**ĐẠI HỌC HUẾ**

**TRƯỜNG ĐẠI HỌC KINH TẾ**

**---🙡🕮🙣---**

****

**Công nghệ chuổi khối và ứng dụng**

**SINH VIÊN THỰC HIỆN: TRẦN THIỆN NHÂN**

**LỚP: K56 CNTC**

**MSV: 22K4070087**

**GIẢNG VIÊN HƯỚNG DẪN: TS. NGUYỄN MINH ĐỨC**

**HUẾ, 2025**

**MỤC LỤC**

[1.Giới thiệu : 1](#_Toc194339697)

[1.1) Lý do chọn lĩnh vực ứng dụng 1](#_Toc194339698)

[1.2) Tầm quan trọng của blockchain và ứng dụng trong lĩnh vực này 1](#_Toc194339699)

[2. Cơ sở lý thuyết 1](#_Toc194339700)

[2.1 Tổng quan về Blockchain 1](#_Toc194339701)

[2.2 Hợp đồng thông minh (Smart Contract) 2](#_Toc194339702)

[2.3. Ứng dụng của Blockchain trong các ngành khác nhau 3](#_Toc194339703)

[3. Đặc điểm kỹ thuật của ứng dụng: 3](#_Toc194339704)

[3.1 Yêu cầu hệ thống và mô hình kiến trúc 3](#_Toc194339705)

[3.2 Các công nghệ và công cụ được sử dụng 4](#_Toc194339706)

[3.3 Mô tả chi tiết về hợp đồng thông minh 4](#_Toc194339707)

[4 .Thiết kế và triển khai: 5](#_Toc194339708)

[4.1 Quy trình thiết kế 5](#_Toc194339709)

[4.2 Cách triển khai hợp đồng thông minh trên testnet 6](#_Toc194339710)

[4.3 Các bước triển khai cụ thể: 6](#_Toc194339711)

[4.3 Các vấn đề kỹ thuật đã gặp phải: 7](#_Toc194339712)

[5. Kết luận và hướng phát triển 7](#_Toc194339713)

[5.1 Tổng kết các kết quả đạt được 7](#_Toc194339714)

[5.2 Những điểm mạnh và điểm yếu của đồ án 7](#_Toc194339715)

[5.3 Đề xuất các hướng phát triển hoặc ứng dụng tiềm năng trong tương lai 8](#_Toc194339716)

[5.4 Tổng kết: 8](#_Toc194339717)

[6. Tài liệu tham khảo: 9](#_Toc194339718)

[7. Phụ lục: 9](#_Toc194339719)

# 1.Giới thiệu :

## 1.1) Lý do chọn lĩnh vực ứng dụng

Blockchain là một công nghệ nền tảng, đã nổi lên như một yếu tố cách mạng trong nhiều lĩnh vực, đặc biệt là trong ngành tài chính. Việc áp dụng blockchain vào các giao dịch tài chính giúp loại bỏ các trung gian, tạo ra một hệ thống giao dịch tự động, an toàn và minh bạch. Một trong những ứng dụng đặc biệt của blockchain chính là hợp đồng thông minh (smart contract), cho phép thực hiện các giao dịch tự động mà không cần sự can thiệp của bên thứ ba. Lĩnh vực tài chính cá nhân, đặc biệt là việc gửi tiền tiết kiệm và vay mượn, là một trong những lĩnh vực có thể tận dụng rất tốt những lợi ích mà blockchain mang lại.

Chọn lĩnh vực ứng dụng này xuất phát từ nhu cầu ngày càng cao về một hệ thống tài chính tự động và không cần sự trung gian của ngân hàng truyền thống. Thực tế, người dùng hiện nay có thể gặp phải các vấn đề như chi phí giao dịch cao, quy trình phức tạp và mất nhiều thời gian khi thực hiện các giao dịch tài chính. Hợp đồng thông minh trên blockchain giúp giải quyết những vấn đề này, tạo ra một giải pháp đơn giản, minh bạch và nhanh chóng.

## 1.2) Tầm quan trọng của blockchain và ứng dụng trong lĩnh vực này

Blockchain không chỉ đơn thuần là một công nghệ, mà nó còn là một cuộc cách mạng trong cách thức chúng ta lưu trữ, xác thực và chia sẻ dữ liệu. Một trong những đặc điểm quan trọng của blockchain là tính phân tán và không thể thay đổi, nghĩa là khi dữ liệu đã được ghi nhận trên blockchain, nó sẽ không thể bị thay đổi hay xóa bỏ. Điều này giúp đảm bảo tính minh bạch và bảo mật tuyệt đối cho tất cả các giao dịch tài chính. Những đặc điểm này chính là yếu tố quyết định giúp blockchain trở thành một công nghệ lý tưởng để phát triển các hợp đồng thông minh trong các giao dịch tài chính, đặc biệt là cho vay tiền và gửi tiền tiết kiệm.

Ứng dụng hợp đồng thông minh trong việc gửi tiền tiết kiệm và vay mượn trực tiếp giữa các cá nhân mà không cần đến ngân hàng hay tổ chức tài chính trung gian mang lại rất nhiều lợi ích. Các giao dịch trở nên nhanh chóng và ít tốn chi phí hơn do không có sự can thiệp của bên thứ ba. Hơn nữa, tính tự động hóa trong hợp đồng thông minh giúp người dùng có thể thực hiện các giao dịch tài chính mọi lúc, mọi nơi, mà không phải phụ thuộc vào giờ làm việc của ngân hàng hay sự xác nhận thủ công từ các tổ chức tài chính.

Việc sử dụng blockchain và hợp đồng thông minh trong lĩnh vực này không chỉ giúp giảm thiểu chi phí giao dịch mà còn đảm bảo tính minh bạch, bảo mật và giảm thiểu rủi ro cho cả người cho vay và người vay. Điều này đặc biệt quan trọng trong bối cảnh nền kinh tế hiện nay, khi mà người dùng ngày càng tìm kiếm các giải pháp tài chính tiện lợi, an toàn và hiệu quả hơn.

# 2. Cơ sở lý thuyết

## 2.1 Tổng quan về Blockchain

**Khái niệm cơ bản**  
Blockchain là một công nghệ lưu trữ dữ liệu phân tán và không thể thay đổi. Dữ liệu trong blockchain được tổ chức thành các khối (block), và mỗi khối chứa một tập hợp các giao dịch hoặc thông tin. Các khối này được liên kết với nhau bằng mã băm (hash), tạo thành một chuỗi (chain) liên tục. Mỗi khối mới thêm vào đều tham chiếu đến khối trước đó, giúp đảm bảo tính toàn vẹn và an toàn cho dữ liệu. (Nofer)

Blockchain hoạt động dựa trên nguyên tắc phi tập trung, nghĩa là dữ liệu không được lưu trữ ở một vị trí trung tâm mà được sao chép và phân phối trên nhiều nút mạng (node). Các nút này cùng tham gia xác minh và đồng thuận để thêm khối mới vào chuỗi. Điều này giúp blockchain trở nên khó bị tấn công, vì hacker cần kiểm soát được hơn 50% số nút để thay đổi dữ liệu.

**Cơ chế hoạt động**

1. **Ghi nhận giao dịch:** Một giao dịch được khởi tạo bởi người dùng sẽ được truyền đến mạng blockchain.
2. **Xác thực giao dịch:** Các nút trong mạng sử dụng các thuật toán đồng thuận (như Proof of Work, Proof of Stake) để xác thực tính hợp lệ của giao dịch.
3. **Ghi nhận vào khối:** Sau khi được xác thực, giao dịch được thêm vào một khối mới.
4. **Liên kết khối:** Khối mới được gắn vào chuỗi blockchain thông qua mã băm.
5. **Phân phối:** Chuỗi blockchain được cập nhật trên tất cả các nút trong mạng.

**Các loại blockchain**

* **Public Blockchain:** Là mạng lưới mở, nơi bất kỳ ai cũng có thể tham gia và thực hiện giao dịch (ví dụ: Bitcoin, Ethereum).
* **Private Blockchain:** Là mạng lưới riêng, chỉ những người được cấp quyền mới có thể tham gia (thường được sử dụng trong doanh nghiệp).
* **Consortium Blockchain:** Là sự kết hợp giữa public và private blockchain, trong đó quyền truy cập được giới hạn cho một nhóm tổ chức cụ thể.

## 2.2 Hợp đồng thông minh (Smart Contract)

**Định nghĩa**  
Hợp đồng thông minh là một chương trình được triển khai trên blockchain, tự động thực thi các điều khoản của hợp đồng khi các điều kiện được đáp ứng. Ý tưởng này được giới thiệu bởi Nick Szabo vào những năm 1990 và được hiện thực hóa nhờ sự ra đời của Ethereum.

**Cách thức hoạt động**  
Hợp đồng thông minh được viết bằng các ngôn ngữ lập trình như Solidity và triển khai trên blockchain. Khi các điều kiện trong hợp đồng (ví dụ: "A chuyển tiền cho B nếu X xảy ra") được đáp ứng, chương trình sẽ tự động thực thi các hành động đã được quy định, mà không cần sự can thiệp của bên thứ ba. (nguyennsh)

**Lợi ích**

1. **Tự động hóa:** Giảm thiểu sự can thiệp của con người, đảm bảo tính chính xác.
2. **Tiết kiệm chi phí:** Loại bỏ chi phí trung gian.
3. **Minh bạch:** Tất cả các giao dịch và điều khoản đều được lưu trữ trên blockchain và có thể kiểm tra.
4. **Bảo mật:** Dữ liệu trong hợp đồng thông minh được mã hóa và bảo vệ bằng công nghệ blockchain.

**Hạn chế**

1. **Không thể sửa đổi:** Một khi được triển khai, hợp đồng không thể thay đổi.
2. **Phụ thuộc vào lập trình:** Lỗi trong mã nguồn có thể dẫn đến rủi ro tài chính.
3. **Khả năng mở rộng hạn chế:** Các blockchain hiện tại thường gặp khó khăn trong việc xử lý lượng lớn giao dịch.

## 2.3. Ứng dụng của Blockchain trong các ngành khác nhau

1. **Tài chính (DeFi):** Blockchain được ứng dụng để phát triển các dịch vụ tài chính phi tập trung, như trao đổi tiền mã hóa, cho vay, gửi tiết kiệm, và thanh toán ngang hàng mà không cần ngân hàng. (Dash)
2. **Y tế:** Blockchain giúp quản lý hồ sơ bệnh án, đảm bảo dữ liệu y tế chính xác, minh bạch và được bảo mật.
3. **Chuỗi cung ứng:** Công nghệ này giúp theo dõi và xác minh nguồn gốc hàng hóa, giảm thiểu gian lận và tăng tính minh bạch.
4. **Bất động sản:** Blockchain hỗ trợ các giao dịch bất động sản minh bạch, tự động hóa quy trình mua bán và quản lý hợp đồng.
5. **Giải trí:** Các nghệ sĩ có thể bảo vệ quyền sở hữu trí tuệ, phân phối nội dung kỹ thuật số và quản lý doanh thu thông qua blockchain.
6. **Bầu cử:** Blockchain giúp xây dựng các hệ thống bầu cử trực tuyến an toàn, minh bạch và khó bị gian lận.
7. **Quản trị công:** Ứng dụng blockchain trong quản lý tài sản công, cấp giấy phép, hoặc quản lý dữ liệu công dân.

Blockchain không chỉ dừng lại ở một công nghệ đột phá mà còn đang dần trở thành nền tảng cho các giải pháp trong nhiều lĩnh vực, góp phần thay đổi cách con người tương tác với dữ liệu và giao dịch trong thế giới hiện đại.

# 3. Đặc điểm kỹ thuật của ứng dụng:

## 3.1 Yêu cầu hệ thống và mô hình kiến trúc

1. **Yêu cầu hệ thống:**
   * **Phần mềm:**
     + Hệ điều hành: Windows, macOS, hoặc Linux.
     + Trình duyệt hỗ trợ: Chrome, Firefox (dùng để truy cập các công cụ như Remix IDE).
     + Node.js: Phiên bản mới nhất để chạy các công cụ phát triển blockchain như Truffle hoặc Hardhat.
     + Trình quản lý gói: npm hoặc yarn để cài đặt các thư viện cần thiết.
2. **Mô hình kiến trúc:**  
   Ứng dụng được thiết kế theo mô hình phân lớp, bao gồm:
   * **Lớp giao diện người dùng (Frontend):**
     + Giao diện web hoặc ứng dụng di động cho người dùng tương tác, bao gồm chức năng gửi tiền, rút tiền, và vay vốn.
     + Công cụ: React, Angular hoặc Vue.js, tích hợp Web3.js để kết nối với blockchain.
   * **Lớp hợp đồng thông minh (Smart Contract Layer):**
     + Xử lý logic chính của ứng dụng, bao gồm quản lý tài khoản tiết kiệm, lãi suất, và khoản vay.
     + Được triển khai trên blockchain Ethereum.
   * **Lớp mạng blockchain (Blockchain Layer):**
     + Lưu trữ toàn bộ dữ liệu và thực thi các giao dịch thông qua các nút mạng.(**sepolia ETH**)

## 3.2 Các công nghệ và công cụ được sử dụng

1. **Ngôn ngữ lập trình:**
   * **Solidity:** Ngôn ngữ chính để viết hợp đồng thông minh.
2. **Công cụ phát triển:**
   * **Remix IDE:** Môi trường lập trình trực tuyến cho Solidity, hỗ trợ biên dịch và triển khai hợp đồng nhanh chóng.
3. **Blockchain và testnet:**
   * **Ethereum:** Nền tảng blockchain chính để triển khai hợp đồng.
   * **Testnet:** Sepolia dùng để kiểm tra ứng dụng hoặc hơp đồng mà không ảnh hưởng đến mạng chính (mainnet).
4. **Thư viện và công cụ hỗ trợ:**
   * **OpenZeppelin:** Các thư viện mã nguồn mở cho hợp đồng thông minh, hỗ trợ quản lý quyền và bảo mật.

## 3.3 Mô tả chi tiết về hợp đồng thông minh

1. **Cấu trúc hợp đồng thông minh:**  
   Hợp đồng thông minh được tổ chức thành các phần chính:
   * **Khai báo:** Các biến lưu trữ thông tin như số dư tiết kiệm, lãi suất, và khoản vay.
   * **Quyền hạn:** Quy định quyền truy cập của người dùng. Mỗi người dùng sẽ sẽ hữu hợp đồng riêng.
   * **Hàm khởi tạo:** Được gọi khi triển khai hợp đồng, dùng để thiết lập các tham số ban đầu như owner( ở đây là user).
   * **Hàm chính:** Xử lý các chức năng cốt lõi như gửi tiền, rút tiền, vay vốn, và thanh toán nợ.
2. **Chức năng chính:**
   * **Gửi tiết kiệm:** Người dùng chuyển tiền vào hợp đồng thông minh, số tiền này sẽ được lưu trữ trên blockchain và tính lãi suất dựa theo thời gian gửi và số lượng đã gửi.
   * **Rút tiết kiệm:** Người dùng rút tiền từ tài khoản tiết kiệm, bao gồm cả gốc và lãi.
   * **Vay vốn:** Người dùng có thể vay tiền dựa trên khoản tiết kiệm hiện có, với một tỷ lệ giá trị thế chấp (collateral ratio) xác định.
   * **Thanh toán nợ:** Người vay phải trả nợ gốc và lãi thông qua hợp đồng .
3. **Các phương thức chính:**
   * Deposit : Gửi tiền vào hợp đồng, lưu thông tin người dùng như số tiền, kỳ hạn và lãi suất.
   * withdraw: Rút tiền từ hợp đồng, kiểm tra khoản gửi trước khi thực hiện giao dịch.
   * calculate\_and\_checkValidity: tính toán và đưa ra những kỳ hạn số tiền tối đa bạn có thể vay dựa trên thông tin Khoản gửi tiết kiệm của bạn.
   * borrow: Cho phép vay tiền dựa trên khoản tiền gửi và kỳ hạn bạn có thể vay và những thông tin trên đã cấp ở Function calculate\_and\_checkValidity.
   * repayLoan :Thanh toán khoản vay, cập nhật lại sổ tiết kiệm hay khoản gửi của bạn
   * CalculateExpired và CalculateStartTime: Đưa ra thông tin thời gian hết hạn cũng như thời gian của khoản gửi và khoản vay.
   * get\_IndexDeposits và get\_IndexLoans : Kiểm tra thông tin của tất cả khoản gửi và khoản vay của bạn.
   * BaseinterestRate và OptineTime: kiểm tra thông tin lãi suất ( ở đây là lãi suất không kỳ hạn áp dụng cho trường hạn bạn rút trước hạn ) và kỳ hạn bạn có thể gửi tiết kiệm.
4. **Các yêu cầu bảo mật:**
   * **Kiểm tra quyền hạn:** Đảm bảo rằng chỉ người dùng sở hữu tài khoản mới có thể thực hiện các giao dịch liên quan.
   * **Bảo vệ khỏi tấn công Reentrancy:** Sử dụng modifier onlyOwner để tránh khai thác lỗ hổng trong hợp đồng.
   * **Xác thực dữ liệu:** Kiểm tra đầu vào để đảm bảo tính hợp lệ và tránh lỗi logic.
   * **Giới hạn rủi ro:** Áp dụng tỷ lệ thế chấp tối thiểu và giới hạn số tiền vay để bảo vệ tài sản trong hệ thống.

Hợp đồng thông minh này là nền tảng cốt lõi của ứng dụng, giúp đảm bảo tính tự động hóa, minh bạch, và công bằng trong việc quản lý tiết kiệm và vay vốn.

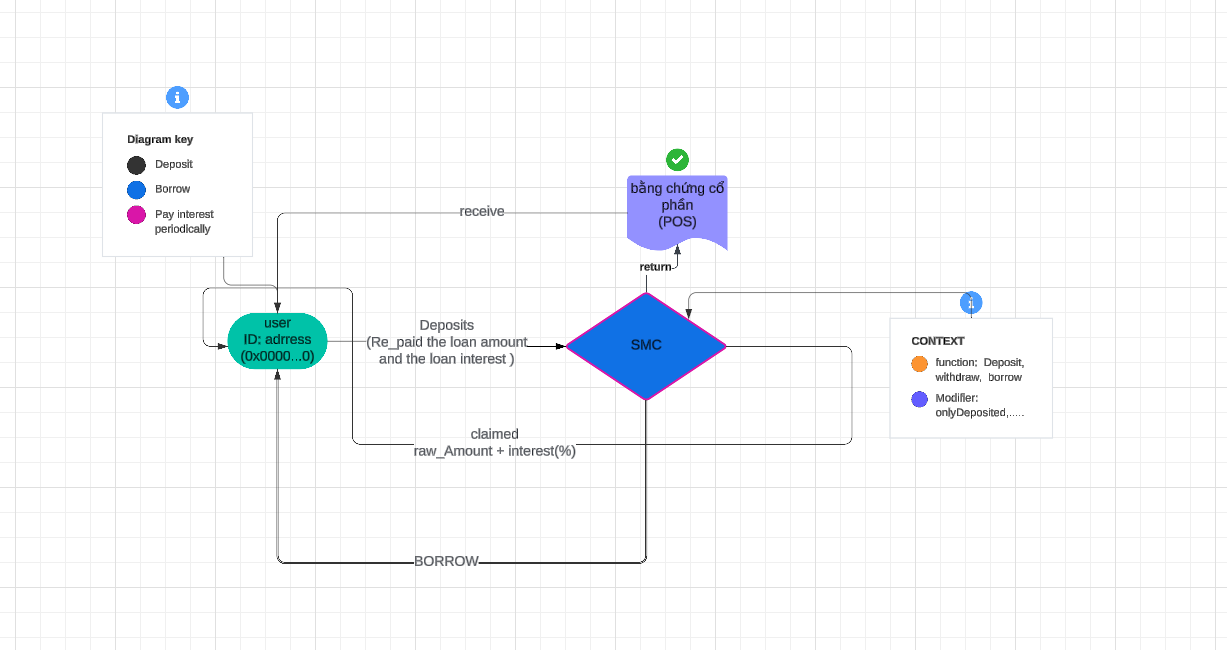
# 4 .Thiết kế và triển khai:

## 4.1 Quy trình thiết kế

1. **Xác định yêu cầu:**
   * Hệ thống cần hỗ trợ hai chức năng chính: gửi tiền tiết kiệm và vay tiền dựa trên số dư tiết kiệm.
   * Yêu cầu các thông số như lãi suất, tỷ lệ thế chấp (collateral ratio), và giới hạn vay tối đa.
2. **Thiết kế sơ đồ hoạt động:**  
   **Sơ đồ hoạt động của ứng dụng:**
   * **Người dùng:**
     1. Gửi tiền tiết kiệm.
     2. Yêu cầu vay tiền.
     3. Rút tiền tiết kiệm.
     4. Thanh toán nợ vay.
   * **Hợp đồng thông minh:**
     1. Quản lý số dư người dùng, khoản vay, và tính lãi suất.
     2. Xử lý các giao dịch một cách tự động.

**Flowchart (luồng hoạt động):**

**A diagram of a diagram

Description automatically generated**

* + **Gửi tiền:**
    1. Người dùng gửi một số tiền đến hợp đồng thông minh.
    2. Hợp đồng lưu trữ số tiền và bắt đầu tính lãi.
  + **Vay tiền:**
    1. Người dùng yêu cầu vay một số tiền.
    2. Hợp đồng kiểm tra số dư tiết kiệm và tỷ lệ thế chấp.
    3. Nếu hợp lệ, khoản vay được phê duyệt và chuyển tiền cho người dùng.
  + **Rút tiền:**
    1. Người dùng yêu cầu rút một số tiền từ tài khoản tiết kiệm.
    2. Hợp đồng kiểm tra số dư còn lại và lãi suất tích lũy.
    3. Nếu hợp lệ, tiền được chuyển lại cho người dùng.
    4. Nếu khoản tiền gửi đang được mang đi thế chấp sẽ bị khóa và phải hoàn thành khoản vay rồi mới có thể rút.

## 4.2 Cách triển khai hợp đồng thông minh trên testnet

1. **Chuẩn bị môi trường:**
   * **Công cụ:** Remix IDE, Truffle, hoặc Hardhat.
   * **Ví tiền mã hóa:** MetaMask để kết nối với testnet Sepolia Ethereum.
2. **Các bước triển khai:**
   * **Viết hợp đồng thông minh:**
     + Sử dụng Solidity để xây dựng hợp đồng với các chức năng như gửi tiền, rút tiền, vay vốn, và thanh toán.
   * **Kiểm tra hợp đồng:**
     + Dùng Remix IDE hoặc Truffle để biên dịch và chạy thử.
     + Kiểm tra các lỗi logic và bảo mật.
   * **Triển khai lên testnet:**
     + Kết nối với MetaMask và chọn testnet.
     + Triển khai hợp đồng thông minh thông qua Remix.
   * **Tích hợp giao diện người dùng: ( Update trong tương lai)**
     + Sử dụng Web3.js hoặc Ethers.js để kết nối ứng dụng frontend với hợp đồng thông minh đã triển khai.

## ****4.3 Các bước triển khai cụ thể:****

1. **Viết hợp đồng:**
2. **Kiểm tra và triển khai:**
   * **Biên dịch:** Sử dụng Remix IDE để kiểm tra lỗi cú pháp và logic.
   * **Triển khai:**
     + Kết nối Remix với MetaMask và chọn testnet.
     + Nhấn nút "Deploy" để triển khai hợp đồng.
3. **Kiểm tra chức năng:**
   * Thực hiện các giao dịch thử nghiệm như gửi tiền, vay tiền, và thanh toán.
   * Theo dõi nhật ký giao dịch trên Etherscan để đảm bảo hoạt động đúng.

## 4.3 Các vấn đề kỹ thuật đã gặp phải:

1. **Lỗi bảo mật:**
   * **Vấn đề:** Đã khắc phục và bảo mật tốt..
   * **Giải pháp:** Sử dụng modifier onlyOwner và HasDeposited để bảo mật và ngăn chặn BUG.
2. **Hạn chế chi phí gas:**
   * **Vấn đề:** Chi phí triển khai và thực thi hợp đồng cao trên testnet.
   * **Giải pháp:** Tối ưu hóa mã nguồn và sử dụng testnet với phí thấp hơn.
3. **Kiểm tra sai lệch dữ liệu:**
   * **Vấn đề:** Lỗi logic khi tính toán số dư hoặc lãi suất.
   * **Giải pháp:** đã update.

Hệ thống sau khi triển khai trên testnet cho kết quả ổn định, đảm bảo và trực quan hơn cho người dùng.

# 5. Kết luận và hướng phát triển

## 5.1 Tổng kết các kết quả đạt được

Trong quá trình thực hiện đồ án, chúng tôi đã đạt được các kết quả quan trọng sau:

1. **Xây dựng hợp đồng thông minh hoàn chỉnh:**
   * Đã thiết kế và triển khai thành công một hợp đồng thông minh cho phép người dùng gửi tiết kiệm, rút tiền, vay vốn, và thanh toán nợ.
   * Hợp đồng hoạt động ổn định trên testnet Ethereum, với các chức năng được kiểm tra đầy đủ.
2. **Ứng dụng các công nghệ blockchain:**
   * Tích hợp các công cụ hiện đại như Remix IDE để phát triển và thử nghiệm ứng dụng.
   * Đảm bảo hợp đồng tuân thủ các nguyên tắc bảo mật như chống tấn công Reentrancy.
3. **Xây dựng kiến trúc hệ thống:**
   * Mô hình kiến trúc rõ ràng, dễ mở rộng, với phân lớp logic cho hợp đồng thông minh và giao diện người dùng.
   * Đảm bảo tính minh bạch, tự động hóa và phi tập trung trong xử lý giao dịch.

## 5.2 Những điểm mạnh và điểm yếu của đồ án

**Điểm mạnh:**

* **Tính thực tiễn cao:**
  + Ứng dụng cung cấp giải pháp tài chính sáng tạo, tận dụng công nghệ blockchain để quản lý tài sản và vay vốn một cách minh bạch và hiệu quả.
* **Tính bảo mật:**
  + Áp dụng các kỹ thuật lập trình an toàn như kiểm tra quyền truy cập, bảo vệ chống tấn công Reentrancy, và xác thực dữ liệu.
* **Khả năng mở rộng:**
  + Kiến trúc hệ thống linh hoạt, dễ dàng tích hợp thêm các tính năng mới.

**Điểm yếu:**

* **Chi phí giao dịch cao:**
  + Phí gas trên mạng Ethereum có thể là rào cản đối với người dùng phổ thông, đặc biệt trong trường hợp mạng bị tắc nghẽn.
* **Giới hạn tính năng:**
  + Hợp đồng thông minh hiện tại chỉ hỗ trợ các chức năng cơ bản. Các tình huống phức tạp như lãi suất linh hoạt áp dụng cho khoản vay hoặc thanh toán nợ theo kỳ hạn chưa được hỗ trợ.

## 5.3 Đề xuất các hướng phát triển hoặc ứng dụng tiềm năng trong tương lai

1. **Phát triển thêm tính năng:**
   * **Lãi suất linh hoạt:** Tùy chỉnh lãi suất dựa trên các điều kiện thị trường.
   * **Hỗ trợ đa dạng tài sản:** Mở rộng hợp đồng để hỗ trợ gửi tiết kiệm và vay vốn bằng các loại tiền mã hóa khác nhau.
   * **Hệ thống cảnh báo và nhắc nhở:** Gửi thông báo cho người dùng khi đến hạn thanh toán nợ.
2. **Tích hợp công nghệ DeFi:**
   * **Staking và Yield Farming** (Allinstation)**:** Kết hợp các cơ chế tạo lợi nhuận khác để tăng tính hấp dẫn cho người gửi tiết kiệm.
   * **Tương tác với các giao thức DeFi khác:** Cho phép người dùng sử dụng tài sản của họ để tham gia vào các giao thức vay/cho vay phi tập trung khác.
3. **Ứng dụng thực tế:**
   * **Tích hợp vào hệ sinh thái ngân hàng:** Xây dựng các giải pháp blockchain cho ngân hàng để quản lý tài khoản tiết kiệm và khoản vay.
   * **Ứng dụng trong cộng đồng phi tập trung (DAO):** Cho phép các tổ chức phi tập trung sử dụng hợp đồng thông minh để quản lý tài chính minh bạch.
4. **Nghiên cứu bảo mật:**
   * Nâng cấp hợp đồng để đối phó với các hình thức tấn công phức tạp hơn trong tương lai.
   * Tích hợp thêm các giao thức bảo mật như ZKP để bảo vệ quyền riêng tư của người dùng. (trangtran.c98)

## 5.4 Tổng kết:

Dự án đã thành công trong việc xây dựng một ứng dụng tài chính phi tập trung cơ bản, minh bạch và an toàn, đáp ứng được nhu cầu gửi tiết kiệm và vay vốn của người dùng. Với các đề xuất phát triển trên, ứng dụng có tiềm năng trở thành một giải pháp tài chính toàn diện hơn, mở rộng phạm vi sử dụng và gia tăng giá trị thực tiễn trong tương lai.

# 6. Tài liệu tham khảo:

Allinstation. (n.d.). Staking và Yield Farming: Kiếm tiền thụ động từ Crypto dễ dàng. *ALL-IN station*.

Dash, B. A. (n.d.). Future Ready Banking With Smart Contracts-CBDC and Impact on the Indian Economy. *nternational Journal of Network Security and Its Applications, 14(5).*

nguyennsh. (n.d.). Smart contract là gì? Tìm hiểu về hợp đồng thông minh trong blockchain. *COIN98 INSIGHTS*.

Nofer, M. G. (n.d.). Blockchain. Business & information systems engineering,.

trangtran.c98, v. (n.d.). Zero-knowledge Proof là gì? Ưu điểm và hạn chế của công nghệ ZKP. *C98*. Retrieved from https://coin98.net/zero-knowledge-proof-la-gi

VietQ. (2022, 12 20). *Doanh nghiệp nhanh chóng nắm bắt xu hướng sản xuất thông minh*. Retrieved from Công nghiệp công nghệ cao: https://congnghiepcongnghecao.com.vn/tin-tuc/t24261/doanh-nghiep-nhanh-chong-nam-bat-xu-huong-san-xuat-thong-minh.html

# 7. Phụ lục:

contract DepositSaving {

    using BokkyPooBahsDateTimeLibrary for uint;

    // Lãi suất cố định

     uint public BaseinterestRate = 1;  // 1% lãi suất không kỳ hạn

     string public OptineTime="co 3 ky han: 3,6,9,12";

     address public owner;

    // cấu trúc khoản gửi

    struct Deposit {

        uint amount;

        uint startTime;

        uint optionTime;

        uint interestRate;

    }

    constructor (){

        owner = msg.sender;

    }

    modifier onlyOwner(){

        require( msg.sender == owner, "only the owner have the permission to operate");\_;

    }

    mapping(address => Deposit[]) private deposits;

    event Deposited(address indexed user, uint amount, uint optionsTime, uint interestRate, uint startTime);

    event Withdrawn( uint Raw\_amount, uint optionsTime, uint interest, uint totalAmount, uint startTime, uint TimeExprised);

    event Reinvested ( uint new\_amount, uint OptineTime, uint interestRate, uint startTime );

    // Hàm gửi tiền

    function deposit(uint amount, uint optionTime) public   onlyOwner {

        require(amount >= 10000000, "so tien gui phai lon hon 10 000 000");

        require(optionTime == 2 || optionTime == 6 || optionTime == 9 || optionTime == 12, "OptionTime no valid");

        uint interestRate;

        if (amount <= 100000000) {

            if (optionTime == 2) interestRate = 10;

            else if (optionTime == 6) interestRate = 15;

            else if (optionTime == 9) interestRate = 20;

            else interestRate = 25;

        } else if (amount > 100000000 && amount <= 500000000) {

            if (optionTime == 2) interestRate = 15;

            else if (optionTime == 6) interestRate = 20;

            else if (optionTime == 9) interestRate = 25;

            else interestRate = 30;

        } else {

            if (optionTime == 2) interestRate = 20;

            else if (optionTime == 6) interestRate = 25;

            else if (optionTime == 9) interestRate = 30;

            else interestRate = 35;

        }

        // Ghi nhận khoản gửi của người dùng

        deposits[msg.sender].push(Deposit({

            amount: amount,

            startTime: block.timestamp,

            optionTime: optionTime,

            interestRate: interestRate

          }));

        emit Deposited(msg.sender, amount, optionTime, interestRate, block.timestamp);

    }

 function withdraw( uint depositIndex) public  onlyOwner noActiveLoans returns (uint, string memory) {

    Deposit memory userDeposit = deposits[msg.sender][depositIndex];

    require(userDeposit.amount > 0, "No deposit found");

    // Tính thời gian

    uint timeDeposited = (block.timestamp - userDeposit.startTime) ;

    uint requireTime = userDeposit.optionTime \* 60;

    uint interestRate;

    // Tính lãi suất

    if (timeDeposited < requireTime) {

        interestRate = BaseinterestRate;

    } else {

        interestRate = userDeposit.interestRate;

    }

    // Tính lãi suất và tổng tiền

    uint totalInterest = (userDeposit.amount \* interestRate / 100 \* timeDeposited) / 3600; // Tính theo ngày

    uint totalAmount = userDeposit.amount + totalInterest;

    // Kiểm tra nếu đã đến hạn nhưng chưa rút => tái hợp đồng

    if (timeDeposited >= (requireTime + requireTime \* 25 / 100)) {

        // Cập nhật khoản gửi mới

        deposits[msg.sender][depositIndex] = Deposit({

            amount: totalAmount,

            startTime: block.timestamp,

            optionTime: userDeposit.optionTime,

            interestRate: userDeposit.interestRate

        });

        // Tạo thông báo

        string memory message = "Your deposit has been reinvested. Check your updated deposit details.";

        // Emit sự kiện tái hợp đồng

        emit Reinvested( totalAmount, userDeposit.optionTime, userDeposit.interestRate, block.timestamp);

        // Trả về số tiền tái hợp đồng và thông báo

        return (totalAmount, message);

    }

    // Nếu người dùng muốn rút đúng hạn

    delete deposits[msg.sender][depositIndex];

    emit Withdrawn(

        userDeposit.amount,

        userDeposit.optionTime,

        totalInterest,

        totalAmount,

        userDeposit.startTime,

        block.timestamp

    );

    // Trả về số tiền đã rút và thông báo

    return (totalAmount, "You have successfully withdrawn your deposit fund.");

}

//////////////////// Hàm tính ngày hết hạn, kèm theo giờ, phút, giây

    event Calculate\_Expired( address  user, uint Deposit\_index, uint year, uint month, uint day, uint hour,uint minute, uint second );

    event CalculateS\_tartTime( address  user, uint Deposit\_index, uint year, uint month, uint day, uint hour,uint minute, uint second );

function CalculateStartTime(uint depositIndex)

    public

    returns (uint year, uint month, uint day, uint hour, uint minute, uint second) {

    require(depositIndex < deposits[msg.sender].length, "Invalid deposit index");

    Deposit memory userDeposit = deposits[msg.sender][depositIndex];

    uint starttime = BokkyPooBahsDateTimeLibrary.convertToVietnamTime(userDeposit.startTime); // Chuyển sang giờ Việt Nam

    (year, month, day, hour, minute, second) = BokkyPooBahsDateTimeLibrary.timestampToDateTime(starttime);

    emit CalculateS\_tartTime(msg.sender, depositIndex, year, month, day, hour, minute, second);

}

function CalculateExpired(uint depositIndex)

    public

    returns (uint year, uint month, uint day, uint hour, uint minute, uint second) {

    require(depositIndex < deposits[msg.sender].length, "Invalid deposit index");

    Deposit memory userDeposit = deposits[msg.sender][depositIndex];

    uint maturity\_Date = userDeposit.startTime + (userDeposit.optionTime \* 60);

    maturity\_Date = BokkyPooBahsDateTimeLibrary.convertToVietnamTime(maturity\_Date); // Chuyển sang giờ Việt Nam

    (year, month, day, hour, minute, second) = BokkyPooBahsDateTimeLibrary.timestampToDateTime(maturity\_Date);

    emit Calculate\_Expired(msg.sender, depositIndex, year, month, day, hour, minute, second);

}

    /// hàm kiểm tra thông tin

function get\_IndexDeposits() public  view returns (

    uint[] memory amounts,

    uint[] memory startTimes,

    uint[] memory optionTimes,

    uint[] memory interestRates

) {

    Deposit[] memory userDeposits = deposits[msg.sender];

    uint depositCount = userDeposits.length;

    amounts = new uint[](depositCount);

    startTimes = new uint[](depositCount);

    optionTimes = new uint[](depositCount);

    interestRates = new uint[](depositCount);

    for (uint i = 0; i < depositCount; i++) {

        Deposit memory dep = userDeposits[i];

        amounts[i] = dep.amount;

        startTimes[i] = dep.startTime;

        optionTimes[i] = dep.optionTime;

        interestRates[i] = dep.interestRate;

    }

}

///////////////////////// LOAN /////////////////////////////////////////////////////////////////////

    struct Loan {

        uint loanAmount;    // Số tiền vay

        uint interestRate;  // Lãi suất vay

        uint startTime;     // Thời gian bắt đầu vay

        uint duration;      // Thời hạn vay (tháng)

        uint remainingRepayment;   // Số tiền còn lại phải trả (gồm cả gốc và lãi)

        uint total\_Repayment;      // Tổng số tiền phải trả ban đầu

    }

    mapping(address => Loan[]) private loans;

    mapping(address => uint[]) private userOptionsDuration;

    event LoanTaken(address indexed user, uint loanAmount, uint interestRate, uint startTime, uint duration, uint debt\_Amount\_mustRePaid);

    event LoanRepaid(address indexed user, uint repaymentAmount, uint loan\_amount\_remaining);

    event ValidityChecked(uint maxLoanAmount, uint remainingMonths,uint[] options\_duration,string  interest\_rate\_borrow);

    modifier noActiveLoans() {

    for (uint i = 0; i < loans[msg.sender].length; i++) {

        if (loans[msg.sender][i].remainingRepayment > 0) {

            revert("You must repay all active loans before withdrawing deposits.");

        }}\_;}

    modifier hasDeposit(uint depositIndex ) {

        require(deposits[msg.sender][depositIndex].amount > 0, "No deposit found");

        \_;

    }

/////////////////////////////

function calculate\_and\_checkValidity( uint depositIndex)

    public

    onlyOwner

    returns (

        uint maxLoanAmount,

        uint remainingMonths,

        uint[] memory options\_duration,

        string  memory interest\_rate\_borrow

    ) {

    Deposit memory userDeposit = deposits[msg.sender][depositIndex];

    uint maturityDate = userDeposit.startTime + (userDeposit.optionTime \* 30 days);

    require(maturityDate > block.timestamp, "Deposit fund has been withdrawn");

    remainingMonths = (maturityDate - block.timestamp) / 30 days;

    options\_duration = new uint[](remainingMonths);

    for (uint i = 0; i < remainingMonths; i++) {

        options\_duration[i] = i + 1;

    }

    userOptionsDuration[msg.sender] = options\_duration; // Lưu lại danh sách này

    maxLoanAmount = userDeposit.amount \* 80 / 100;

    interest\_rate\_borrow = "if you borrow an amount that exceeds 50% of the maxloanAmount, the interest\_borrow is 2 % , otherwise it is 3%";

    emit ValidityChecked( maxLoanAmount, remainingMonths, options\_duration, interest\_rate\_borrow);

    return (maxLoanAmount, remainingMonths, options\_duration, interest\_rate\_borrow);

}

    function isValidDuration(uint[] memory options\_duration, uint duration) internal pure returns (bool) {

        for (uint i = 0; i < options\_duration.length; i++) {

            if (options\_duration[i] == duration) {

                return true;

            }

        }

        return false;

    }

///////////////////////////////

    /////////increase protection

    mapping(address => mapping(uint => bool)) private hasBorrowed;

function borrow(uint depositIndex, uint loanAmount, uint \_duration) public hasDeposit(depositIndex) {

    // Kiểm tra xem depositIndex này đã được vay chưa

    require(!hasBorrowed[msg.sender][depositIndex], "This deposit has already been used for a loan");

    Deposit memory userDeposit = deposits[msg.sender][depositIndex];

    uint interest\_rate\_borrow;

    // Ensure the loan amount doesn't exceed 80% of the deposit

    uint maxLoanAmount = userDeposit.amount \* 80 / 100;

    require(loanAmount <= maxLoanAmount, "Loan amount exceeds 80% of deposit");

    // Tính lãi suất

    if (loanAmount <= (50 \* maxLoanAmount) / 100) {

        interest\_rate\_borrow = (userDeposit.interestRate + 20); // 2% lãi suất

    } else {

        interest\_rate\_borrow = (userDeposit.interestRate + 30); // 3% lãi suất

    }

    uint totalRepayment = loanAmount + (loanAmount \* interest\_rate\_borrow / 100 \* \_duration / 12);

    // Ensure duration is valid

    uint[] memory options\_duration = userOptionsDuration[msg.sender];

    require(isValidDuration(options\_duration, \_duration), "Invalid duration");

    // Thêm khoản vay vào mảng

    loans[msg.sender].push(Loan({

        loanAmount: loanAmount,

        interestRate: interest\_rate\_borrow,

        startTime: block.timestamp,

        duration: \_duration,

        remainingRepayment: totalRepayment,

        total\_Repayment: totalRepayment

    }));

    // Đánh dấu depositIndex đã được sử dụng để vay

    hasBorrowed[msg.sender][depositIndex] = true;

    emit LoanTaken(msg.sender, loanAmount, interest\_rate\_borrow, block.timestamp, \_duration, totalRepayment );}

function repayLoan(uint loanIndex, uint repaymentAmount) public onlyOwner {

    Loan storage userLoan = loans[msg.sender][loanIndex];

    require(repaymentAmount <= userLoan.total\_Repayment, "Incorrect repayment amount. Please pay the correct amount.");

    // Giảm số dư khoản vay

    userLoan.remainingRepayment -= repaymentAmount;

    // Phát sự kiện khi trả nợ thành công

    emit LoanRepaid(msg.sender, repaymentAmount, userLoan.remainingRepayment);

    // Nếu khoản vay đã trả hết

    if (userLoan.remainingRepayment == 0) {

        delete loans[msg.sender][loanIndex];

    }

}

///////////////////////////////////////

function get\_IndexLoans()

    public

    view

    returns (

        uint[] memory loanAmounts,

        uint[] memory interestRates,

        uint[] memory startTimes,

        uint[] memory durations

    )

{

    // Lấy danh sách tất cả các khoản vay của người dùng

    Loan[] memory userLoans = loans[msg.sender];

    uint loanCount = userLoans.length;

    // Khởi tạo các mảng để lưu thông tin khoản vay

    loanAmounts = new uint[](loanCount);

    interestRates = new uint[](loanCount);

    startTimes = new uint[](loanCount);

    durations = new uint[](loanCount);

    // Lặp qua tất cả các khoản vay và lưu thông tin vào các mảng tương ứng

    for (uint i = 0; i < loanCount; i++) {

        Loan memory loan = userLoans[i];

        loanAmounts[i] = loan.loanAmount;

        interestRates[i] = loan.interestRate;

        startTimes[i] = loan.startTime;

        durations[i] = loan.duration;

    }

}

}