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

GLOSARY 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) [1] 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 thesis, we extend OCL to support the specification of temporal properties and event-based behaviors, aiming to enhance its expressiveness in modeling dynamic system aspects. Our approach is implemented within the UML-based Specification Environment (USE) [**USE**], a tool that supports specification, and validation of software systems using UML and OCL. USE allows modelers to work with various UML diagrams and define constraints through invariants and pre-/postconditions. It also provides validation and verification capabilities via the model validator component, which translates models and properties into relational logic using Kodkod. As a realization of this thesis, we worked on a plugin for USE that called USE-TemporalOCL, which allows users to specify temporal properties and events in OCL.

The thesis is structured as follows:

- **Chapter 1**: This chapter lays the foundation for the background of this thesis. We explore theoretical concepts and tools that are used in this thesis.
- **Chapter 2**: This chapter presents our OCL extension to specify temporal properties and events.
- **Chapter 3**: This chapter describes the implementation and evaluation of the USE-TemporalOCL plugin.
- **Conclusion**: This chapter summarizes the contributions of this thesis and discusses future work.

Each chapter starts with an *Introduction* section, then ends with a *Summary* section.

Chương 1

FOUNDATIONAL KNOWLEDGE

1.1 Introduction

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

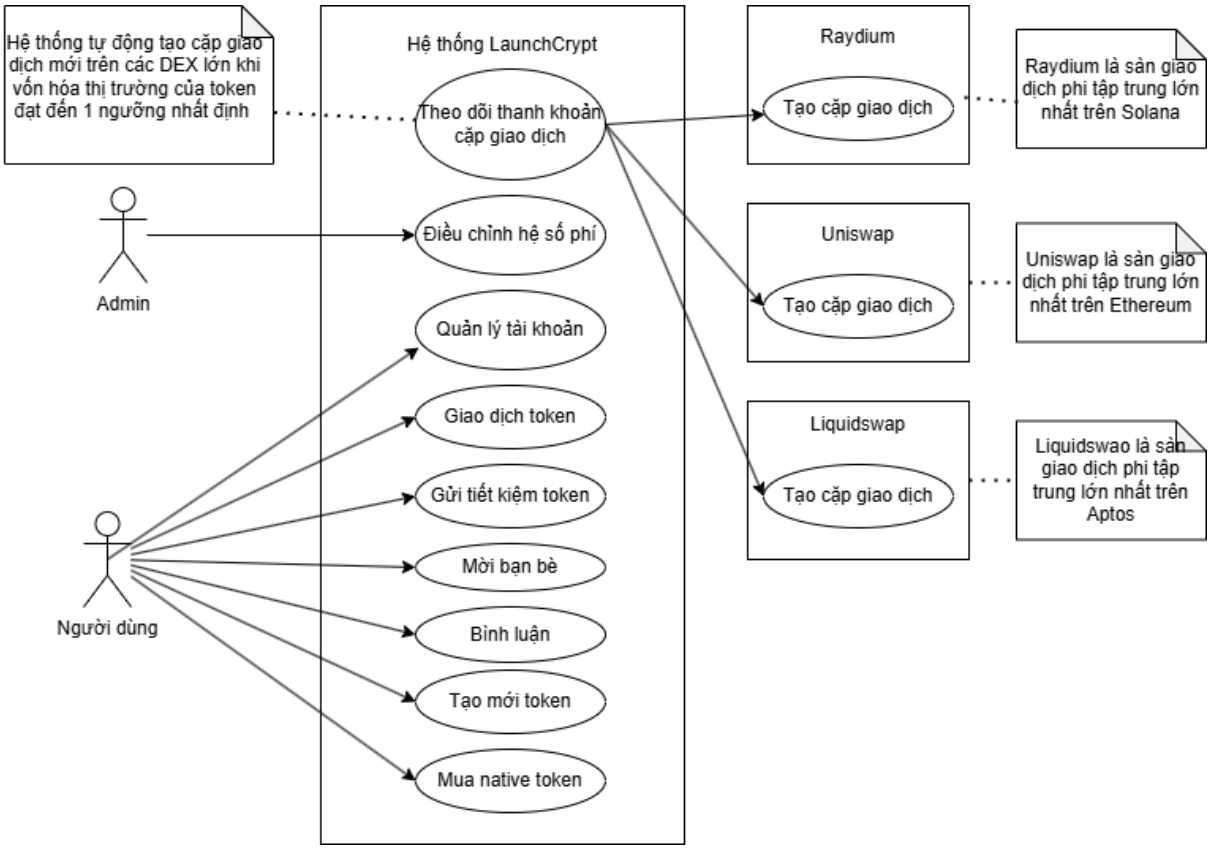
Trong bối cảnh tài chính phi tập trung (DeFi) đang phát triển mạnh mẽ, nhu cầu về một nền tảng toàn diện cho phép người dùng dễ dàng tạo, quản lý và giao dịch token trên nhiều chuỗi khối khác nhau đang ngày càng tăng cao. Hiện tại, việc phát hành và quản lý token trên các blockchain như

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.

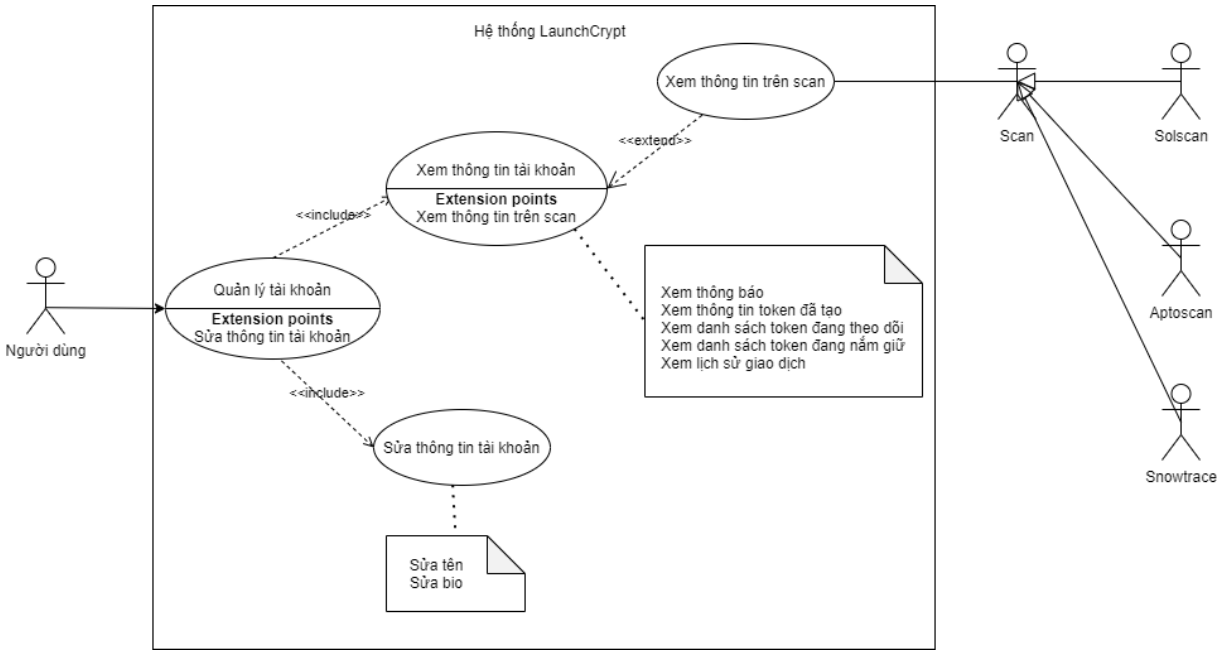
Các tác nhân tham gia trong biểu đồ:

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.

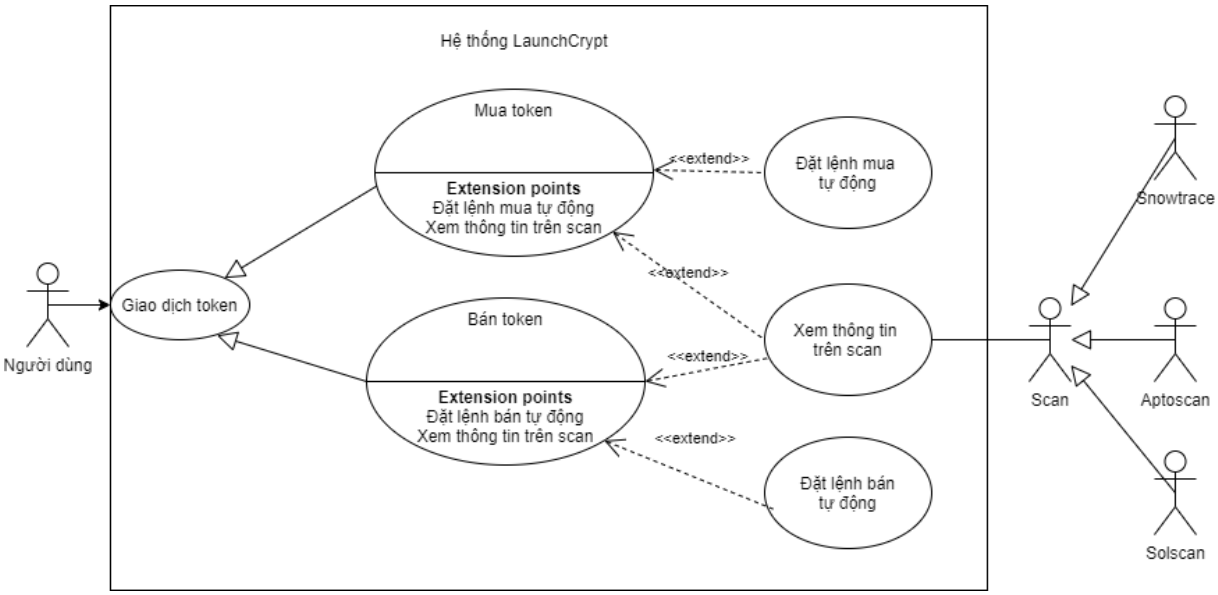
4. **CoinBase**: Là sàn giao dịch tập trung được dùng phổ biến, được sử dụng khi người dùng có nhu cầu mua native token của các chuỗi khối layer 1.



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

Tác nhân: Người dùng

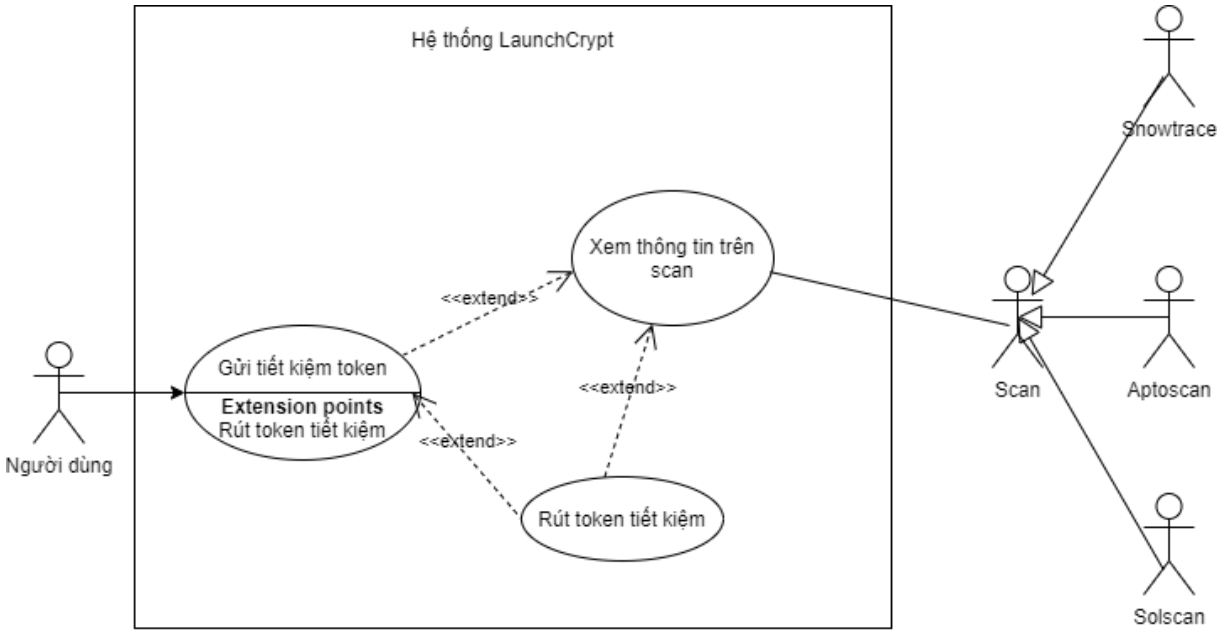
Mô tả: Người dùng xem thông tin cá nhân và chỉnh sửa thông tin bao gồm tên và tiểu sử (bio). Ngoài ra, người dùng có thể xem thông tin về ví của mình trên scan.



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

Tác nhân: Người dùng

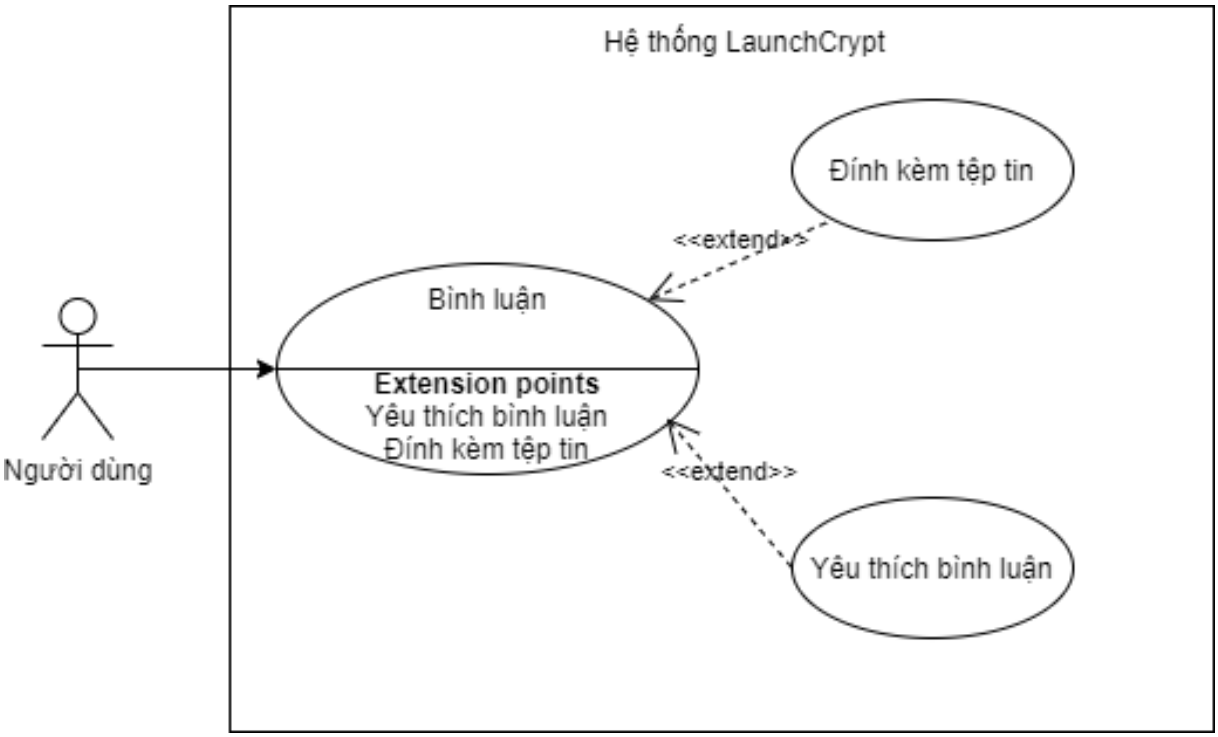
Mô tả: Người dùng giao dịch, mua bán, trao đổi giữa các token được tạo trên hệ thống và native token trên nền tảng LaunchCrypt. Ngoài ra, người dùng có thể đặt các lệnh mua, bán tự động hoặc xem thông tin đầy đủ về giao dịch trên trang scan.



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

Tác nhân: Người dùng

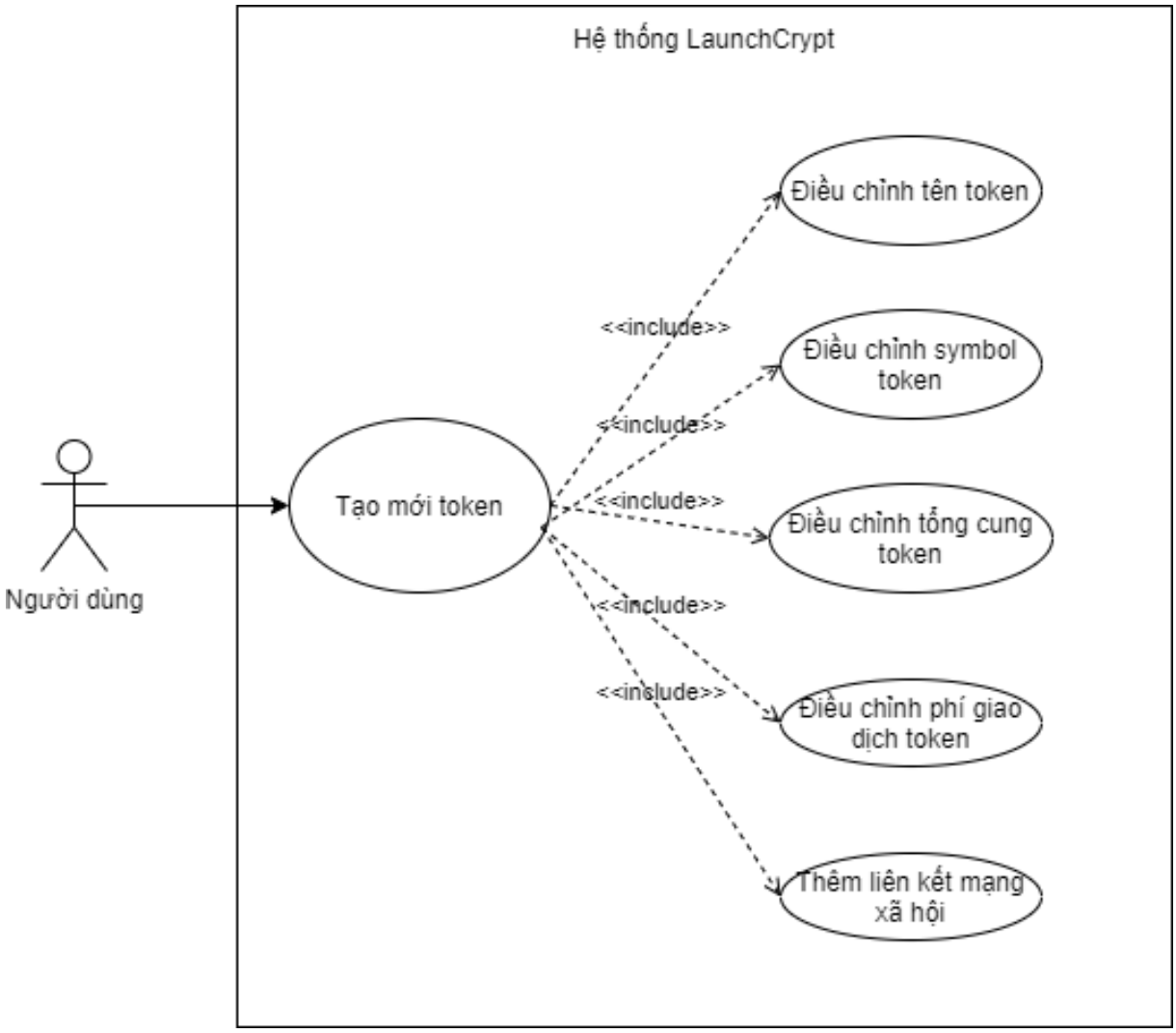
Mô tả: Người dùng gửi tiết kiệm để nhận về 1 khoản lãi suất theo tỷ lệ đã quy định trên nền tảng LaunchCrypt. Người dùng có thể rút tiền gửi tiết kiệm khi đạt đủ mốc thời gian và xem thông tin chi tiết về các giao dịch trên trang scan.



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

Tác nhân: Người dùng

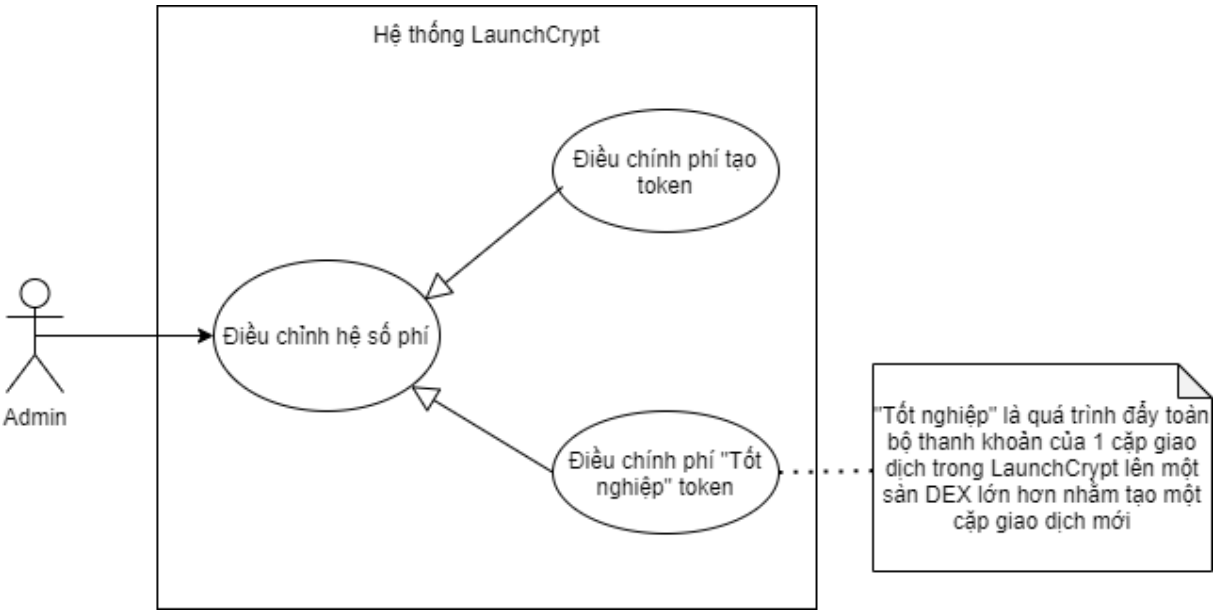
Mô tả: Khi người dùng xem thông tin giao dịch của 1 token, họ có thể để lại bình luận hoặc trao đổi về thông tin token đó với những người dùng khác. Người dùng cũng có thể yêu thích các bình luận, trả lời các bình luận của người dùng khác hoặc đính kèm tệp tin trong bình luận của mình.



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.

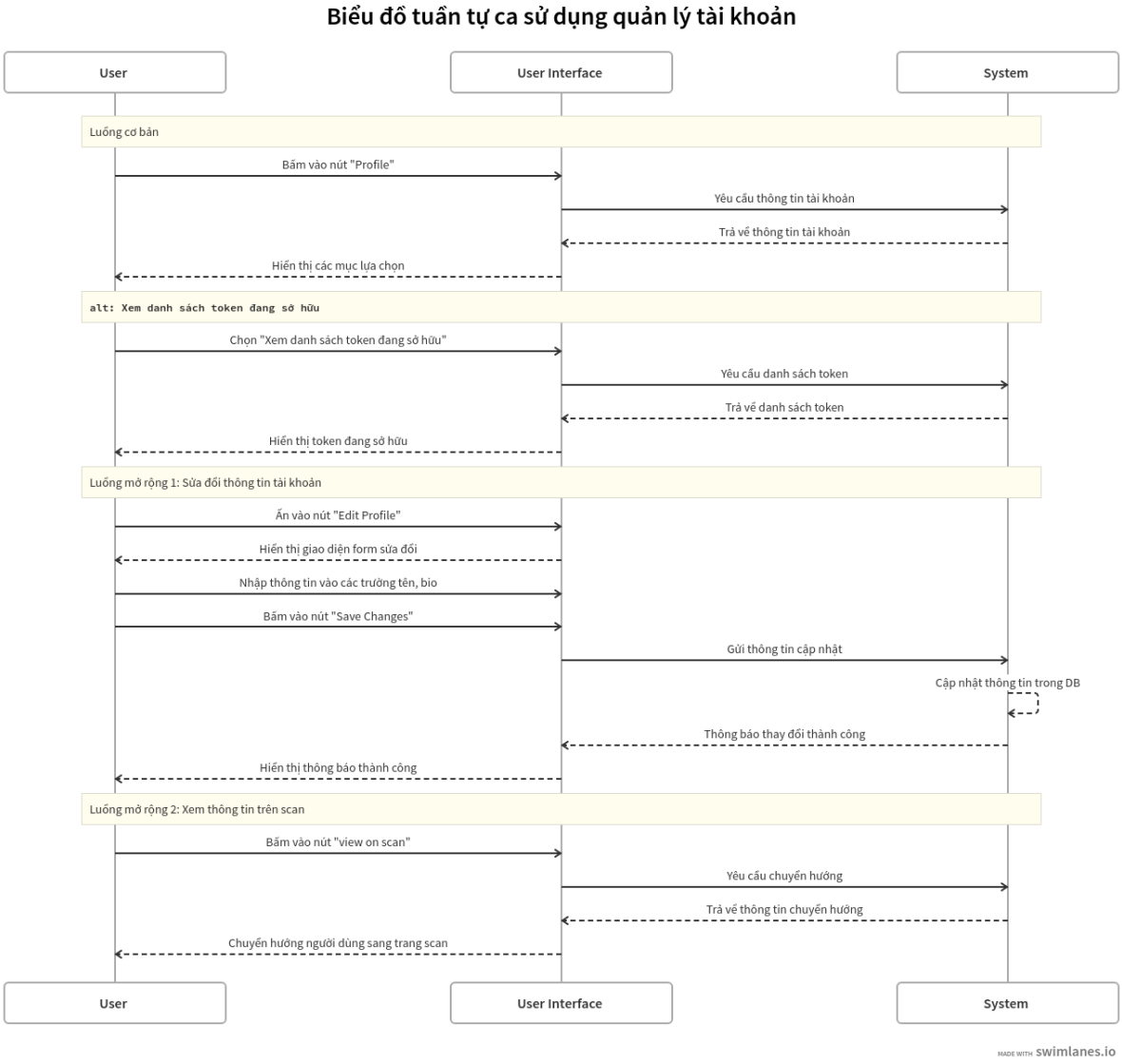
Tác nhân: Admin

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, ...

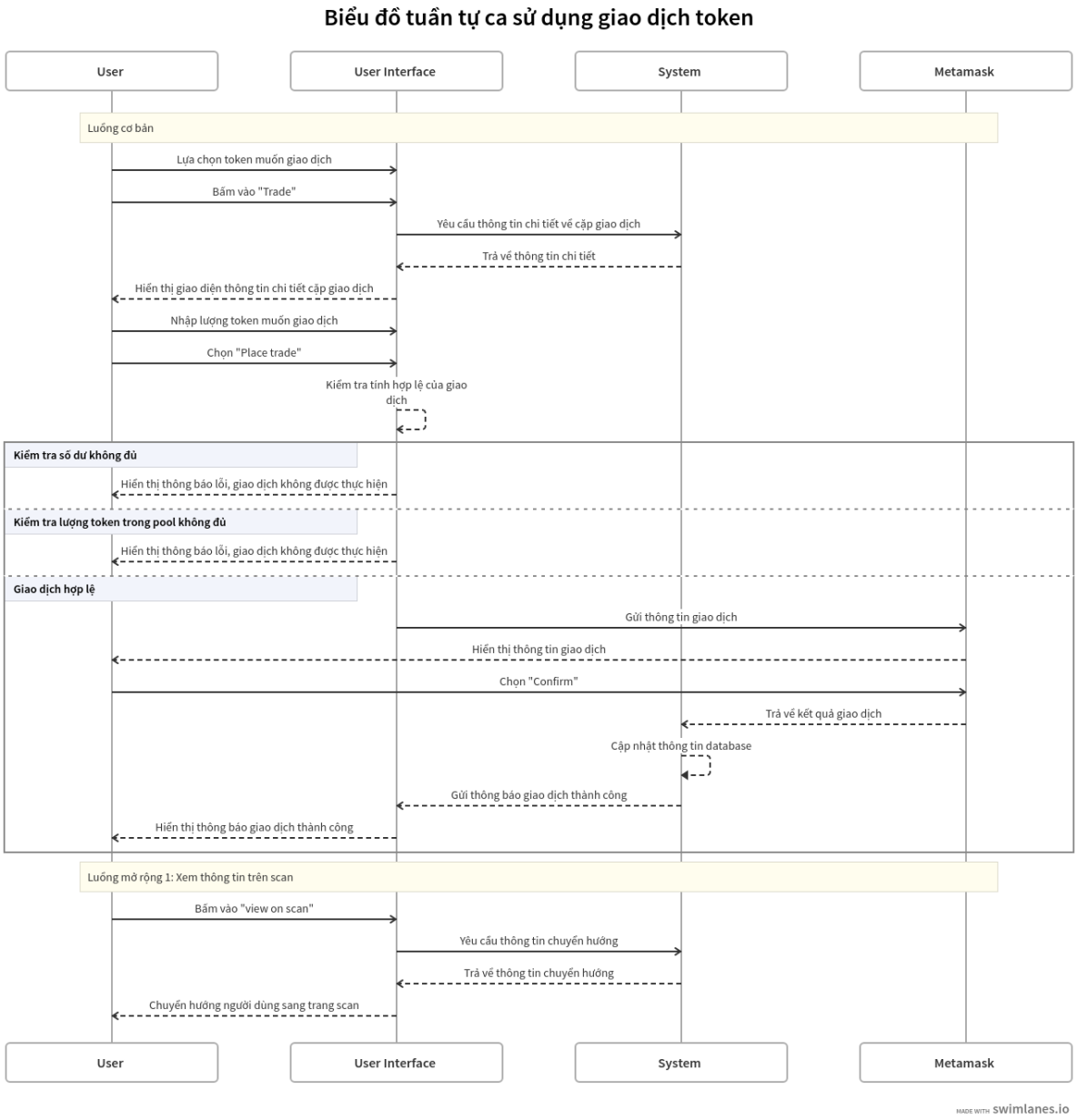
Bảng 2.1: Đặc tả ca sử dụng quản lý tài khoản.



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

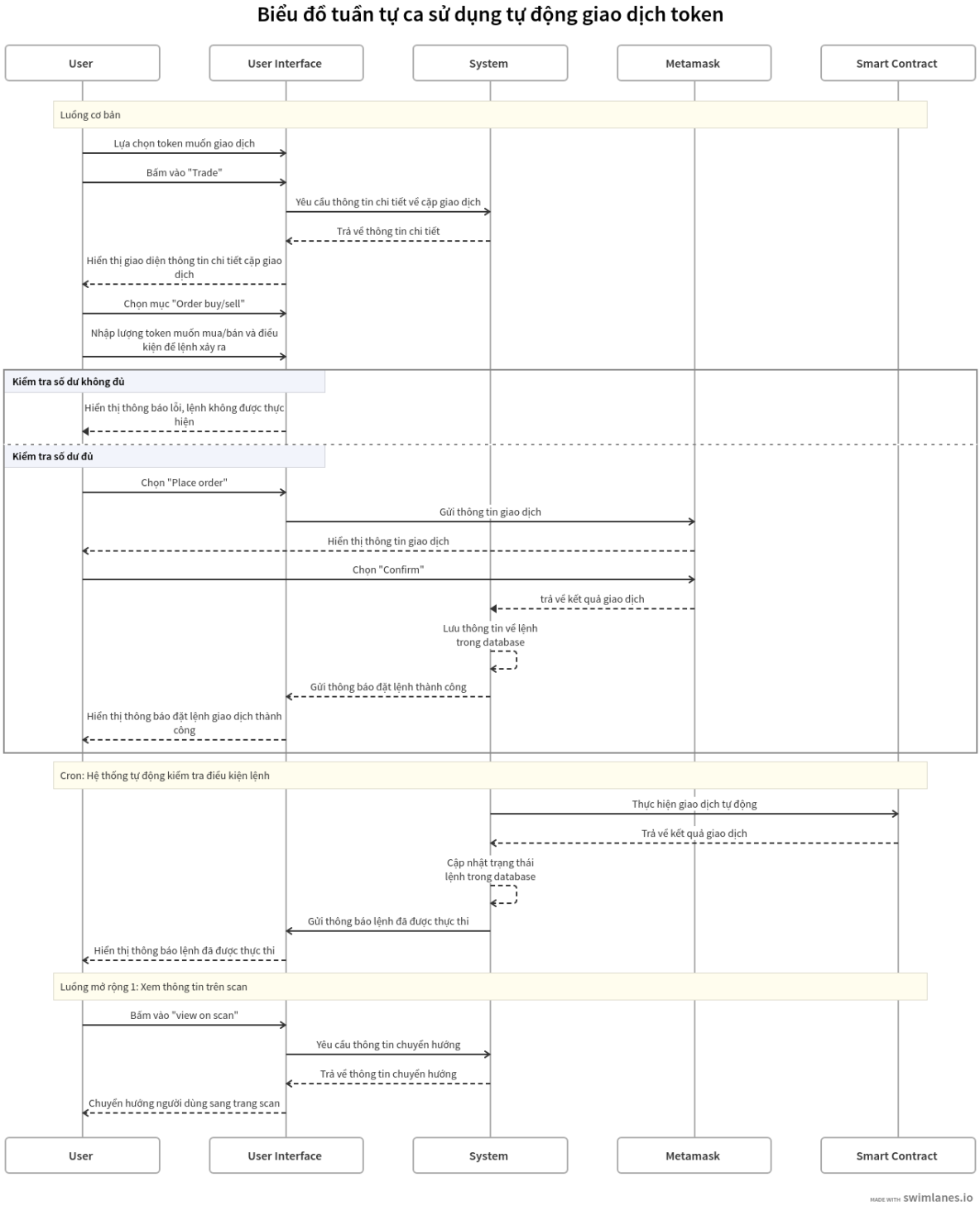
Bảng 2.2: Đặc tả ca sử dụng giao dịch token



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.

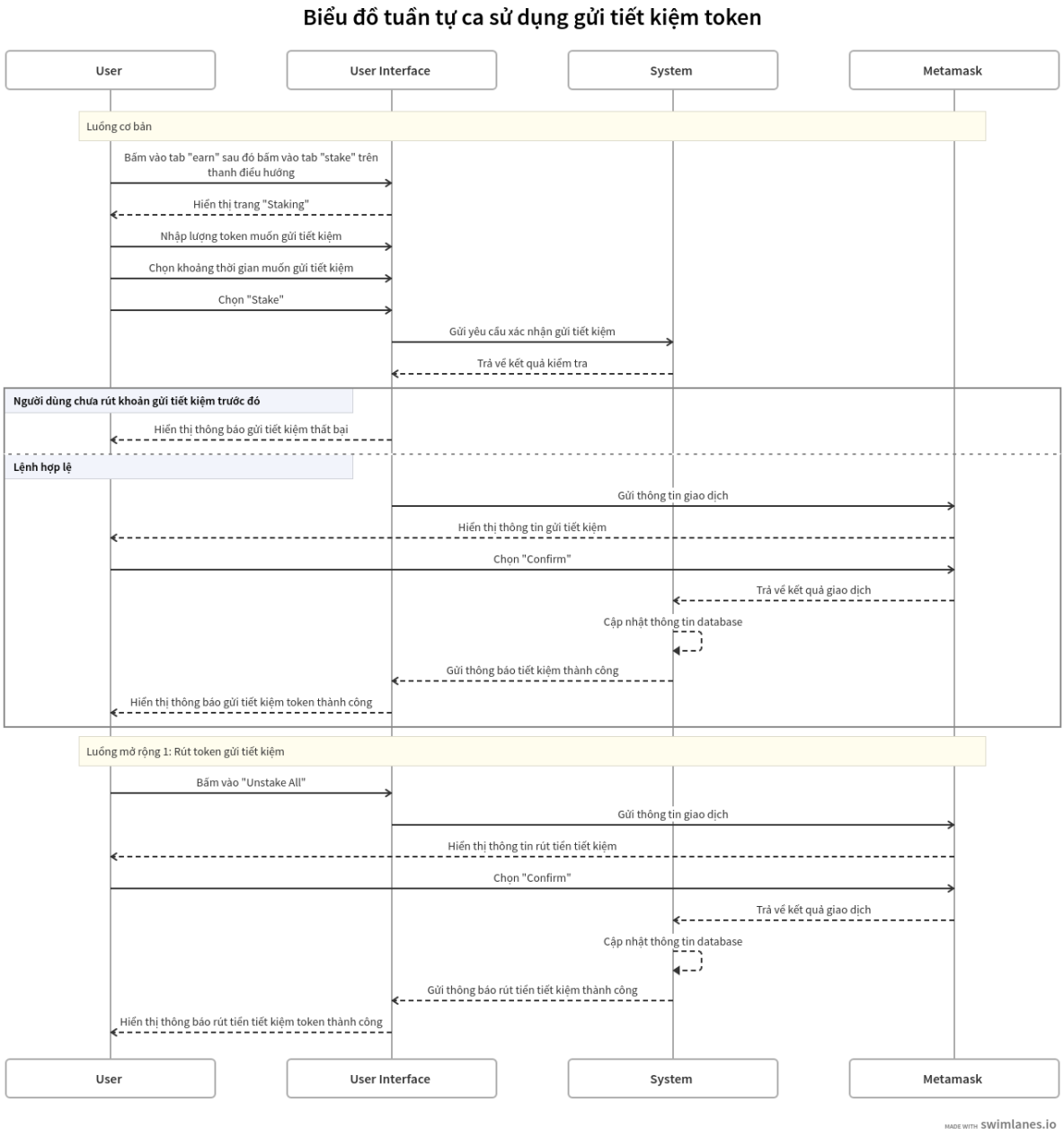
Bảng 2.3: Đặc tả ca sử dụng tự động đặt lệnh giao dịch token



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.

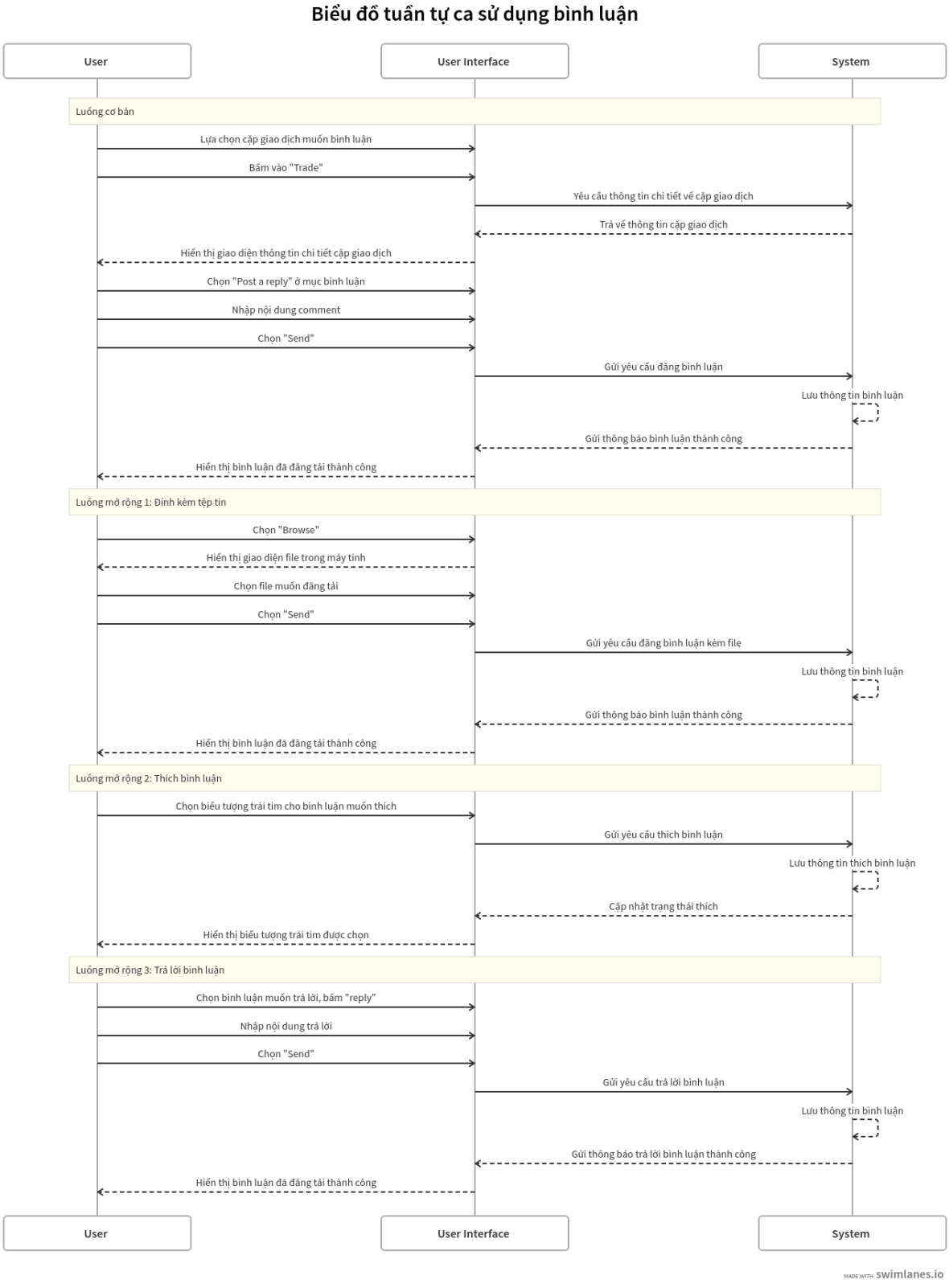
Bảng 2.4: Đặc tả ca sử dụng gửi tiết kiệm token



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"

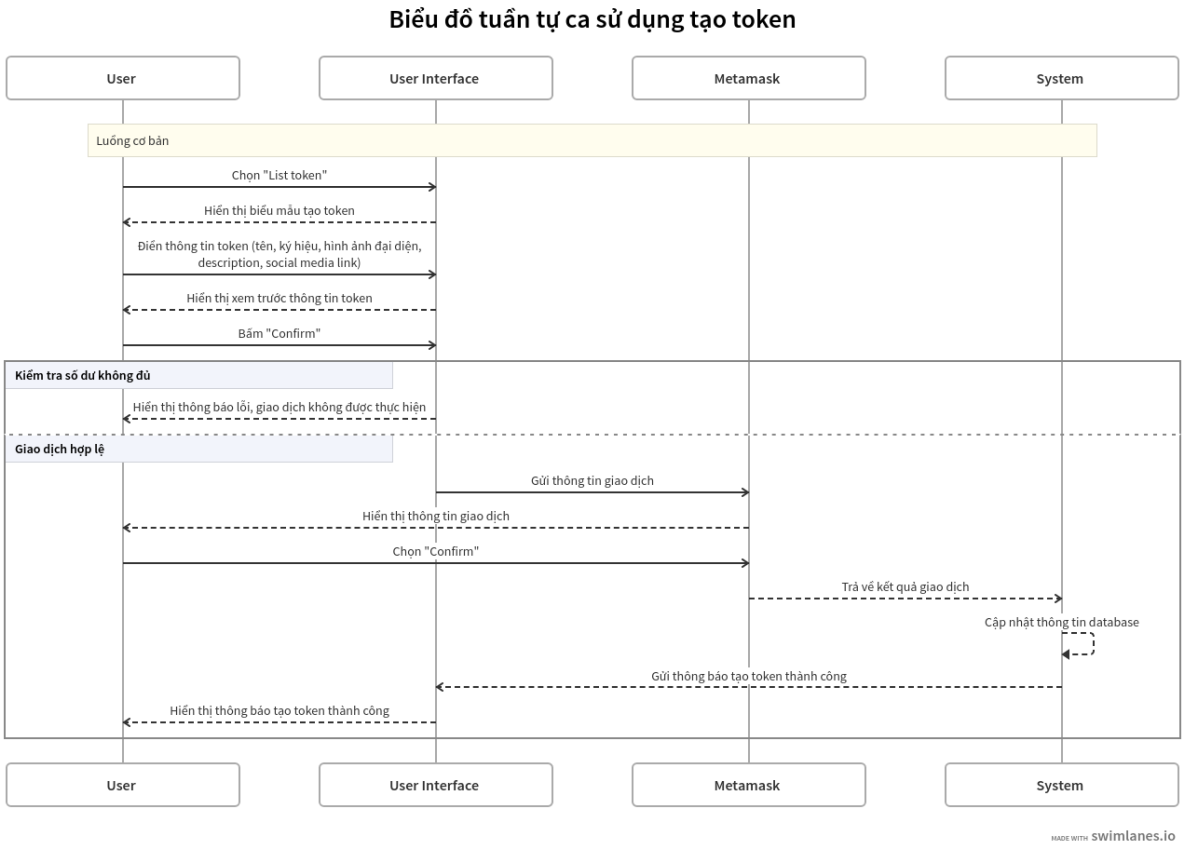
Bảng 2.5: Đặc tả ca sử dụng bình luận



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

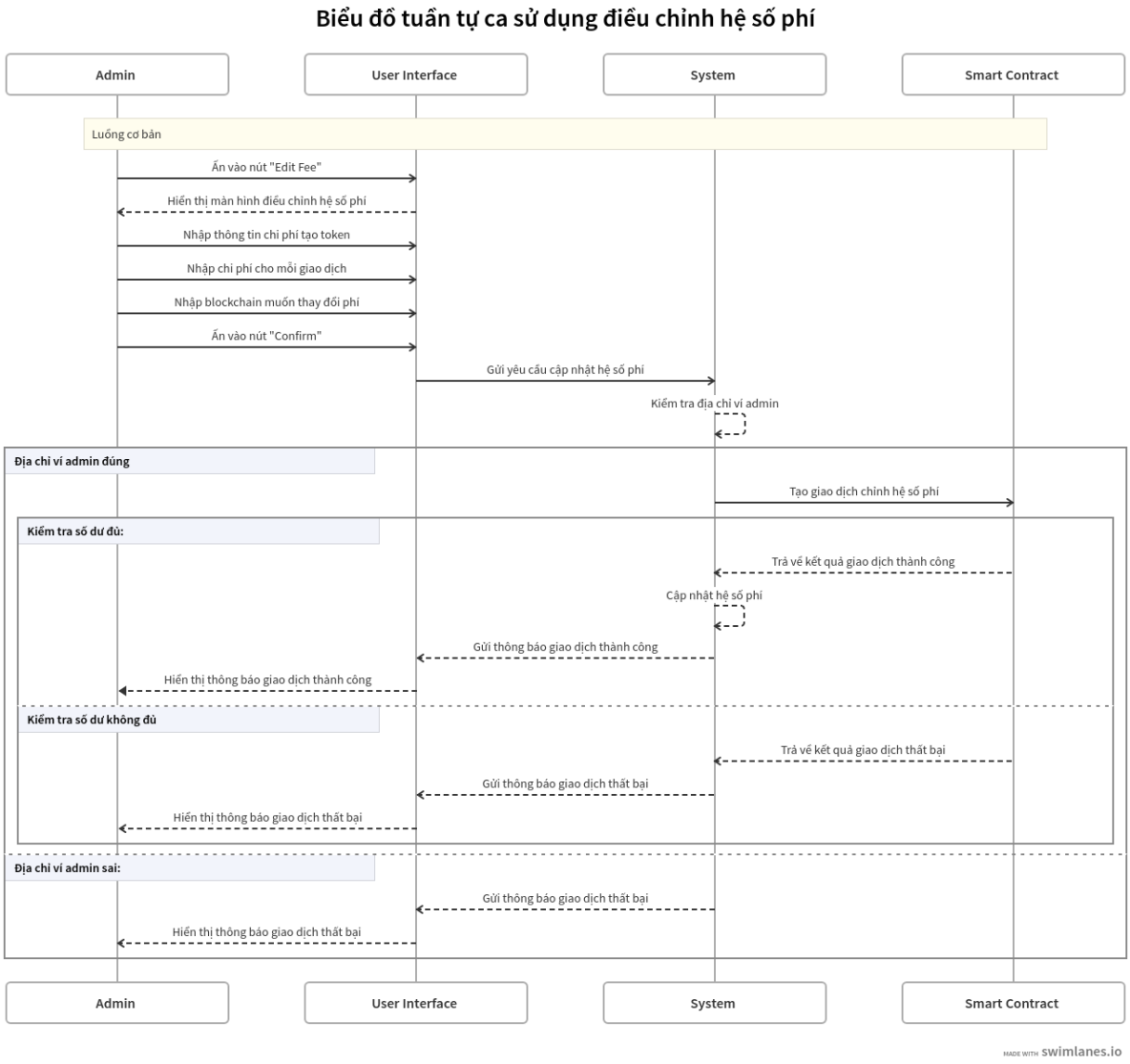
Bảng 2.6: Đặc tả ca sử dụng tạo token



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

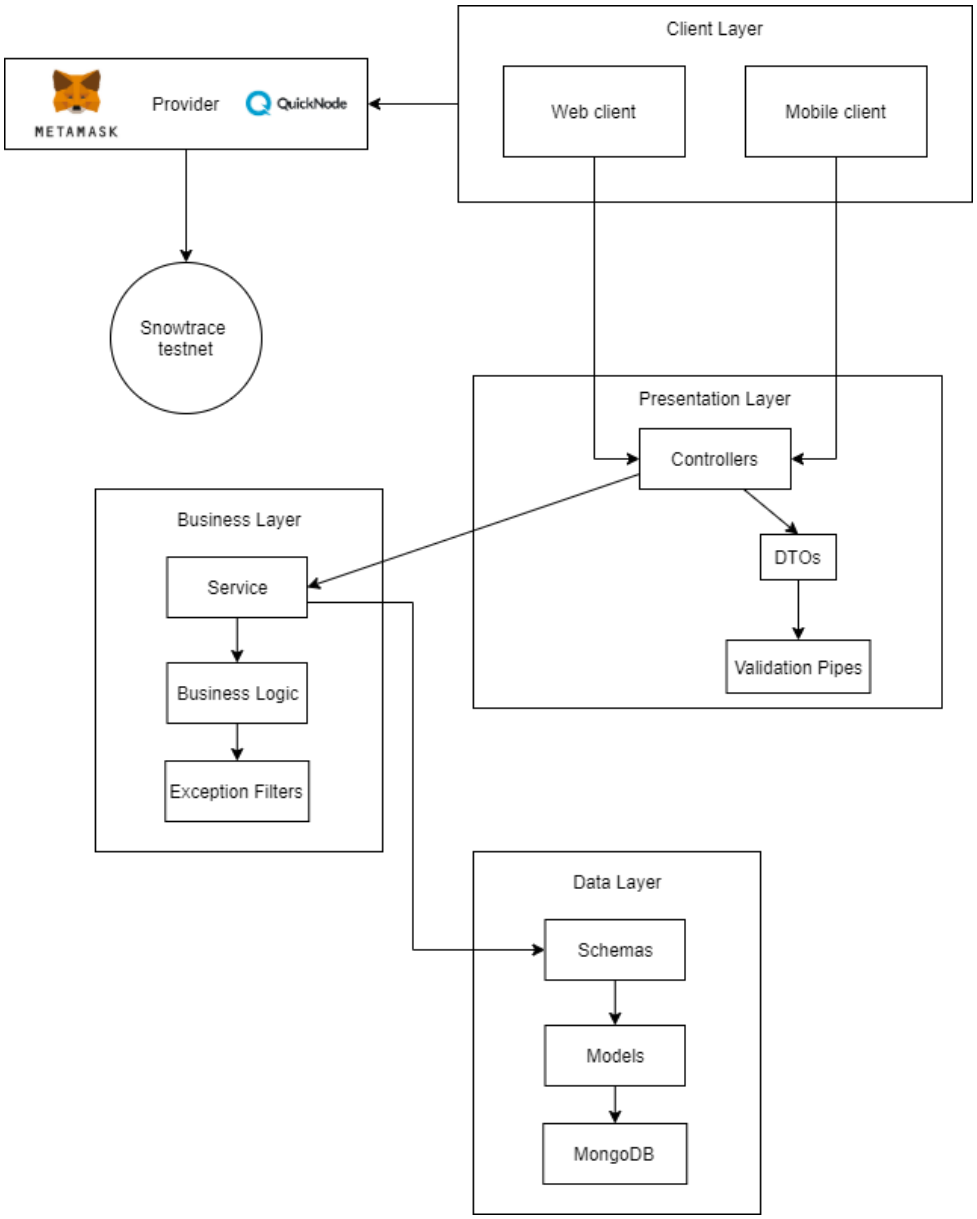
Business Layer (Service Layer):

- 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;
}
```

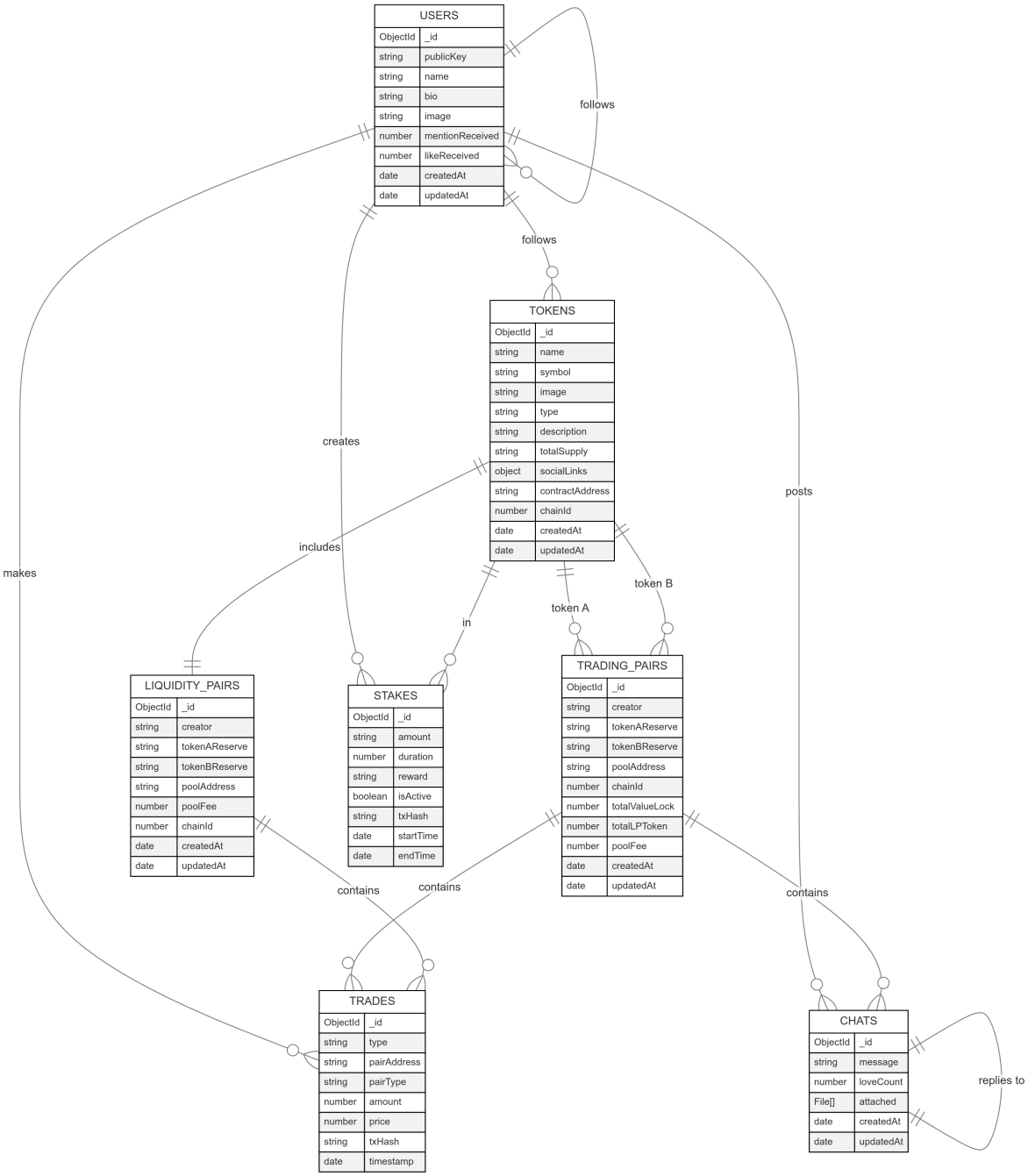
Chức năng chính của Data Layer

- **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

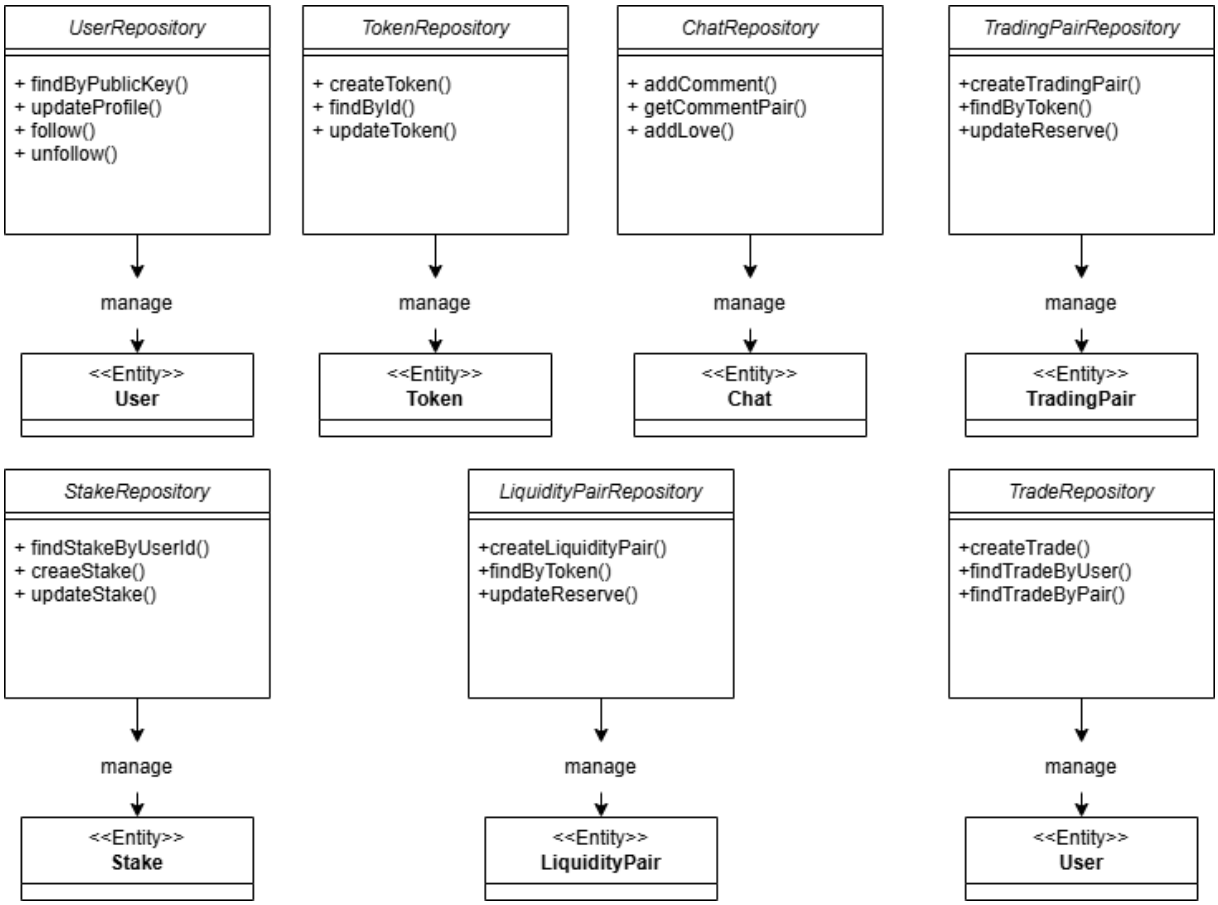
2.4 Thiết kế chi tiết

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	

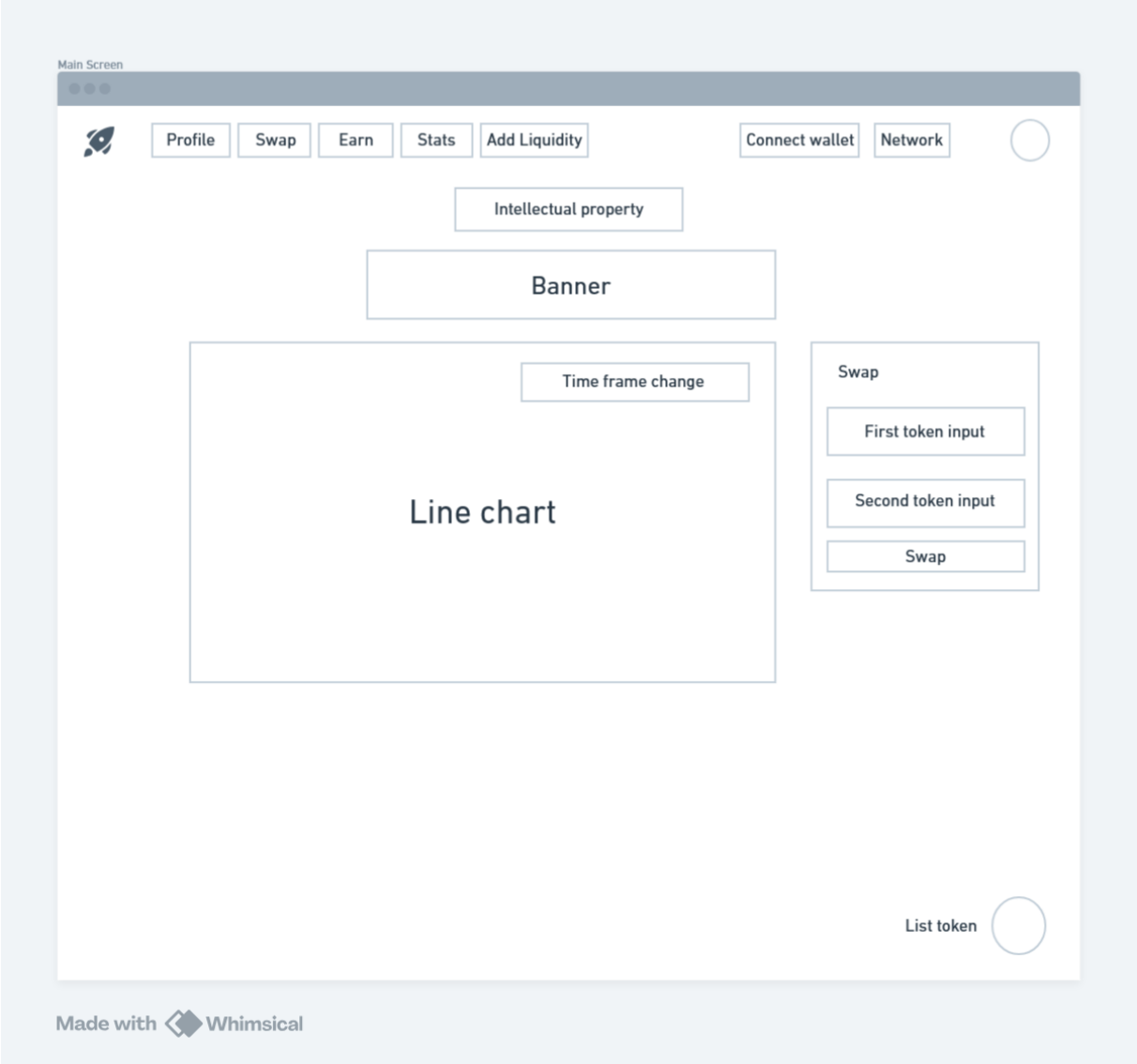
Bảng 2.14: Bảng Stakes

Bảng Trades

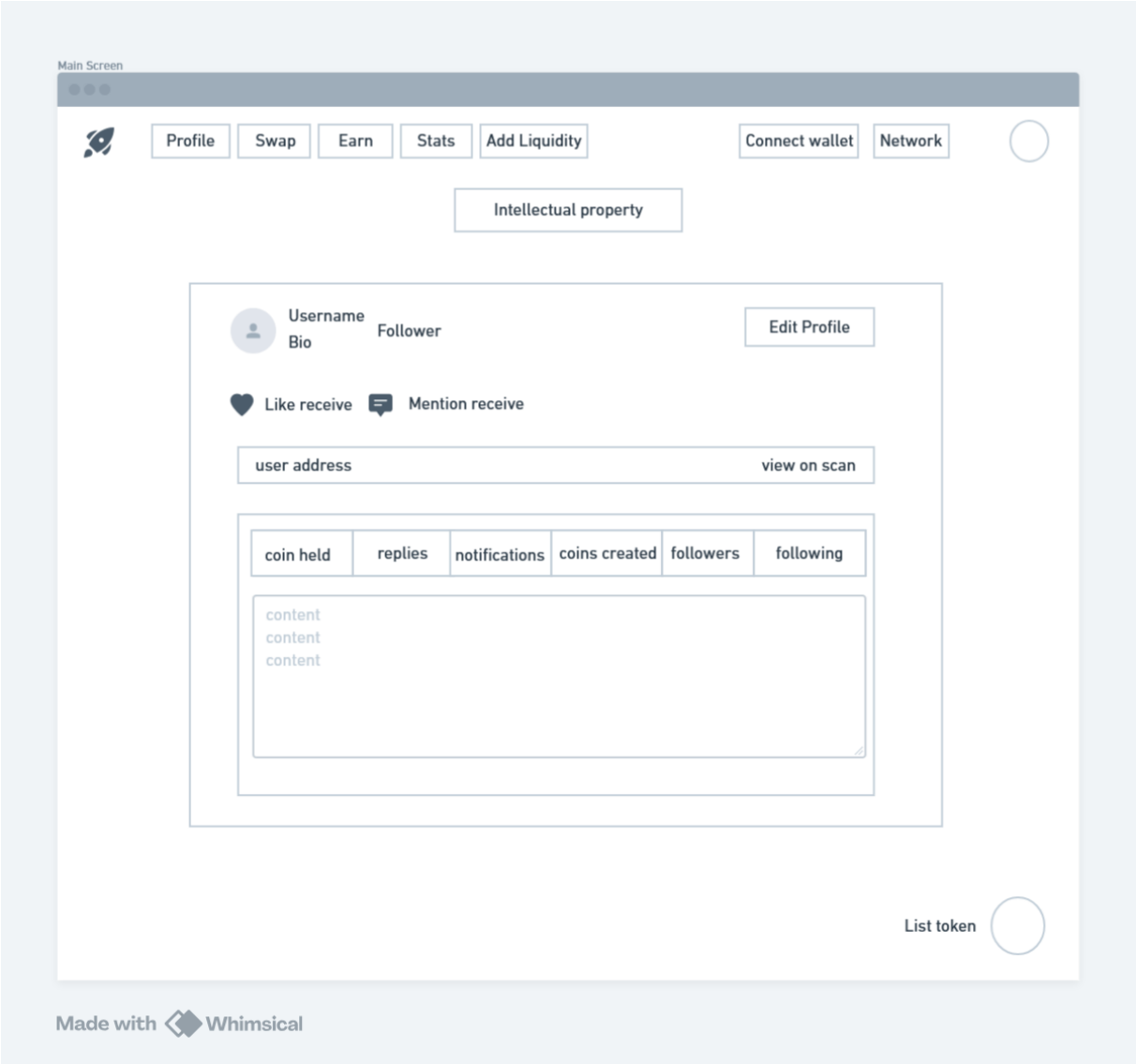
STT	Thuộc tính	Kiểu dữ liệu	Thông tin khác
1	id	ObjectId	
2	type	string	
3	pairId	ObjectId	reference to LiquidityPairs || TradingPairs
4	tokenId	ObjectId	reference to Tokens
5	pairAddress	string	
6	pairType	string	
7	amount	number	
8	price	number	
9	txHash	string	
10	timestamp	date	

Bảng 2.15: Bảng Trades

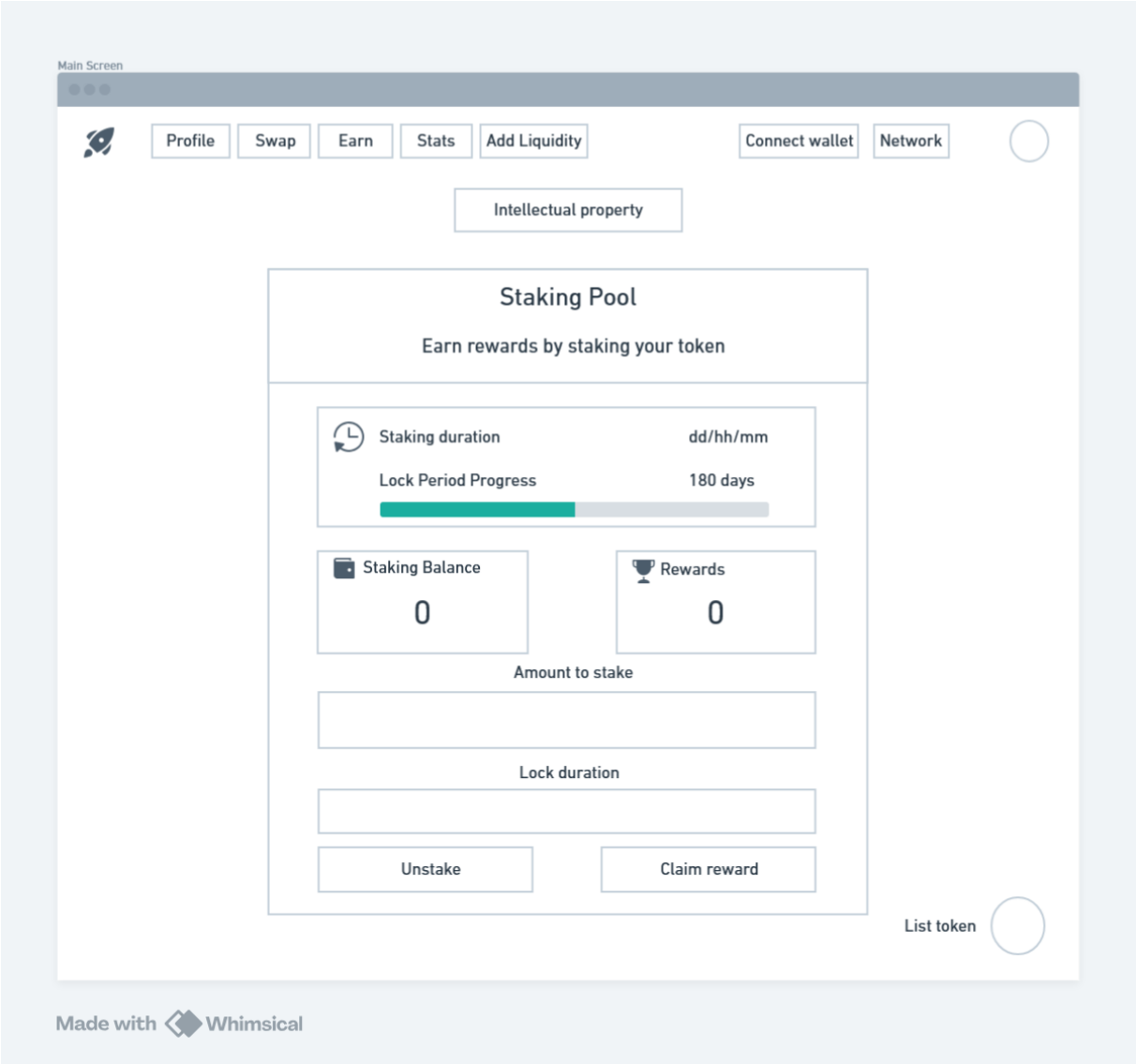
2.4.2 Thiết kế giao diện



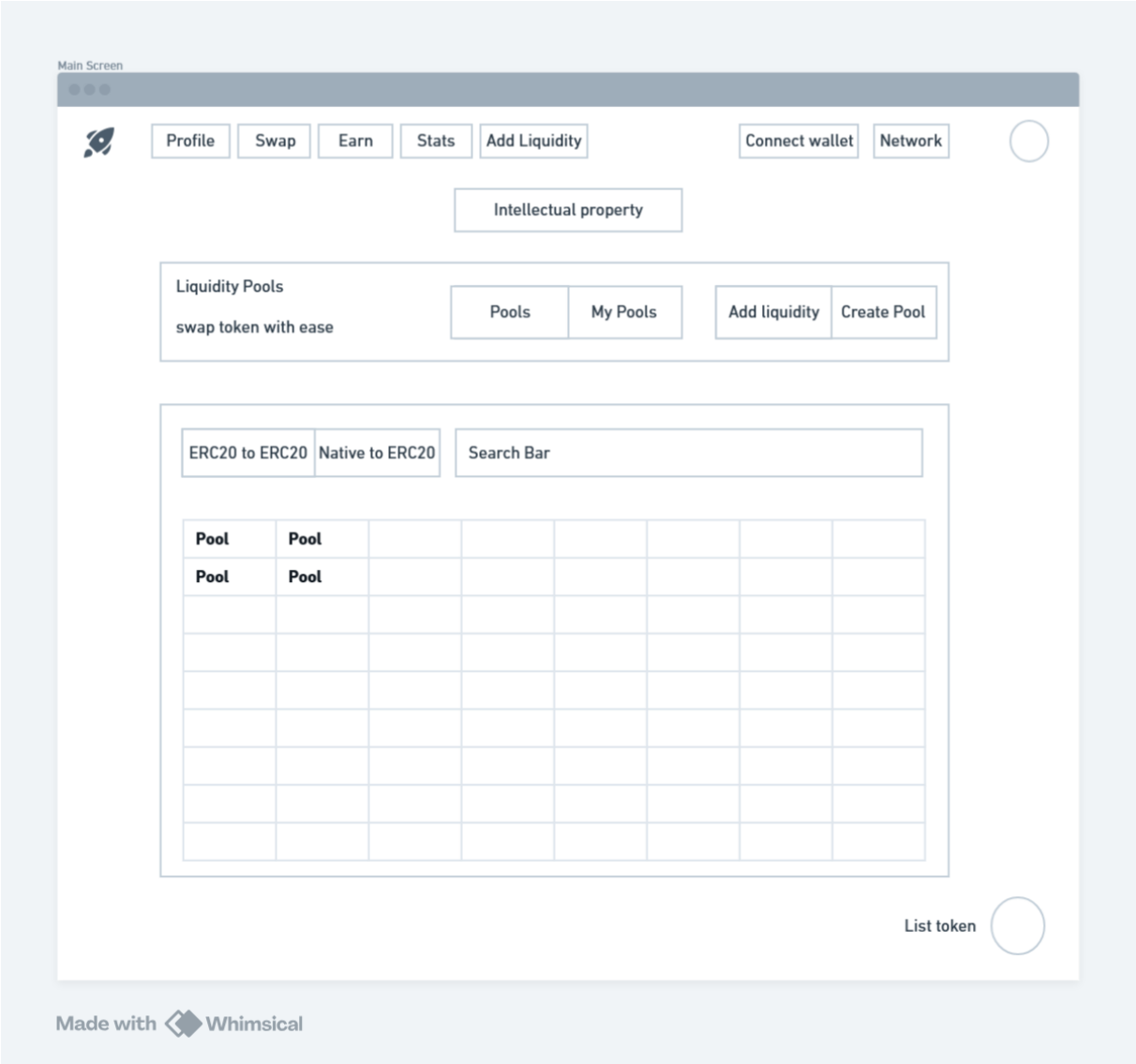
Hình 2.18: Thiết kế trang chủ.



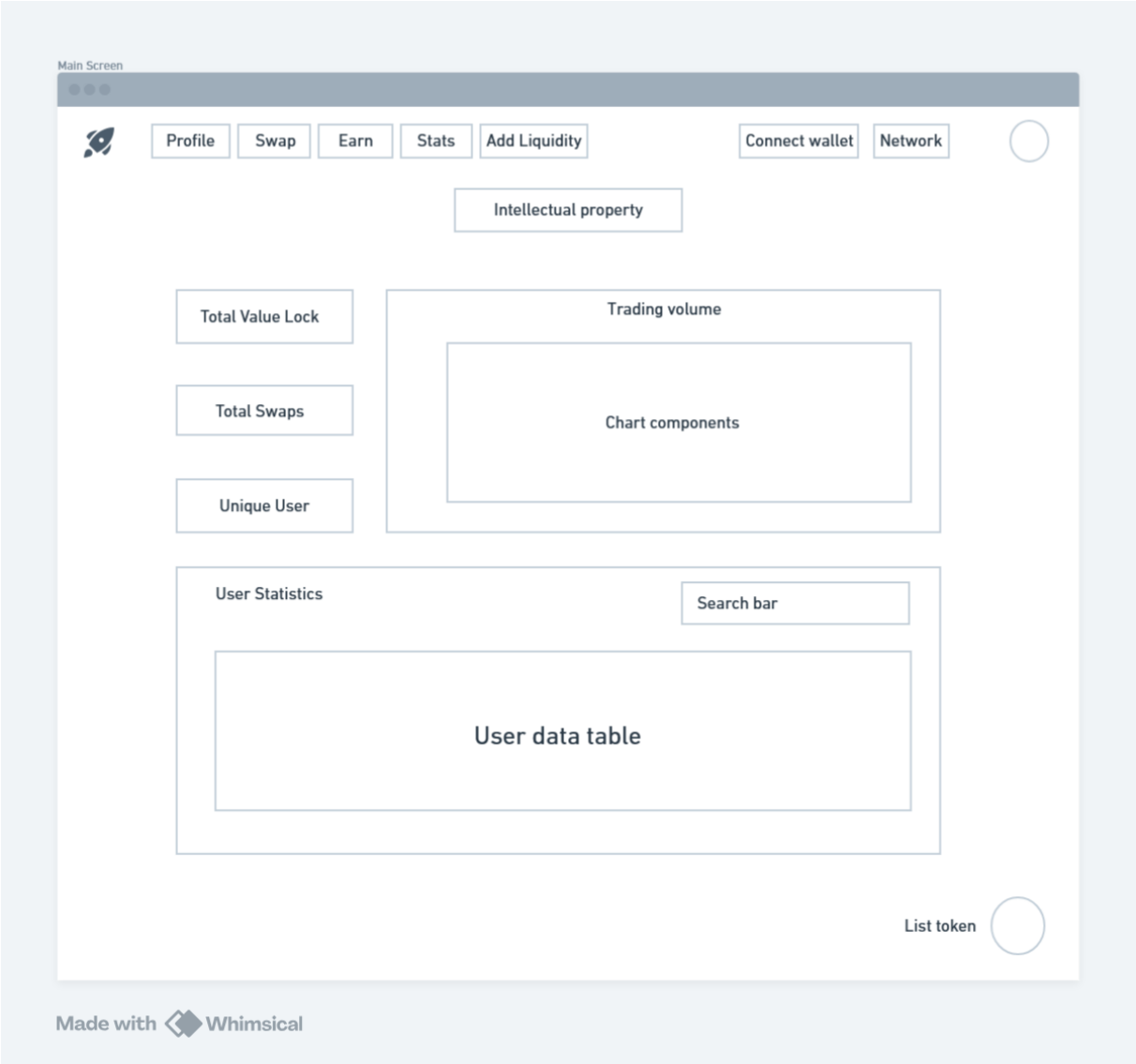
Hình 2.19: Thiết kế trang thông tin người dùng.



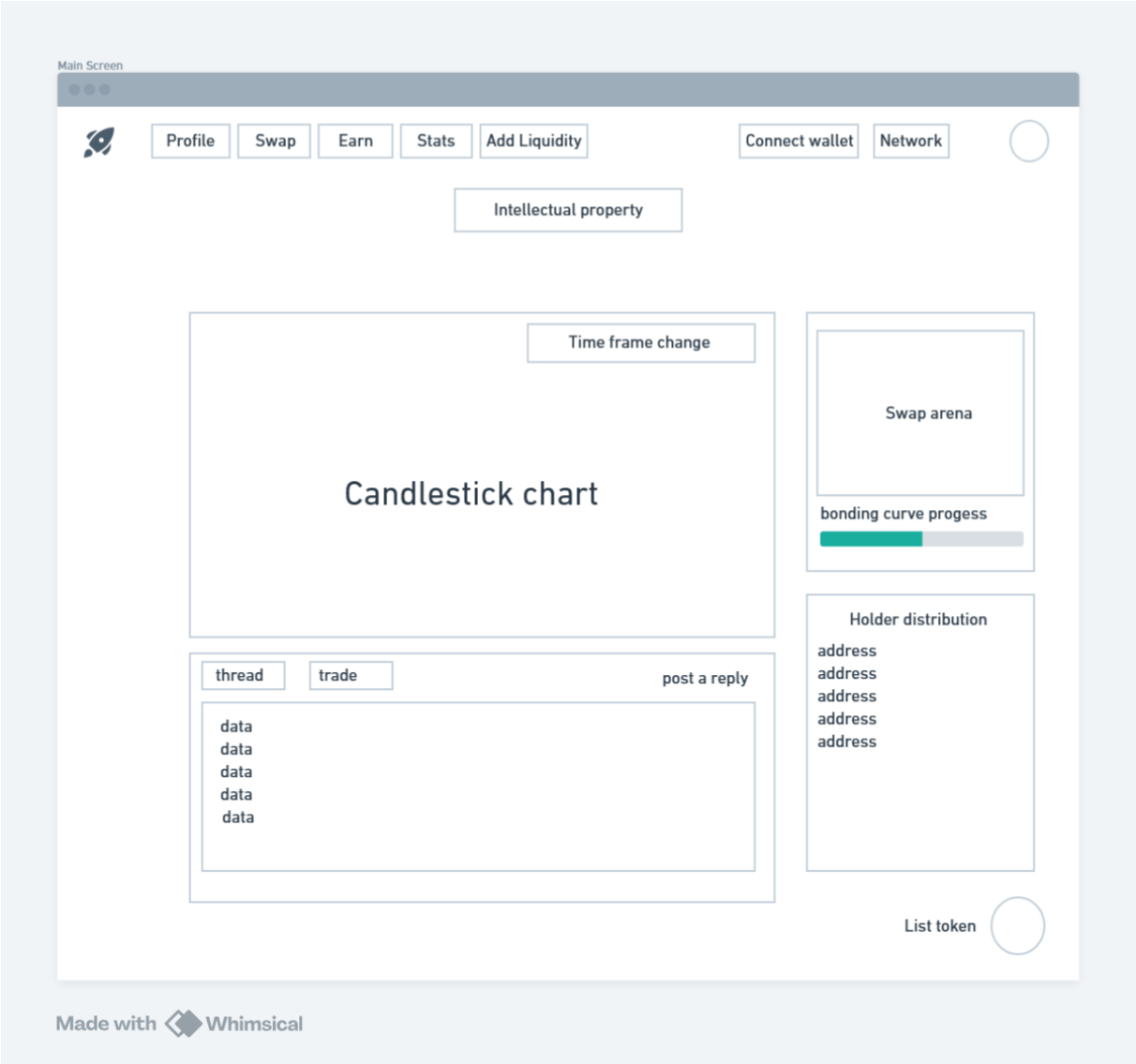
Hình 2.20: Thiết kế trang gửi tiết kiệm token.



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

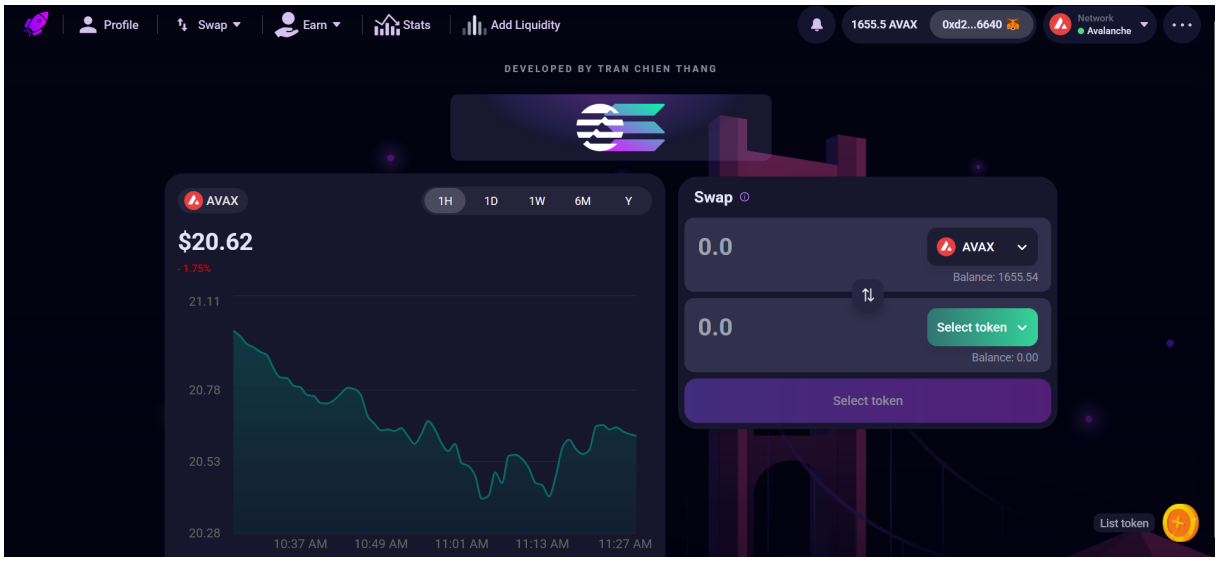
3.1.1 Thư viện và công cụ sử dụng

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

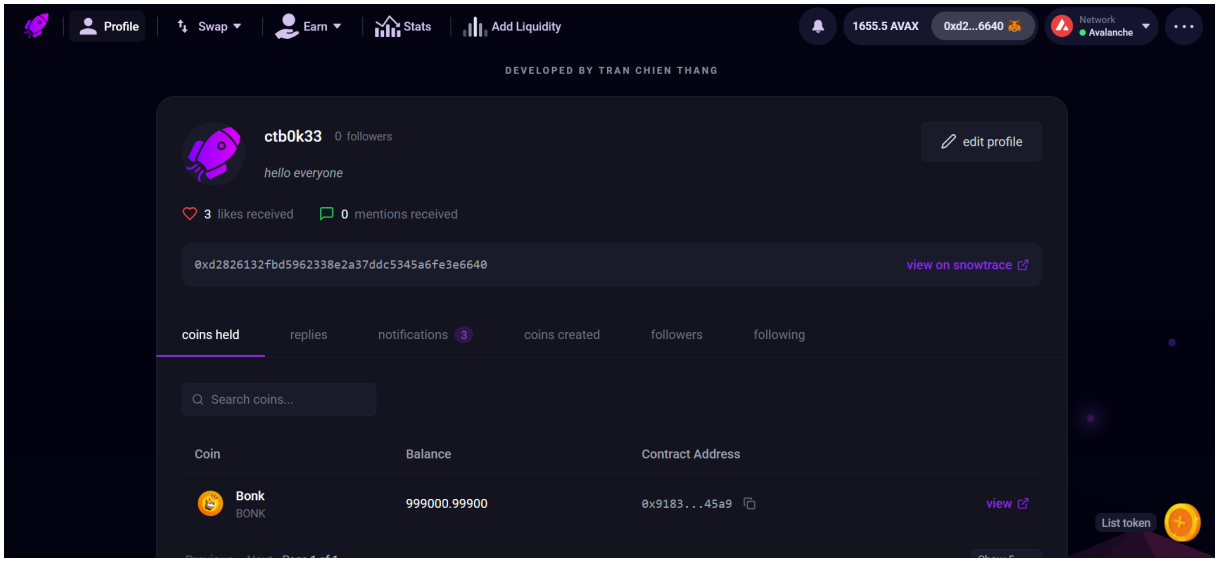
Bảng 3.1: Bảng tổng hợp các công cụ được sử dụng trong đề tài

3.1.2 Kết quả thực nghiệm

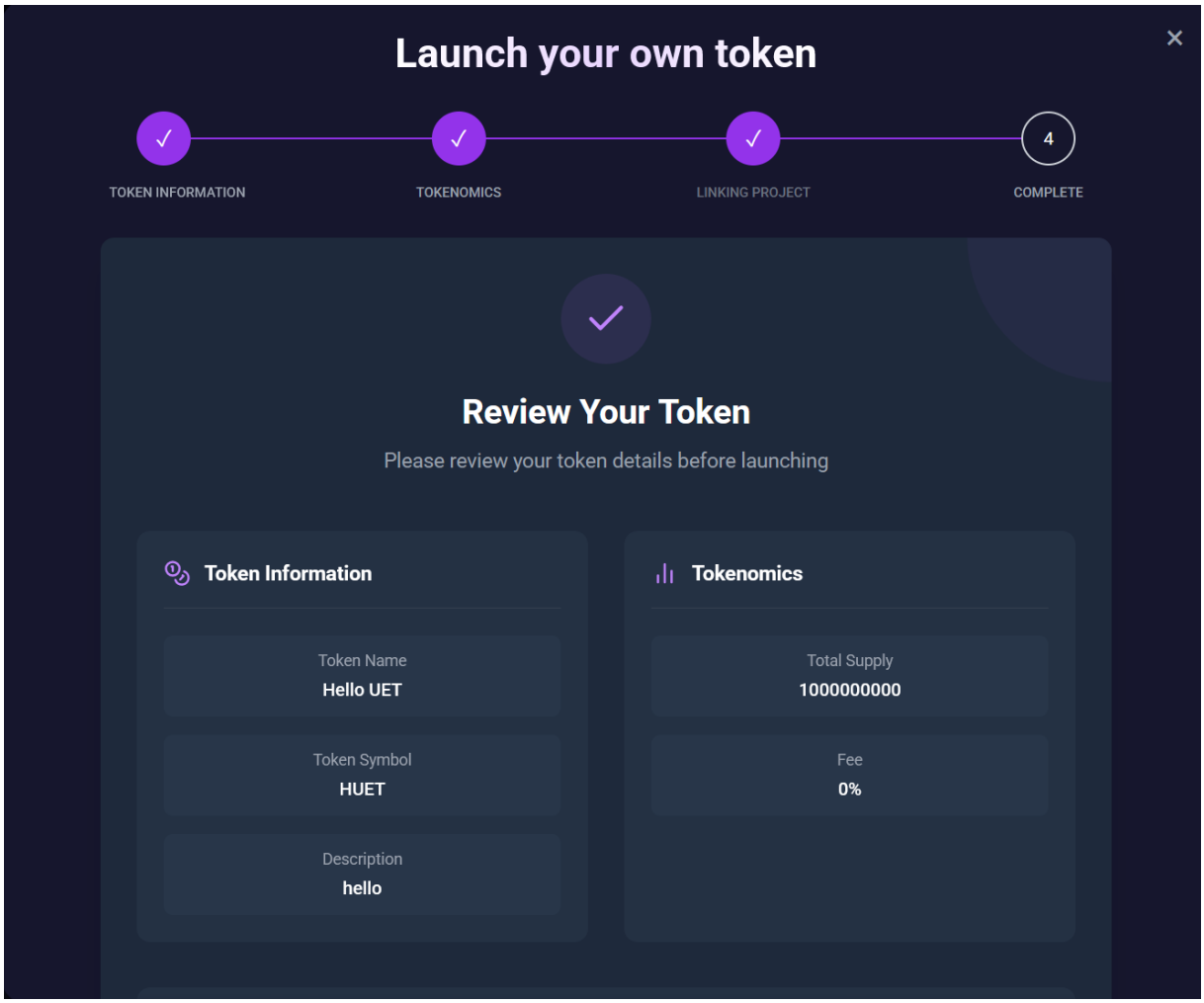
Sau đây là các hình ảnh của thực tế của ứng dụng web đã được phát triển được chạy trên trình duyệt Firefox (Windows 11). Với các thiết bị khác, giao diện sẽ co giãn phù hợp với kích thước màn hình nhưng lượng thông tin không thay đổi nhiều



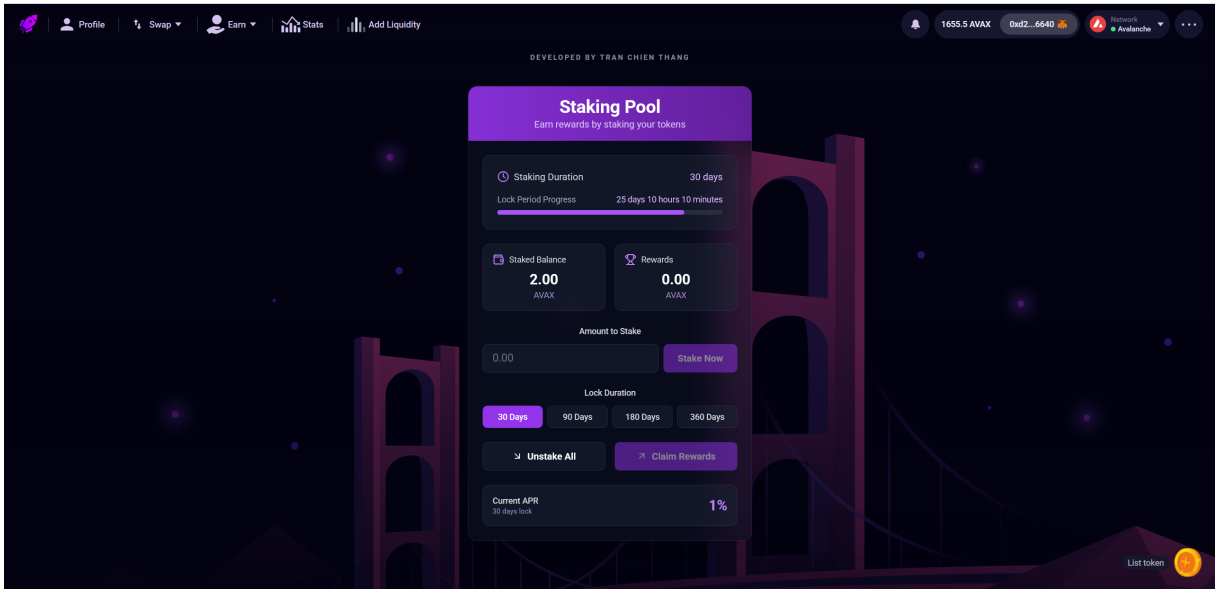
Hình 3.1: Giao diện màn hình chính của ứng dụng.



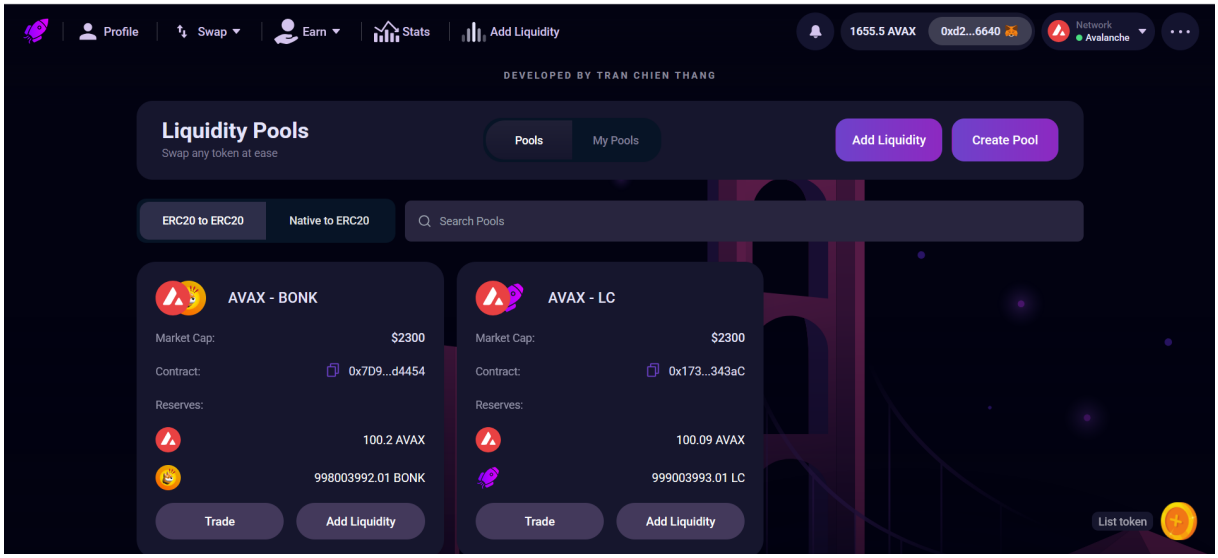
Hình 3.2: Giao diện chức năng quản lý thông tin cá nhân.



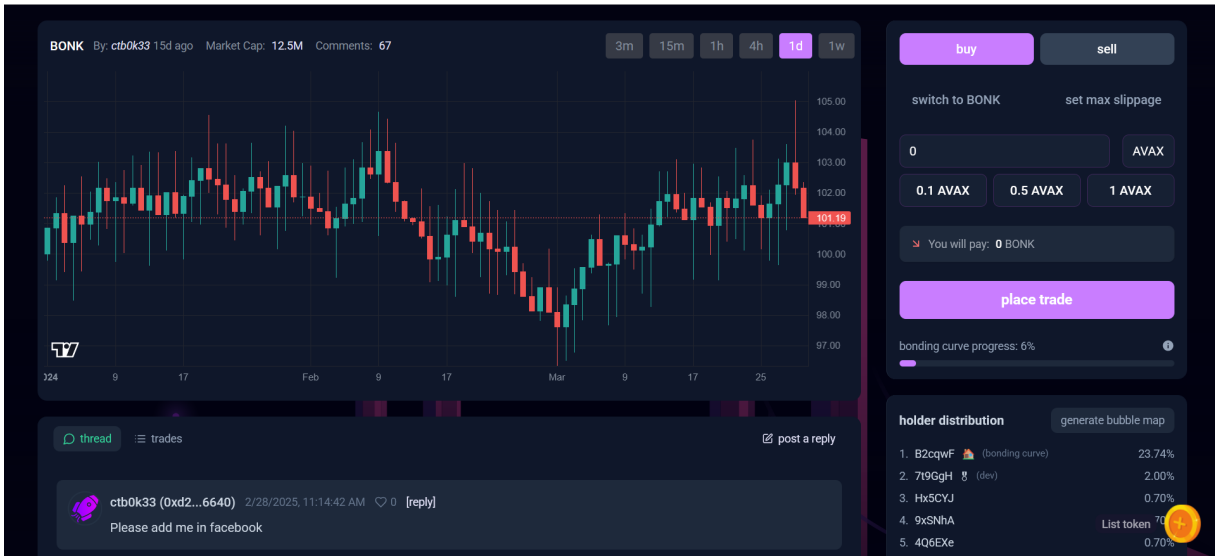
Hình 3.3: Giao diện chức năng tạo token mới.



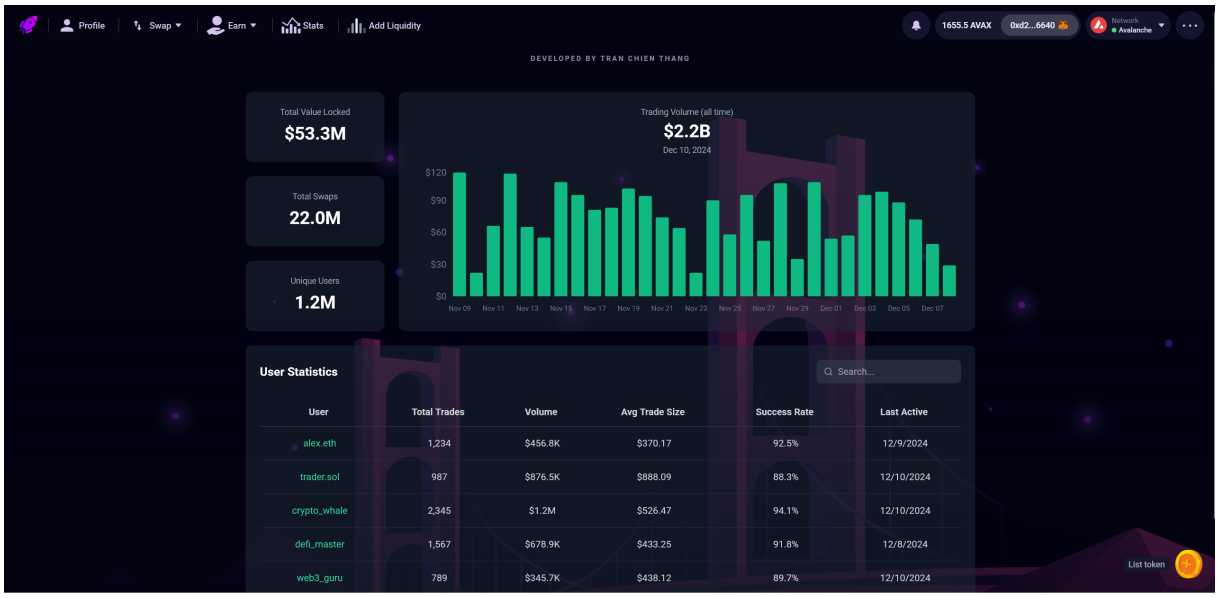
Hình 3.4: Giao diện chức năng gửi tiết kiệm token.



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.