Đối với dự án này, bạn sẽ làm việc trên một ứng dụng khách web và hai hợp đồng Solidity để triển khai trao đổi tiền điện tử phi tập trung. Vào cuối dự án, sàn giao dịch của bạn sẽ có nhiều chức năng được sở hữu bởi các sàn giao dịch phi tập trung đầy đủ như Uniswap. Ngoài ra, bạn sẽ tạo mã thông báo ERC20 của riêng mình , sau đó bạn sẽ giao dịch trên sàn giao dịch của mình.

Trong suốt dự án này, hãy đảm bảo viết mã có khả năng phục hồi trước các cuộc tấn công của đối thủ.
Hãy nhớ rằng những kẻ tấn công có thể gọi hợp đồng thông minh của bạn bằng các đầu vào tùy ý, không chỉ thông qua giao diện người dùng được cung cấp. Do đó, hãy sử dụng tự do các câu lệnh yêu cầu để kiểm tra bất kỳ giả định nào được đưa vào mã của bạn.

Đối với hầu hết sinh viên, dự án này có thể sẽ đại diện cho dự án toàn diện nhất của họ được triển khai cho đến nay trong Solidity. Vì vậy, chúng tôi khuyến khích bạn bắt đầu sớm và đặt câu hỏi. Chúng tôi sẽ hướng dẫn bạn thực hiện trao đổi phi tập trung theo nhiều giai đoạn và chúng tôi tin rằng sản phẩm cuối cùng của bạn sẽ mang lại cho bạn điều gì đó rất đáng tự hào! Bắt đầu nào.

Là một phần của điều này, chúng tôi cũng sẽ xây dựng giao diện người dùng tính toán thông tin hữu ích cho người dùng và cho phép những người không phải là lập trình viên sử dụng DApp.

1 Bắt đầu

1.1 Cài đặt

1. Cài đặt phần mềm tiên quyết: bạn cần tải xuống và cài đặt Node.js. Hardhat chỉ hỗ trợ Node.js V12.xx, 14.xx và 16.xx. Bạn có thể tìm thấy các bản phát hành trước tại đây và chọn một phiên bản thích hợp để cài đặt.

2. Tải xuống và giải nén mã bắt đầu từ trang web của khóa học.

3. cd vào thư mục mã khởi động.

4. Chạy `npm install --save-dev` hardhat để cài đặt Hardhat môi trường phát triển Ethereum mà bạn sẽ sử dụng để mô phỏng nút Ethereum trên máy cục bộ của mình. Nếu bạn gặp lỗi liên quan đến phiên bản Node không tương thích, vui lòng cài đặt phiên bản Node.js được hỗ trợ và sử dụng nvm use <version> để chuyển sang phiên bản đó.

5. Chạy `npm install --save-dev @nomiclabs/hardhat-ethers ethers` để cài đặt plugin Hardhat nhằm triển khai nút Hardhat mà tập lệnh của bạn sẽ sử dụng.

6. Chạy `npm install --save-dev @openzeppelin/contracts` để cài đặt các thư viện OpenZeppelin.

1.2 Biên dịch, Triển khai và Kiểm tra

1. Mở thư mục mã khởi động trong IDE hoặc trình soạn thảo văn bản yêu thích của bạn (thứ gì đó như Sublime Text, Atom hoặc Visual Studio Code hoạt động tốt). Bạn sẽ sửa đổi hợp đồng/token.sol và hợp đồng/exchange.sol để xác định hợp đồng Solidity của mình và web_app/exchange.js để xây dựng chương trình phụ trợ Javascript. Nhìn vào các tệp khác có thể giúp hiểu cách thức hoạt động của nút Hardhat và máy khách web. Có những nơi được đánh dấu bằng các chức năng cần sửa đổi và bạn có thể thêm các chức năng trợ giúp vào ba tệp được liệt kê ở trên. Vui lòng không sửa đổi bất kỳ mã nào khác hoặc cài đặt các gói nút bổ sung.

2. Xem kỹ mã khởi động, tài liệu ethers.js, tài liệu Solidity và triển khai hợp đồng OpenZeppelin ERC-20. Hãy suy nghĩ cẩn thận về thiết kế tổng thể của hệ thống trước khi bạn viết mã. Dữ liệu nào nên được lưu trữ trên chuỗi? Tính toán nào sẽ được thực hiện bởi hợp đồng so với khách hàng?

npx hardhat node

3. Sau khi bạn hoàn thành việc triển khai, hãy chạy nút npx hardhat để bắt đầu nút cục bộ. Nếu nút được khởi động chính xác, bạn sẽ thấy trong thiết bị đầu cuối: Máy chủ HTTP và WebSocket JSON-RPC đã khởi động tại https://localhost:8545

4. Triển khai Hợp đồng: Mở một tab hoặc cửa sổ đầu cuối khác và cd vào bộ khởi động thư mục mã.
   Chạy npx hardhat run --network localhost scripts/deploy_token.js để biên dịch và triển khai hợp đồng mã thông báo của bạn. Sau khi thành công, bạn sẽ thấy thông báo này trên thiết bị đầu cuối: Đã hoàn tất việc viết địa chỉ hợp đồng mã thông báo: .... Sao chép giá trị đó và dán vào trường địa chỉ tokenAddr trong hợp đồng/exchange.sol.

5. Chạy npx hardhat run --network localhost scripts/deploy_exchange.js để triển khai hợp đồng trao đổi của bạn. Khi thành công, bạn sẽ thấy Viết xong địa chỉ hợp đồng trao đổi: .... Lưu địa chỉ đó cho bước tiếp theo.
   Bước 5 giống như Bước 4 nhưng thay vào đó là đối với trao đổi hợp đồng.sol. Địa chỉ hợp đồng mới được tạo sẽ được sao chép trong Bước 6.

6. Cập nhật địa chỉ hợp đồng và ABI trong web_app/exchange.js.
   Cập nhật các biến const token_address và const Exchange_address với hai địa chỉ hợp đồng. Không giống như trong Solidity, các địa chỉ trong Javascript không cần phải được kiểm tra tổng hợp.

   ABI có thể được sao chép từ các tạo tác/hợp đồng/token.sol/token.json và tạo tác/hợp đồng/exchange.sol/exchange.json. Để cập nhật chính xác ABI, vui lòng sao chép toàn bộ danh sách sau trường 'abi', bắt đầu từ dấu ngoặc vuông. Địa chỉ được lưu ở bước trước. Hãy chắc chắn rằng địa chỉ hợp đồng của bạn là một chuỗi.

7. Mở tệp web_app/index.html trong trình duyệt của bạn. Cho đến khi bạn thực hiện xong Phần 3 (hợp đồng/token.sol), sẽ có lỗi trong bảng điều khiển trình duyệt. Nếu không, bạn có thể nghịch trang và chạy sanityCheck! Xem Phần 10 để biết thêm chi tiết.

Lưu ý trên hệ điều hành: Tất cả các bước trên sẽ hoạt động trên các hệ thống dựa trên Unix và Windows. Các lệnh chúng tôi yêu cầu bạn thực thi sẽ hoạt động trong thiết bị đầu cuối Unix tiêu chuẩn và Dấu nhắc Lệnh của Windows.

2 thành phần

Dự án có ba hợp phần chính:

- hợp đồng/token.sol là một hợp đồng thông minh đư ợc viết bằng Solidity sẽ đư ợc triển khai trên chuỗi khối. Bạn sẽ phải sửa đổi nó để tạo và triển khai mã thông báo của riêng mình. Xem Phần 3 để biết chi tiết triển khai.
- hợp đồng/exchange.sol là một hợp đồng thông minh khác đư ợc viết bằng Solidity sẽ đư ợc triển khai trên chuỗi khối. Bạn sẽ phải sửa đổi nó để tạo và triển khai một sàn giao dịch phi tập trung (DEX), còn đư ợc gọi là nhà tạo lập thị trư ờng tự động (AMM), đư ợc mô phỏng theo Uniswap V1. Xem Phần 4 để biết chi tiết triển khai.
- web app/exchange.js là ứng dụng khách chạy cục bộ trong trình duyệt web đư ợc viết bằng JavaScript. Nó quan sát chuỗi khối bằng thư viện ethers.js và có thể gọi các chức năng trong hợp đồng thông minh token.sol và exchange.sol. Xem Phần 4 để biết thêm chi tiết. Để biết thêm thông tin về cách thức hoạt động của ethers.js và thiết lập nhiệm vụ này, hãy xem bản ghi Phần 5 trên Panopto.

Ngoài ra, tệp web app/index.html khi đư ợc mở trong trình duyệt của bạn (hoạt động tốt nhất trong Chrome) cho phép bạn truy cập vào giao diện ngư ời dùng mà qua đó bạn có thể kiểm tra trao đổi của mình. Trên trang này, bạn có thể chọn một địa chỉ và thêm thanh khoản, xóa thanh khoản và hoán đổi mã thông báo của mình lấy ETH và ngư ợc lại. Không nên sửa đổi tệp này và các tệp khác. Chỉ nên sửa đổi token.sol, Exchange.sol và Exchange.js.

3 Tạo và triển khai mã thông báo của riêng bạn

Trong phần đầu tiên của dự án này, bạn sẽ tạo và triển khai mã thông báo ERC-20 của riêng mình. ERC-20, như đã đề cập trong lớp, là một tiêu chuẩn để triển khai các mã thông báo có thể thay thế đư ợc. May mắn cho chúng tôi, phần lớn mã cho tiêu chuẩn ERC-20 đã đư ợc viết và là mã nguồn mở. Trong dự án này, chúng tôi sẽ sử dụng triển khai ERC-20 tiêu chuẩn từ dự án OpenZeppelin. Đảm bảo rằng bạn hiểu tiêu chuẩn ERC-20, cũng như việc triển khai OpenZeppelin của ERC-20 và Ownable.

Khi bạn đã đọc qua mã khởi động, hãy hoàn thành các bư ớc sau trong hợp đồng/token.sol và web_app/exchange.js:

1. Nghĩ ra một cái tên thú vị (như ng phù hợp!) cho mã thông báo của bạn. Bất kỳ tên nào cũng đư ợc, vì vậy hãy thoải mái vui chơ i và nghĩ ra thứ gì đó sáng tạo. Đặt chuỗi riêng tư _name của token.sol thành tên mã thông báo của bạn. Ngoài ra, hãy cập nhật biến token_name ở đầu Exchange.js thành cùng tên.
2. Quyết định ký hiệu ngắn cho mã thông báo của bạn (ví dụ: ETH, thay vì Ethereum). Đặt chuỗi riêng tư _symbol thành ký hiệu đó và cập nhật biến ký hiệu mã thông báo ở đầu Exchange.js có cùng tên.
3. Triển khai các hàm mint(uintmount) và vô hiệu hóa hàm mint(). bạc hà là một chức năng công cộng tạo ra số lư ợng mã thông báo. Mặc dù hợp đồng OpenZeppelin ERC-20 không cung cấp chức năng mint(), như ng bạn vẫn có thể gọi _mint() một cách thích hợp để làm như vậy. vô hiệu hóa_bạc hà () làm cho việc gọi bạc hà () sẽ không bao giờ thành công nữa. Như vậy, việc thực hiện đúc tiền của bạn phải

thất bại nếu vô hiệu hóa bạc hà () đã được gọi. Ngoài ra, hãy lưu ý công cụ sửa đổi Có thể sở hữu trên cả hai chức năng, điều này làm cho nó sao cho chỉ quản trị viên hợp đồng mới có thể gọi một trong hai chức năng. Mô hình bạc hà ()/ vô hiệu hóa bạc hà () là một cách đơn giản để tạo mã thông báo ban đầu và đảm bảo rằng tổng nguồn cung sẽ không đổi sau khi nguồn cung cấp mã thông báo được tạo.

4. Triển khai hợp đồng mã thông báo của bạn. Sao chép địa chỉ và ABI của hợp đồng mã thông báo vào địa chỉ mã_thông báo và các biến abi của mã thông báo trong Exchange.js.

5. Cuối cùng, sao chép địa chỉ của hợp đồng mã thông báo vào biến tokenAddr trong exchange.sol.
Hãy nhớ sử dụng phiên bản tổng kiểm tra của địa chỉ trong hợp đồng. Mỗi khi bạn triển khai lại hợp đồng mã thông báo của mình, bạn phải lặp lại bước này.

Sau khi hoàn thành các bước trên, bạn sẽ có tất cả mã thông báo của riêng mình được triển khai trên Hardhat!

Bây giờ bạn cũng có thể thử triển khai hợp đồng trao đổi của mình. Sao chép địa chỉ và ABI của hợp đồng mã thông báo sang địa chỉ trao đổi và trao đổi các biến abi trong Exchange.js. Tất cả các chức năng ngoại trừ thiết lập nhóm ban đầu (mà chúng tôi đã triển khai cho bạn) sẽ bị thiếu. Điều đó nói rằng, nếu bạn đã hoàn thành Phần 2 và quy trình triển khai đúng cách, bạn sẽ thấy không có lỗi nào xảy ra trong bảng điều khiển trình duyệt của mình và sẽ thấy tỷ giá hối đoái 1:1 giữa ETH và mã thông báo của bạn.
Cần có 5000 ETH và 5000 mã thông báo của bạn trong màn hình "Thanh khoản hiện tại".

4 Thiết lập Exchange cơ bản của bạn

Trong phần này của bài tập, bạn sẽ triển khai chức năng cơ bản của trao đổi tiền điện tử của mình. Trao đổi của chúng tôi được mô phỏng theo Uniswap V1. Trao đổi của bạn sẽ chỉ cho phép hoán đổi giữa Mã thông báo của bạn và ETH thử nghiệm. Những thay đổi trong phần này sẽ ảnh hưởng chủ yếu đến hai tệp: Exchange.js và Exchange.sol. Tự làm quen với mã khởi động cho các tệp đó.

Một sàn giao dịch phi tập trung bao gồm hai loại người tham gia: nhà cung cấp thanh khoản (LP) và nhà giao dịch. Đối với nhóm trao đổi nhất định giữa hai mã thông báo, các nhà cung cấp thanh khoản cung cấp một số lượng giá trị bằng nhau của cả hai loại thanh khoản (trong trường hợp của bạn là ETH và mã thông báo của riêng bạn). Khi các nhà giao dịch hoán đổi giữa hai loại tiền tệ, họ sẽ thêm một số lượng của một loại tiền tệ vào nhóm thanh khoản và sẽ được gửi một giá trị tương đương của loại tiền tệ kia từ nhóm. Tỷ giá hối đoái giữa hai đồng tiền được xác định theo công thức tích hằng số:

Gọi x là lượng tiền tệ A có trong nhóm thanh khoản và gọi y là lượng tiền tệ B. Gọi k là một hằng số nào đó. Sau mỗi lần hoán đổi, nó phải đúng là

$$x \cdot y = k.$$

Trong mỗi lần hoán đổi, sàn giao dịch phải gửi đúng số tiền được hoán đổi sang tiền tệ sao cho nhóm luôn duy trì trên đường sản phẩm không đổi. Giá của tiền tệ B tính theo tiền tệ A có thể được tính là x/y, trong khi giá của tiền tệ A tính theo tiền tệ B có thể được tính là y/x. Do đó, mỗi lần hoán đổi sẽ làm thay đổi tỷ giá hối đoái. Điều này có ý nghĩa, vì mỗi lần hoán đổi là một dấu hiệu cho thấy nhu cầu đối với một loại tiền tệ nhất định. Tác động của mỗi lần hoán đổi đối với giá của các loại tiền tệ sẽ có liên quan trong phần 6.

Khi một nhà cung cấp thanh khoản thêm thanh khoản, họ phải cung cấp các giá trị bằng nhau của tiền tệ A và B, được xác định theo tỷ giá hối đoái hiện tại. Sau đó lưu ý rằng việc thêm thanh khoản sẽ không thay đổi

tỷ giá hối đoái giữa các đồng tiền, như ng sẽ làm tăng giá trị của hằng số k. Tư ơ ng tự, khi một nhà cung cấp thanh khoản rút thanh khoản của họ, họ phải rút các giá trị bằng nhau của tiền tệ A và tiền tệ B. Do đó, k sẽ giảm như ng tỷ giá hối đoái sẽ không đổi.

Điều này có một hệ quả đáng chú ý khác: vì nhà cung cấp thanh khoản chỉ có thể rút các giá trị bằng nhau của mỗi loại tiền tệ, nên họ không thực sự có quyền rút chính xác khoản đầu tư ban đầu của mình (về số lư ợng của mỗi mã thông báo). Thay vào đó, việc cung cấp tính thanh khoản không đồng nghĩa với việc sở hữu một tỷ lệ phần trăm của nhóm thanh khoản, sau đó nhà cung cấp có quyền rút lại sau đó. Nhà cung cấp thanh khoản đã cung cấp 10% tiền tệ A và tiền tệ B có quyền rút 10% của mỗi khoản dự trữ cho các loại tiền tệ đó (giả sử tỷ lệ phần trăm của chúng không bị pha loãng bởi các nhà cung cấp khác), ngay cả khi 10% không khớp chính xác với số lư ợng của mỗi mã thông báo họ đã cung cấp. Bằng cách này, các nhà cung cấp thanh khoản có thể bị lỗ tạm thời, nếu giá trị của những gì họ đư ợc quyền rút thấp hơ n giá trị mà họ đã đầu tư ban đầu. Nếu bạn không quen với mất mát vô thư ờng, vui lòng xem lại bài giảng và phần về trao đổi phi tập trung.

Với suy nghĩ trên, bây giờ bạn sẽ triển khai chức năng cơ bản của sàn giao dịch của mình. Chúng tôi đảm nhận việc khởi tạo nhóm cho bạn bằng cách triển khai và gọi hàm createPool. Chúng tôi lấy ETH và mã thông báo từ địa chỉ đầu tiên để khởi tạo nhóm và bạn không cần theo dõi số tiền ban đầu này khi theo dõi các nhà cung cấp thanh khoản. Để các địa chỉ khác nhận đư ợc mã thông báo và/hoặc cung cấp tính thanh khoản, trư ớc tiên họ phải hoán đổi mã thông báo trên sàn giao dịch. Trong Exchange.sol, thực hiện các chức năng sau:

- chức năng addLiquidity() phải trả bên ngoài: Thêm
  thanh khoản vào nhóm nếu nhà cung cấp sở hữu đủ ETH và mã thông báo (nếu không thì giao dịch sẽ không thành công). Ngư ời gọi sẽ gửi ETH đến hợp đồng, có thể đư ợc truy cập bằng cách sử dụng msg.value. Chức năng này cũng sẽ chuyển số lư ợng mã thông báo tư ơ ng đư ơ ng dựa trên tỷ giá hối đoái hiện tại từ địa chỉ của ngư ời gửi sang hợp đồng (sử dụng phư ơ ng thức chuyển hoặc chuyển từ mã thông báo) và cập nhật trạng thái trao đổi tư ơ ng ứng. Giao dịch phải thất bại nếu tiền của nhà cung cấp không đủ. Xem Phần 9 để đư ợc tư vấn về cách theo dõi thanh khoản tốt nhất.

- chức năng removeLiquidity(uintmountETH) phải trả công khai:
  Xóa một lư ợng thanh khoản nhất định khỏi nhóm (nếu nhà cung cấp có quyền loại bỏ một lư ợng thanh khoản nhất định) và cập nhật trạng thái trao đổi tư ơ ng ứng. lư ợngETH là số lư ợng ETH mà nhà cung cấp thanh khoản muốn rút ra, vì vậy họ sẽ nhận đư ợc tổng giá trị tư ơ ng đư ơ ng 2 * lư ợngETH sau khi họ nhận đư ợc mã thông báo và ETH. Đảm bảo cập nhật lư ợng thanh khoản đư ợc cung cấp bởi từng nhà cung cấp thanh khoản tư ơ ng ứng. Chức năng này sẽ không thành công nếu ngư ời dùng cố gắng loại bỏ nhiều thanh khoản hơ n mức họ có quyền hoặc nếu họ cố gắng làm cạn kiệt dự trữ ETH hoặc mã thông báo về 0.

- chức năng removeAllLiquidity() phải trả bên ngoài:
  Xóa lư ợng thanh khoản tối đa mà ngư ời gửi đư ợc phép xóa và cập nhật trạng thái trao đổi tư ơ ng ứng. Ngoài ra, hãy nhớ cập nhật lư ợng thanh khoản đư ợc cung cấp bởi mỗi nhà cung cấp thanh khoản. Tư ơ ng tự, chức năng này sẽ không thành công nếu nhà cung cấp thanh khoản rút ETH hoặc dự trữ mã thông báo về 0.

- hàm swapTokensForETH(uintmountTokens) phải trả bên ngoài:
  Hoán đổi số lư ợng mã thông báo nhất định lấy giá trị tư ơ ng đư ơ ng của ETH và cập nhật trạng thái trao đổi tư ơ ng ứng. Nếu nhà cung cấp không có đủ mã thông báo để hoán đổi, giao dịch

nên thất bại. Ngoài ra, nếu việc hoàn tất hoán đổi sẽ xóa hoàn toàn tất cả ETH khỏi nhóm, thì giao dịch sẽ không tránh khỏi việc không có ETH và (do đó) tỷ giá hối đoái không xác định. Đảm bảo luôn để lại ít nhất 1 ETH và 1 mã thông báo trong nhóm.

- chức năng hoán đổiETHForTokens() phải trả bên ngoài:
  Hoán đổi số lượng ETH nhất định lấy giá trị tương đương trong mã thông báo của bạn và cập nhật trạng thái trao đổi tương ứng. Tương tự như addLiquidity(), người gửi sẽ gửi ETH vào hợp đồng, có thể được truy cập thông qua msg.value. Nếu việc hoàn tất hoán đổi sẽ xóa hoàn toàn tất cả các mã thông báo khỏi nhóm, thì giao dịch sẽ không tránh khỏi việc không có mã thông báo và (do đó) tỷ giá hối đoái không xác định. Đảm bảo luôn để lại ít nhất 1 ETH và 1 mã thông báo trong nhóm.

Trong mỗi chức năng trên, hãy đảm bảo rằng bạn đang điều chỉnh dự trữ mã thông_báo, dự trữ eth và/hoặc k theo cách chính xác sao cho việc trao đổi luôn nằm trên đường cong sản phẩm không đổi được mô tả ở trên. Ngoài ra, hãy chắc chắn rằng các chức năng không thành công khi người gọi không có đủ tiền. Cuối cùng, hãy nhớ đặt địa chỉ tokenAddr làm địa chỉ của hợp đồng mã thông báo đã triển khai của bạn.
Bây giờ bạn có thể chạy npx hardhat run --network localhost scripts/deploy exchange.js để gỡ lỗi và triển khai mã của bạn.

Xử lý lỗi làm tròn số. Trao đổi mã thông báo theo quy tắc sản phẩm không đổi liên quan đến việc thực hiện các hoạt động như cộng và chia, và điều này chắc chắn dẫn đến lỗi làm tròn số.
Điều quan trọng là đảm bảo rằng những lỗi này không thể phức tạp. Ví dụ, điều này có thể dẫn đến tổn thất cho các nhà cung cấp thanh khoản nếu nhiều mã thông báo hơn một chút so với dự kiến được rút khỏi nhóm tại mỗi lần hoán đổi.

Kết quả là, tích x y có thể không hoàn toàn bằng k do một số lỗi làm tròn - đây không phải là vấn đề - nhưng vì k hoàn toàn không bị sửa đổi nên các lỗi này không cộng lại mà tự bù trừ khi có nhiều phép hoán đổi thực hiện. Lỗi làm tròn cũng có thể xảy ra khi thêm hoặc rút thanh khoản vì mã thông báo và ETH của bạn không thể được chia xa hơn 1 đơn vị (ít nhất là trong dự án này, hãy xem phần 8 để biết thêm chi tiết). Chúng tôi không mong đợi bạn xử lý các loại lỗi làm tròn này trong dự án này. Do đó, sẽ ổn nếu x y lệch khỏi k khi thực hiện phép tính số học, cũng như các phép làm tròn tương tự khác khi tính toán tính thanh khoản. Tuy nhiên, điều quan trọng vẫn là cập nhật k ở những nơi thích hợp.

5 Triển khai chương trình phụ trợ

Sau khi bạn hoàn thành việc triển khai các chức năng của hợp đồng, hãy triển khai các chức năng sau trong exchange.js. Bạn có thể bỏ qua biến maxSlippagePct lúc này – biến này sẽ được sử dụng trong Phần 6. Phần lớn, chúng sẽ chỉ gọi mã thông báo và các hàm trao đổi mà bạn đã viết ở trên:

- chức năng không đồng bộ addLiquidity(amountEth, maxSlippagePct)
- chức năng không đồng bộ removeLiquidity(amountEth, maxSlippagePct)
- chức năng không đồng bộ removeAllLiquidity(maxSlippagePct)
- chức năng không đồng bộ swapTokensForETH(amountToken, maxSlippagePct)
- chức năng không đồng bộ swapETHForTokens(amountEth, maxSlippagePct)

Bạn có thể gọi mã hợp đồng với await contract.functionName(args) hoặc đang chờ hợp đồng.connect(anotherSigner).functionName(args). Để được trợ giúp thêm về cú pháp, chúng tôi đặc biệt khuyến khích bạn xem tài liệu ethers.js, cũng như Phần 9 để biết một số mẹo.

Sau khi bạn đã triển khai đầy đủ hợp đồng thông minh của mình và mã JavaScript tương ứng, hãy cập nhật biến abi mã thông báo và trao đổi abi ở đầu tệp, đồng thời sao chép địa chỉ hợp đồng vào địa chỉ mã thông báo và các biến địa chỉ trao đổi trong Exchange.js. Đảm bảo bao gồm các dấu ngoặc ngoài cùng khi sao chép ABI. Nếu bạn tải lại index.html, giờ đây bạn có thể cung cấp thanh khoản, xóa thanh khoản và thực hiện giao dịch hoán đổi.

6 Xử lý Trượt

Có một vấn đề quan trọng với sàn giao dịch của chúng tôi khi chúng tôi triển khai nó trong Phần 3, vì nó không tính đến "trượt giá". Hãy nhớ lại rằng với mỗi lần hoán đổi trên một sàn giao dịch phi tập trung, giá của mỗi tài sản sẽ thay đổi một chút. Vì nhiều người dùng có thể đang cố gắng hoán đổi tiền tệ cùng một lúc trên một sàn giao dịch phi tập trung, nên có thể có sự thay đổi về tỷ giá hối đoái giữa việc gửi một giao dịch hoán đổi và quá trình xử lý thực tế của giao dịch đó. Sự thay đổi tỷ giá hối đoái này giữa giá niêm yết và giá thực tế của sàn giao dịch được gọi là "trượt giá". Trượt giá là mối quan tâm đặc biệt khi giao dịch các tài sản dễ bay hơi. Ví dụ: nếu người dùng gửi giao dịch hoán đổi để hoán đổi một số lượng tiền tệ A lấy tiền tệ B và sau đó giá của tiền tệ B tăng đáng kể so với giá báo giá, người dùng có thể không thực sự muốn hoàn thành giao dịch hoán đổi.

Ngoài ra, việc xử lý không đầy đủ sẽ mở ra cho người dùng một kiểu tấn công được gọi là bánh sandwich tấn công. Một cuộc tấn công bánh sandwich hoạt động như sau:

1. Alice gửi một giao dịch hoán đổi để chuyển đổi một lượng lớn tiền tệ A thành tiền tệ b.

2. Đối thủ nhìn thấy giao dịch của Alice và chạy trước giao dịch đó bằng một giao dịch mua tiền tệ rất lớn B, do đó làm tăng giá của tài sản B.

3. Alice mua tiền tệ B với mức giá mới cao hơn, thậm chí còn tăng giá tiền tệ B hơn nữa.

4. Sau đó, kẻ tấn công ngay lập tức bán tất cả đồng tiền B mới mua của họ với giá cao hơn, kiếm lợi nhuận nhanh chóng.

Tính dễ bị tấn công kiểu bánh sandwich có hại cho người dùng của một sàn giao dịch phi tập trung, vì người dùng luôn trả tỷ giá hối đoái cao hơn giá trị tài sản thực. Do đó, điều quan trọng là chúng ta phải nâng cấp sàn giao dịch của mình để xử lý trượt giá đúng cách và chống lại các cuộc tấn công kiểu bánh sandwich.

Biện pháp bảo vệ phổ biến nhất chống lại các cuộc tấn công kiểu bánh sandwich là cho phép người dùng đặt một số mức trượt giá tối đa trong khi gửi giao dịch. Tham số này, thường là tỷ lệ phần trăm, sẽ khiến giao dịch không thành công nếu giá của tài sản thay đổi nhiều hơn mức trượt tối đa cho phép.
Điều này hạn chế thiệt hại có thể gây ra bởi các cuộc tấn công bánh sandwich và bảo vệ người dùng đang hoán đổi tài sản dễ bay hơi.

Để thực hiện yêu cầu trượt giá tối đa, hãy thực hiện các bước sau:

1. Trong exchange.sol, hãy cập nhật các hàm swapTokensForETH và swapETHForTokens của bạn để nhận tham số tỷ giá hối đoái tối đa uint. Bạn cũng có thể chuyển vào các tham số khác nếu cần thiết cho thiết kế của mình. Trong khi hoán đổi, hoán đổi sẽ không thành công nếu giá hiện tại của tài sản mới (tức là tài sản mà người dùng đang hoán đổi) đã tăng lên nhiều hơn mức trao đổi tối đa

tỷ lệ. Lưu ý rằng giá của tài sản giảm là tốt cho người dùng, vì vậy chúng tôi không phải thất bại trong trường hợp đó.

2. Cập nhật các hàm addLiquidity, removeLiquidity và removeAllLiquidity để tiếp nhận tham số tỷ giá hối đoái_uint max và uint min. Bạn cũng có thể chuyển vào các tham số khác nếu cần thiết cho thiết kế của mình. Trong khi cung cấp tính thanh khoản, giao dịch sẽ thất bại nếu giá hiện tại của tài sản mới (tức là tài sản mà người dùng đang hoán đổi) đã tăng lên hơn tỷ giá hối đoái tối đa hoặc giảm xuống dưới tỷ giá hối đoái tối thiểu. Giá thay đổi đột ngột theo một trong hai hướng có thể khiến các nhà cung cấp chịu tổn thất tạm thời trước khi họ gửi thanh khoản, vì vậy chúng tôi muốn các nhà cung cấp thanh khoản chỉ định tỷ giá hối đoái tối đa và tối thiểu.

3. Bây giờ hãy cập nhật tệp Exchange.js của bạn để liên lạc với hợp đồng về tỷ giá hối đoái tối đa/tối thiểu. Tham số maxSlippagePct được cung cấp, đại diện cho phần trăm thay đổi giá tối đa được phép trước khi giao dịch không thành công. Trong thử nghiệm và trong giao diện trình duyệt, tham số này được truyền dưới dạng int, không phải dưới dạng float - tức là 4% được truyền dưới dạng 4, không phải 0,04.
Tham số này có thể được sử dụng trong mỗi hàm JavaScript để tính toán các giá trị chính xác cho tỷ giá hối đoái_tối đa và/hoặc tỷ giá hối đoái tối thiểu, sau đó có thể được chuyển đến hợp đồng.
Hàm getPoolState mà chúng tôi cung cấp cho bạn có thể hữu ích ở đây.

Như mọi khi, sau khi cập nhật hợp đồng của bạn, hãy đảm bảo biên dịch lại, triển khai lại và sao chép ABI mới cũng như địa chỉ hợp đồng vào biến ở đầu tệp Exchange.js của bạn. Tại thời điểm này, bạn cũng có thể bỏ ghi chú hàm sanityCheck() để kiểm tra việc triển khai của mình. Xem Phần 10 để biết thêm chi tiết về sanityCheck.

7 Nhà cung cấp thanh khoản thưởng

Sau khi hoàn thành các phần trên, giờ đây bạn đã có một sàn giao dịch đang hoạt động cho phép người dùng giới hạn mức độ trượt giá mà họ muốn chịu đựng! Tuy nhiên, có một vấn đề lớn hơn. Chúng tôi đã thảo luận nhiều lần về cách các nhà cung cấp thanh khoản chấp nhận rủi ro dưới hình thức thua lỗ tạm thời. Nghĩa là, giá trị cổ phần thanh khoản của họ có thể giảm nếu giá của một trong hai tài sản thay đổi. Trên thực tế, vì nhiều loại tiền điện tử khá biến động nên đây là mức độ rủi ro mà không nhà cung cấp thanh khoản nào sẵn sàng nhận miễn phí.

Do đó, chúng ta cần khuyến khích các nhà cung cấp thanh khoản cung cấp thanh khoản cho nhóm. Trong các sàn giao dịch trong thế giới thực, các nhà cung cấp thanh khoản được khuyến khích cung cấp thanh khoản vì họ nhận được một khoản phí nhỏ từ mỗi giao dịch hoán đổi. Các khoản phí này được tự động tái đầu tư vào nhóm thanh khoản thay mặt cho từng nhà cung cấp thanh khoản. Khi một nhà cung cấp rút thanh khoản của họ, số tiền họ có quyền rút bao gồm tất cả các khoản phí mà họ đã được thưởng kể từ khi cung cấp thanh khoản.

Bây giờ, bạn sẽ triển khai chương trình thưởng phí tương tự cho các nhà cung cấp thanh khoản. Bạn có thể tự do thiết kế của riêng mình miễn là nó đáp ứng các yêu cầu được nêu ở cuối phần này.
Tuy nhiên, chúng tôi đặc biệt đề xuất cách sau, được giải thích bằng văn bản thuần túy.

- Mỗi nhà cung cấp thanh khoản có quyền sở hữu một phần f của nhóm khai thác. Phần này được lưu trữ trong hợp đồng thông minh cho mỗi nhà cung cấp thanh khoản (nghĩa là một mảng lưu trữ cho mỗi địa chỉ phần của nhóm khai thác thuộc sở hữu của địa chỉ này). Đối với nhiệm vụ này, chúng tôi sẽ đặt phí hoán đổi thành 3% để hỗ trợ công cụ tự động chấm điểm, như được biểu thị bằng tử số phí hoán đổi _ _

và hoán đổi các trường mẫu số phí. Tỷ lệ sở hữu của nhà cung cấp thanh khoản sẽ không thay đổi khi thực hiện hoán đổi, vì phần thưởng thanh khoản được phân phối dựa trên tỷ lệ sở hữu của mỗi lp. Lưu ý rằng do phí, tổng thanh khoản l = √xy tăng lên, và do đó k sẽ thay đổi một chút. Điều này mang lại lợi nhuận cho các nhà cung cấp thanh khoản nếu tỷ giá hối đoái không thay đổi và giúp chống lại tổn thất tạm thời.

- Như đã đề cập ở trên, do phần thưởng thanh khoản, giờ đây k sẽ thay đổi một chút trên mỗi lần hoán đổi. Tuy nhiên, với mục đích của dự án này, bạn chỉ nên thay đổi k khi thêm hoặc bớt thanh khoản. Vì vậy, bạn không cần phải lo lắng về những thay đổi nhỏ này trong k do sự thay đổi của x y trong quá trình hoán đổi.

- Khi một nhà cung cấp thanh khoản rút thanh khoản của mình, họ sẽ nhận được một phần f của cả hai mã thông báo trong nhóm tương ứng với phần sở hữu của họ, cũng như phần thưởng của họ. Các nhà cung cấp thanh khoản khác có tỷ lệ sở hữu tăng tương ứng. Tất cả các phân số nên cộng lại thành 1.

- Khi nhà cung cấp thanh khoản thêm thanh khoản vào nhóm, họ sẽ nhận được phần quyền sở hữu f trên nhóm bằng với tỷ lệ mã thông báo của họ ở trạng thái mới của nhóm. Các nhà cung cấp thanh khoản khác có tỷ lệ sở hữu bị thu hẹp tương ứng. Một lần nữa, tất cả các phân số phải cộng lại bằng 1.

Nếu bạn đã triển khai theo dõi thanh khoản bằng phân số, thì phần này sẽ không tốn quá nhiều công sức. Ngoài ra, chúng tôi cũng sẽ chấp nhận bất kỳ thiết kế nào đáp ứng các yêu cầu được liệt kê bên dưới.

Yêu cầu về phần thưởng thanh khoản:

1. Nhóm của bạn phải tính phí cho người thực hiện giao dịch hoán đổi một số khoản phí phần trăm khác không trên mỗi giao dịch hoán đổi.1 Bạn có thể xác định giá trị này bằng cách sử dụng tử số phí hoán đổi và mẫu số phí hoán đổi, như đã giải thích ở trên.

2. Khi một giao dịch hoán đổi xảy ra, giá trị của mã thông báo hoặc ETH được gửi cho người giao dịch phải bằng (1 p) nhân với giá trị của tài sản mà họ đang hoán đổi, trong đó p là phần trăm phí mà nhà cung cấp thanh khoản phải trả. Ví dụ: nếu phí là 1% và người dùng đang hoán đổi 100 ETH lấy mã thông báo của bạn, thì họ chỉ được gửi số tiền tương đương 99 ETH.

3. Khi một khoản phí được thực hiện trong quá trình hoán đổi, khoản phí đó sẽ được phân phối cho các nhà cung cấp thanh khoản để mỗi nhà cung cấp sau đó có thể rút số tiền đặt cọc ban đầu cộng với phí của họ. Các khoản phí phải được phân bổ theo tỷ lệ dựa trên tỷ lệ phần trăm của nhà cung cấp trong nhóm thanh khoản tại thời điểm diễn ra quá trình hoán đổi. Ví dụ: sẽ không chính xác nếu một nhà cung cấp thanh khoản đã cung cấp một nửa tổng số thanh khoản trong nhóm tại thời điểm t được phép rút một nửa số phí đã thực hiện trước thời điểm t. Các nhà cung cấp thanh khoản không cần phải thực hiện bất kỳ bước bổ sung nào để yêu cầu các khoản phí của họ ngoài việc gọi removeLiquidity. Ngoài ra, phần thưởng thanh khoản không nên được gửi ra khỏi sàn giao dịch cho các nhà cung cấp mỗi khi một giao dịch hoán đổi diễn ra, vì làm như vậy sẽ rất tốn kém trong thực tế.

1Để tham khảo, phí mặc định trên Uniswap là 0,3%, trong khi các sàn giao dịch tập trung thường tính phí khoảng 1-4% để hoán đổi tiền tệ.

4. Trong khi quyết định giữa các phư ơ ng án thiết kế khác nhau, chúng tôi khuyến khích bạn chọn giải pháp giảm thiểu chi phí gas. Chúng tôi sẽ không chấm điểm nghiêm ngặt về việc sử dụng gas; tuy nhiên, bạn sẽ đư ợc yêu cầu chứng minh các quyết định thiết kế của mình trong tài liệu thiết kế ở Phần 6.

Sau khi thiết kế và thực hiện phần trên, bạn nên có một cuộc trao đổi làm việc đầy đủ!
Chúc mừng! Kiểm tra các chức năng của bạn bằng cách sử dụng giao diện ngư ời dùng đư ợc cung cấp trong index.html hoặc viết mã kiểm tra bằng JavaScript. Việc thực hiện dự án này thể hiện một thành tích rất ấn tư ợng, vì vậy hãy tự khen ngợi bản thân. Trên thực tế, với một số sửa đổi bảo mật, bạn có thể triển khai cả mã thông báo và trao đổi của mình trên mạng chính Ethereum và do đó có một trao đổi mà bạn có thể gọi là của riêng mình!

8 Lư u ý về Solidity và Javascript Decimals

Không giống như hầu hết các ngôn ngữ lập trình, Solidity không hỗ trợ số học dấu phẩy động. Do đó, tất cả các mã thông báo ERC-20 đều theo dõi một biến số thập phân, biến số này cho biết có bao nhiêu số thập phân ở bên trái mà các số đó sẽ đư ợc hiểu là. Ví dụ: ether sử dụng 18 điểm thập phân, vì vậy 1 ETH sẽ đư ợc biểu thị là 1018 trong hợp đồng. Tư ơ ng tự, 1 wei = 10 18 ETH, vì vậy 1 wei chỉ đư ợc biểu diễn bằng 1. Thật không may, Javascript cũng có giới hạn về số nguyên lớn như thế nào: Numbers.MAX INT = 9 1015. Để cân bằng giữa hai giá trị này, trong trao đổi của chúng tôi, chúng tôi khởi tạo nhóm để có 1010 mã thông báo, có nghĩa là nhóm bắt đầu với 10 8 ETH và 10 8 mã thông báo của bạn.

Hậu quả của việc Solidity không hỗ trợ các thao tác float là tất cả các số thập phân sẽ bị cắt bớt. Ví dụ: 5/2 = 2. Đây là lý do tại sao cần có biến số thập phân, vì đại diện cho 5 ETH là 5 1018 sẽ dẫn đến 5 1018/2 = 25 1016 và vì chúng tôi biết có 18 chữ số thập phân, nên chúng tôi có thể thấy rằng điều này tư ơ ng ứng với 2,5 ETH, như mong muốn. Tuy nhiên, trong một số trư ờng hợp, hiện tư ợng tràn thừa vẫn có thể xảy ra, đặc biệt nếu tử số nhỏ hơ n mẫu số trong các phép tính và do đó dẫn đến 0. Trong trư ờng hợp này, bạn cần cẩn thận với thứ tự các phép tính, chẳng hạn như nhân hoặc cộng đầu tiên trư ớc khi chia. Nói chung, nên trì hoãn phép chia càng muộn càng tốt để tránh gặp phải lỗi làm tròn này.

Trong dự án này, chúng tôi sẽ không kiểm tra bạn trên tràn và chúng tôi đã cố gắng hết sức để loại bỏ tất cả các số thập phân. Do đó, khi kiểm tra trao đổi của chúng tôi, chúng tôi sẽ chọn các giá trị không khiến trao đổi của bạn vư ợt quá INT MAX của Javascript. Tuy nhiên, chúng tôi sẽ kiểm tra việc triển khai của bạn để đảm bảo rằng bạn đã giải thích đư ợc việc thiếu hụt trong trao đổi của mình.

9 Tư vấn Thực hiện

Mặc dù thiết kế tổng thể của hợp đồng của bạn là kết thúc mở, đây là một số lời khuyên có thể giúp hợp lý hóa quy trình thực hiện của bạn:

1. Theo dõi tỷ lệ của nhà cung cấp thanh khoản, thay vì giá trị tuyệt đối.
   Ví dụ: nếu có 1000 ETH và 1000 mã thông báo trong nhóm và Alice cung cấp 500 ETH và 500 mã thông báo, thì Alice đư ợc hư ởng 1/3 của nhóm. Thay vì lư u trữ 500 ETH cho thanh khoản của cô ấy, chúng tôi khuyên bạn nên lư u trữ mà cô ấy sở hữu 1/3. Trong quá trình hoán đổi, giá trị này sẽ không thay đổi (ngay cả khi triển khai phần thư ởng lp). Tuy nhiên, khi nhà cung cấp thanh khoản thêm hoặc xóa thanh khoản, giá trị này (và tất cả các phân số của nhà cung cấp thanh khoản khác) sẽ đư ợc cập nhật tư ơ ng ứng. Ngoài ra, vì Solidity không hỗ trợ số thập phân có dấu phẩy động nên chúng tôi

khuyên bạn nên sửa mẫu số (tức là 100, 1000, v.v.) để bạn có thể lưu trữ tử số của phần trăm dư ới dạng uint.

2. Xử lý lỗi bo tròn cạnh. Một lần nữa, vì Solidity không hỗ trợ dấu phẩy động, chúng tôi khuyên bạn nên thực hiện phép nhân trư ớc khi chia bất cứ khi nào có thể. Điều này tránh chạy vào phép chia làm tròn thành 0, rồi nhân lại để nhận đư ợc 0. Tư ơ ng tự, với các loại uint, hãy đảm bảo thực hiện phép cộng trư ớc phép trừ khi có thể để tránh bị tràn.

3. Phê duyệt chuyển mã thông báo. Để địa chỉ của bên thứ ba (ví dụ: hợp đồng) gửi hoặc nhận mã thông báo thay mặt bạn, trư ớc tiên bạn phải cấp cho họ quyền bằng cách sử dụng chức năng phê duyệt () của hợp đồng mã thông báo. Chức năng này cần đư ợc khởi tạo bởi ngư ời dùng, bạn sẽ không thể chạy phê duyệt() từ chính hợp đồng. Vì vậy, hãy đảm bảo gọi chức năng này trong Javascript trư ớc khi gọi chức năng trao đổi khi thích hợp. Chi tiết về chức năng này có thể đư ợc tìm thấy trong triển khai ERC20 của Openzeppelin.

4. Gửi ETH đến và từ Hợp đồng. Để chuyển thành công ETH từ tài khoản ngư ời dùng sang hợp đồng, chức năng xử lý chuyển khoản phải đư ợc đánh dấu là phải trả.
Điều quan trọng cần lư u ý là chỉ cần chỉ định một đối số trong hàm hợp đồng để chỉ định số lư ợng ETH sẽ không chuyển ETH; thay vào đó, ETH đư ợc chuyển qua tham số msg.value. Để tránh số thập phân, vui lòng sử dụng ethers.utils.parseUnits() với các đơ n vị đư ợc đặt thành "wei" khi chuyển ETH vào hợp đồng. Tư ơ ng tự, nếu bạn muốn gửi ETH từ hợp đồng đến địa chỉ ngư ời dùng, thì bạn có thể sử dụng hàm pay() một cách thích hợp. Nhiều thông tin thêm có thể đư ợc tìm thấy ở đây.

5. Lặp lại thông qua các phím ánh xạ. Solidity không hỗ trợ lặp qua các phím của ánh xạ. Do đó, chúng tôi xác định địa chỉ[] nhà cung cấp lp riêng để bạn lư u trữ địa chỉ của nhà cung cấp thanh khoản. Ngoài ra, hãy nhớ rằng các mảng Solidity không tự động "dịch chuyển" tất cả các phần tử khi một giá trị ở giữa mảng bị loại bỏ. Do đó, chúng tôi cung cấp một hàm trợ giúp removeLP() để xóa nhà cung cấp thanh khoản khỏi mảng trong khi lấp đầy “khoảng trống”.
Điều này sẽ cập nhật lp provider.length tư ơ ng ứng. Tuy nhiên, hãy cẩn thận khi gọi hàm này trong khi bạn đang lặp qua mảng, vì sẽ rất rủi ro khi thay đổi độ dài của mảng trong khi lặp qua nó nếu bạn muốn tiếp cận mọi phần tử.

6. Truy cập địa chỉ hợp đồng. Bạn có thể lấy địa chỉ của hợp đồng trong Solidity bằng cách gọi địa chỉ (này).

7. Truy cập số lư ợng token và ETH của hợp đồng. Mặc dù bạn có thể tính toán số lư ợng token và ETH của hợp đồng theo cách thủ công trong khi thực hiện các phép tính trung gian, như ng bạn có thể truy cập số dư thực bằng các chức năng sau:

   - Số dư ETH: địa chỉ(this).balance
   - Số dư mã thông báo: token.balanceOf(address(this))

   Chúng tôi thực sự khuyên bạn nên sử dụng các giá trị thực này để tính toán k thay vì sử dụng dự trữ mã thông báo và dự trữ eth, đồng thời kiểm tra lại công việc của bạn.

8. Chức năng không đồng bộ Javascript. Trong Javascript, có nhiều lần các chức năng đư ợc chạy không đồng bộ. Nghĩa là, nếu mã của bạn gọi một hàm không đồng bộ, thì mã sẽ tiếp tục chạy qua dòng đó mà không cần đợi hàm không đồng bộ đó thực thi xong. Theo mặc định,

Các hàm async trả về một đối tượng Promise, chỉ định một số giá trị sẽ được trả về khi kết thúc thực thi hàm. Để mã của bạn chờ chức năng async chạy và nhận kết quả, bạn sẽ cần sử dụng từ khóa chờ đợi. Bạn có thể tìm hiểu thêm về các chức năng không đồng bộ Javascript tại đây.

Điều quan trọng cần lưu ý là tất cả các lệnh gọi hàm Solidity sẽ không đồng bộ với phần phụ trợ. Vì vậy, trong Exchange.js, để gọi một hàm từ hợp đồng và nhận đầu ra, hãy đảm bảo sử dụng từ khóa đang chờ, chẳng hạn như var num = đang chờ hợp đồng mã thông báo.function(args).

9. Bảo mật Hợp đồng. Hãy nhớ rằng, vì tất cả các hợp đồng đã triển khai đều được công khai trên chuỗi nên điều quan trọng là phải giữ an toàn cho hợp đồng của bạn. Mặc dù chúng tôi sẽ không tích cực kiểm tra hợp đồng của bạn về tính bảo mật, nhưng chúng tôi vẫn hy vọng sẽ thấy một số biện pháp phòng vệ thông qua việc sử dụng các câu lệnh require() hoặc assert().

10 Kiểm tra vệ sinh

Để kiểm tra việc triển khai của bạn, chúng tôi đã triển khai hai chương trình kiểm tra độ chính xác chạy tùy thuộc vào việc bạn có triển khai phần thưởng thanh khoản hay không. Chúng tôi kiểm tra giá trị này bằng cách đọc giá trị tử số phí hoán đổi trong exchange.sol: nếu giá trị đó bằng 0, thì chúng tôi cho rằng bạn chưa triển khai phí hoán đổi và phần thưởng thanh khoản. Bạn sẽ không nhận được tín dụng đầy đủ cho đến khi phần thưởng thanh khoản được triển khai đầy đủ, nhưng chúng tôi cũng muốn cung cấp tín dụng để hoàn thành việc triển khai cơ bản dự án này.

Để bật tính năng kiểm tra độ chính xác, hãy bỏ ghi chú hàm setTimeout() bằng tính năng kiểm tra độ chính xác() và làm mới trang. Chúng tôi đã cố gắng thiết kế sanityCheck để chạy đúng cách ngay cả sau lần tải đầu tiên, vì vậy bạn không cần phải triển khai lại các hợp đồng và đặt lại trạng thái nhóm mỗi khi bạn muốn chạy sanityCheck. Tuy nhiên, do lỗi làm tròn, có thể có một điểm mà sanityCheck vượt qua khi tỷ giá hối đoái là 1:1 nhưng không phải với trạng thái hối đoái hiện tại của bạn. Điểm số sanityCheck của bạn cho dự án này sẽ dựa trên trạng thái nhóm ban đầu – nghĩa là tỷ giá hối đoái giữa các mã thông báo là 1:1 và có 5000 ETH và 5000 mã thông báo trong nhóm.

11 Tài liệu thiết kế

Vui lòng điền vào DesignDoc.txt câu trả lời của bạn cho các câu hỏi sau:

1. Giải thích tại sao việc thêm và xóa thanh khoản vào sàn giao dịch của bạn không làm thay đổi sàn giao dịch tỷ lệ.

2. Giải thích kế hoạch thưởng cho các nhà cung cấp thanh khoản của bạn và biện minh cho các quyết định thiết kế mà bạn đã đưa ra. Nó đáp ứng các yêu cầu về phần thưởng thanh khoản được nêu trong Phần 7 như thế nào?

3. Mô tả ít nhất một phương pháp bạn đã sử dụng để giảm thiểu việc sử dụng gas trong hợp đồng trao đổi của mình. Tại sao phương pháp này có hiệu quả không?

4. Phản hồi tùy chọn

   (a) Bạn đã dành bao nhiêu thời gian cho bài tập? (b) Bạn nên biết một điều gì trước khi bắt đầu nhiệm vụ?

(c) Nếu bạn có thể thay đổi một câu về bài tập này, bạn sẽ thay đổi điều gì?

(d) Vui lòng bao gồm bất kỳ phản hồi nào khác mà bạn có thể có.