này sẽ góp phần làm giảm tối thiểu các giao dịch với độ trượt giá cao, một điều rất quan trọng trong mọi hệ thống giao dịch.

Để nâng cao hiệu suất của hệ thống, khóa luận đặt ra các hướng phát triển trong tương lai bao gồm: *Áp dụng kỹ thuật caching, xây dựng dịch vụ worker riêng để quản lý giao dịch và phân bổ dịch vụ theo chiều rộng.*

Áp dụng hệ thống caching sẽ giúp cải thiện hiệu suất của hệ thống, đặc biệt với những truy vấn offchain như lấy thông tin người dùng hay lấy thông tin về các cặp giao dịch. Việc caching cũng làm giảm tải cho cơ sở dữ liệu, hạn chế các truy vấn trực tiếp đến cơ sở dữ liệu chính. Việc lưu trữ dữ liệu trong bộ nhớ cũng giảm dữ liệu cần truyền qua mạng, tăng tính sẵn sàng của dữ liệu trong hệ thống.

Đối với mô đun giao dịch, việc xây dựng một dịch vụ worker riêng hoặc sử dụng worker từ các provider uy tín như quicknode là cực kỳ quan trọng. Khi hệ thống mở rộng và số lượng các cặp giao dịch tăng lên, việc truy vấn cơ sở dữ liệu về giao dịch liên tục để xây dựng mô hình nến cho từng cặp giao dịch ở các timeframe khác nhau như khóa luận đang triển khai là vô cùng tốn kém. Điều này sẽ có thể gây ra các vấn đề tiềm ẩn về hiệu năng của hệ thống khi số lượng cặp giao dịch tăng cao.

Cuối cùng, khóa luận đã triển khai chỉ sử dụng một dịch vụ chạy trên một máy chủ duy nhất, điều này không thể nào đáp ứng được nếu lượng người dùng và dữ liệu lớn phân bố tại các khu vực. Trong tương lai, lượng người dùng có thể tăng do vậy không thể nào áp dụng việc nâng cao cấu hình máy chủ và cơ sở dữ liệu vì nó sẽ dần không thể đáp ứng được nhu cầu thực tế của người dùng. Đây là lúc việc triển khai nên suy nghĩ đến việc mở

rộng các dịch vụ theo chiều rộng. Tức là thay vì nâng cao phần cứng để đáp ứng tạm thời với lượng người dùng tăng thì hoàn toàn có thể triển khai thêm một máy chủ với cùng một dịch vụ tương ứng và thực hiện cân bằng tải giữa các dịch vụ này. Việc áp dụng kiến trúc vi dịch vụ trong hệ thống đã triển khai giúp việc mở rộng các dịch vụ trở nên dễ dàng hơn vì không phải dịch vụ nào cũng cần phải mở rộng. Theo đó, trong khóa luận đã phát triển, dịch vụ giao dịch token là nơi chịu tải nhiều nhất vì hầu hết các chức năng đều chủ yếu xoay quanh dịch vụ này. Do vậy, nếu lượng người dùng trong tương lai tăng, dịch vụ giao dịch token là nơi cần xem xét việc mở rộng và phân bổ máy chủ mới để đáp ứng lượng yêu cầu mới.

KẾT LUẬN

Phương hướng phát triển trong tương lai

REFERENCES

[1] Robert France **and** Bernhard Rumpe. **?**Model-driven Development of Complex Software: A Research Roadmap**? in***Future of Software Engineering (FOSE '07)***:** 2007, **pages** 37–54. DOI: 10.1109/FOSE.2007.14.