Project 1: Dùng solidity, ethers.js tạo Dapp và Smart contract

1. Blockchain Splitwise

Đề bài yêu cầu tạo ra một hệ thống phi tập trung để theo dõi nợ và có của - một phiên bản blockchain của ứng dụng Splitwise. Splitwise là một cách đơn giản để theo dõi ai nợ ai tiền trong một nhóm người (ví dụ sau khi chia sẻ bữa trưa, mua hàng hoá hoặc hóa đơn). Đề bài cung cấp một ví dụ minh họa cho ứng dụng này:

Alice, Bob và Carol là bạn bè thích đi ăn cùng nhau. Bob đã trả tiền cho bữa trưa lần trước khi anh ta và Alice đi ăn, vì vậy Alice nợ Bob 10 đô la. Tương tự, Carol đã trả tiền khi cô ấy và Bob đi ăn, vì vậy Bob nợ Carol 10 đô la.

Giả sử Carol thiếu tiền mặt và vay 10 đô la từ Alice. Lúc này, thay vì mỗi người trả lại "khoản vay" của họ vào một thời điểm nào đó, họ có thể đồng ý rằng không ai nợ ai. Nói cách khác, khi có một chu trình nợ, chúng ta có thể loại bỏ nó khỏi sổ sách, làm cho mọi thứ đơn giản hơn và giảm số lần tiền mặt cần được trao đổi.

Chúng ta sẽ xây dựng một cách phi tập trung để theo dõi ai nợ ai bao nhiêu, để không phải dựa vào bên thứ ba đáng tin cậy. Nó sẽ hiệu quả: không tốn một lượng gas khổng lồ để lưu trữ dữ liệu này. Không có giá trị nào được chuyển "trên blockchain" bằng ứng dụng này; ether duy nhất liên quan đến việc thanh toán gas.

Vì nó nằm trên blockchain, khi Carol trả tiền cho bữa ăn của cô và Bob, cô có thể yêu cầu Bob gửi một hợp đồng nợ (IOU) (mà anh ta có thể làm bằng cách sử dụng ứng dụng phi tập trung của chúng ta), và cô có thể xác minh rằng anh ta đã làm như vậy. Việc lưu trữ công khai trên blockchain sẽ phục vụ như một nguồn thông tin duy nhất cho việc ai nợ ai. Sau đó, khi chu trình nợ được giải quyết như minh họa ở trên, Carol sẽ thấy rằng Bob không còn nợ cô tiền nữa.

Là một phần của đề bài này, chúng ta cũng sẽ xây dựng một 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 ứng dụng phi tập trung này.

2. Getting Started

2.1 Setup

* Cài nodejs
* Tải và giải nén code started
* Cd vào code
* Chạy lệnh: “npm install --save-dev hardhat” để cài đặt môi trường phát triển Ethereum Hardhat, mà bạn sẽ sử dụng để mô phỏng một nút Ethereum trên máy cục bộ của bạn.
* Chạy lệnh "npm install --save-dev @nomiclabs/hardhat-ethers ethers" để cài đặt một plugin Hardhat, được gọi là "hardhat-ethers", mà script "scripts/deploy.js" của bạn sẽ sử dụng để triển khai một nút Hardhat.

2.2. Compile, Deploy, Test

* Mở thư mục mã nguồn mẫu trong trình soạn thảo Visual Studio Code. Bạn chỉ sẽ chỉnh sửa tập tin "contracts/mycontract.sol" để định nghĩa hợp đồng Solidity của bạn và "web\_app/script.js" để xây dựng phần client. Xem qua các tập tin khác có thể giúp hiểu cách nút Hardhat và client web hoạt động. Có các vị trí được đánh dấu với các hàm để sửa đổi và bạn có thể thêm các hàm trợ giúp vào "web\_app/script.js". Vui lòng không sửa đổi bất kỳ mã nguồn hoặc cài đặt gói node bổ sung nào.
* Xem qua mã nguồn mẫu, tài liệu ethers.js và Solidity. Hãy suy nghĩ cẩn thận về thiết kế tổng quan của hệ thống trước khi viết mã. Dữ liệu nào nên được lưu trữ trên blockchan? Phần tính toán nào sẽ được thực hiện bởi hợp đồng so với phía client?
* Sau khi hoàn thành việc triển khai, chạy lệnh "npx hardhat node" để bắt đầu nút cục bộ. Nếu nút được khởi động đúng, bạn sẽ thấy trên terminal: "Started HTTP and WebSocket JSON-RPC server at https://localhost:8545", tiếp theo là 20 tài khoản với các khóa riêng tương ứng.
* Mở một tab hoặc cửa sổ terminal khác. Di chuyển đến thư mục mã nguồn mẫu. Chạy lệnh "npx hardhat run --network localhost scripts/deploy.js" để biên dịch và triển khai hợp đồng của bạn. Khi thành công, bạn sẽ nhìn thấy thông báo sau trên terminal: "Finished writing contract address: ...." Lưu địa chỉ này để sử dụng trong bước tiếp theo.
* Cập nhật địa chỉ hợp đồng và ABI trong tập tin "web\_app/script.js". ABI có thể được sao chép từ tệp tự động tạo "artifacts/contracts/mycontract.sol/Splitwise.json". Để cập nhật ABI đúng, hãy sao chép toàn bộ danh sách sau trường 'abi', bắt đầu từ dấu ngoặc vuông. Địa chỉ đã được lưu trong bước trước. Đảm bảo địa chỉ hợp đồng của bạn là một chuỗi.
* Mở tệp "web\_app/index.html” trong trình duyệt của bạn. Bạn có thể sử dụng giao diện người dùng để thao tác với hệ thống của mình hoặc chạy kiểm tra hợp lý của chúng tôi! Thêm thông tin trong phần 7.2 Development and Debugging

Lưu ý về các hệ điều hành: Tất cả các bước trên nên hoạt động trên hệ điều hành dựa trên Unix và Windows. Các lệnh chúng tôi yêu cầu bạn thực thi sẽ hoạt động trên terminal Unix tiêu chuẩn và Command Prompt của Windows.

3. Components:

Dự án có hai thành phần chính: một hợp đồng thông minh, được viết bằng Solidity và chạy trên blockchain, và một client chạy cục bộ trên trình duyệt web, theo dõi blockchain bằng thư viện ethers.js và có thể gọi các hàm trong hợp đồng thông minh. Để biết thêm thông tin về cách ethers.js hoạt động và cài đặt bài tập này, xem phần ghi âm mục 3 trên panopto.

3.1 Các hàm trong client

Vui lòng lưu ý, tất cả các hàm client mà chúng tôi yêu cầu bạn triển khai đều được cung cấp dưới dạng các hàm async trong mã nguồn mẫu. Điều này có nghĩa là chúng trả về một Promise theo mặc định. Hệ thống đánh điểm của chúng tôi sẽ giả định rằng mỗi hàm client trả về một Promise. Để biết thêm về async functions, Promises và await, xem tài liệu tham khảo ở cuối tài liệu này.

* getUsers(): Trả về một Promise cho danh sách các địa chỉ. Danh sách này sẽ là các địa chỉ của 'tất cả những người đã gửi hoặc nhận một IOU'. Bạn có thể tìm thấy điều này hữu ích như một hàm trợ giúp cho các hàm khác.
* getTotalOwed(user): Trả về một Promise cho tổng số tiền mà người dùng đang nợ.
* getLastActive(user): Trả về một Promise cho một dấu thời gian UNIX (giây kể từ ngày 1 tháng 1 năm 1970) của hoạt động ghi nhận cuối cùng của người dùng này (gửi một IOU hoặc được liệt kê là 'điền chỗ trống' trong một IOU). Trả về null nếu không tìm thấy hoạt động nào.
* add\_IOU(creditor, amount): Gửi một IOU tới hợp đồng, với người cho vay và số tiền đã cho vay. Xem ghi chú về việc giải quyết vòng lặp bên dưới.

3.2 Các hàm trong hợp đồng

* lookup(address debtor, address creditor) public view returns (uint32 ret): Trả về số tiền mà người nợ phải trả cho người cho vay.
* add\_IOU(address creditor, uint32 amount, ...): Thông báo cho hợp đồng rằng người gửi thông báo (msg.sender) hiện tại nợ thêm một số tiền (amount) đến người cho vay (creditor). Đây là hàm tích lũy: nếu bạn đã nợ tiền, hàm này sẽ cộng thêm vào số tiền đó. Số tiền phải là số dương. Bạn có thể thêm bất kỳ số lượng tham số bổ sung nào vào hàm này, miễn là hai tham số đầu tiên là người cho vay (creditor) và số tiền (amount). Xem ghi chú về việc giải quyết vòng lặp bên dưới.

Bạn có thể viết thêm các hàm trợ giúp cho client hoặc hợp đồng. Client có thể gọi các hàm của hợp đồng bằng cách sử dụng BlockchainSplitwise.functionName(arguments) và BlockchainSplitwise.connect(anotherSigner).functionName(arguments). Vui lòng xem tài liệu tại đây về cách tương tác với các hàm hợp đồng Solidity từ client bằng ethers.js. Hãy đảm bảo bạn biết sự khác biệt giữa hai phương pháp này. Hãy nhớ rằng các hàm client sẽ được viết bằng JavaScript và các hàm hợp đồng sẽ được viết bằng Solidity.

4. Giải quyết vòng lặp nợ nần

Hữu ích khi xem IOUs như là một đồ thị nợ. Tức là, giả sử mỗi người dùng là một nút, và mỗi cạnh có trọng số từ A đến B với trọng số X đại diện cho việc 'A nợ $X cho B'. Chúng ta sẽ viết thành A −!X B. Ứng dụng của chúng ta muốn 'giải quyết' bất kỳ chu trình nào trong đồ thị này bằng cách trừ đi giá trị nhỏ nhất trong chu trình đó từ mỗi bước trong chu trình (điều này sẽ làm cho ít nhất một bước trong chu trình có trọng số '0').

Ví dụ, nếu A −!15 B và B −!11 C, khi C thêm C −!16 A, các số dư thực tế sẽ được cập nhật để phản ánh rằng A −!4 B, B −!0 C và C −!5 A.

Ảnh có chứa xe đạp, bản phác thảo

Mô tả được tạo tự động

Tương tự, nếu C thêm C −!9 A, các số dư thực tế sẽ được cập nhật để phản ánh rằng A −!6 B, B −!2 C và C −!0 A.

Ảnh có chứa biểu đồ, vòng tròn, bản phác thảo, thiết kế

Mô tả được tạo tự động

Yêu cầu là nếu có bất kỳ chu trình tiềm năng nào được tạo ra khi bạn định thêm một IOU bằng khách hàng (add\_IOU), bạn phải 'giải quyết' ít nhất một trong số chúng. Bạn không cần lo lắng về các trường hợp phức tạp liên quan đến nhiều vòng lặp hoặc tối ưu hóa đường đi nào để chọn (giống như luồng cực đại) trong những trường hợp đó. Bạn có thể giả định rằng trước điều kiện cho cả hai hàm hợp đồng (add\_IOU và lookup), không có chu trình trong đồ thị. Cuối cùng, bạn cũng có thể giả định rằng bất kỳ chu trình nào được tìm thấy sẽ có kích thước nhỏ (ví dụ: ít hơn 10).

Chúng tôi cung cấp cho bạn một thuật toán tìm kiếm theo chiều rộng trong mã nguồn - để sử dụng nó, hãy truyền vào một nút bắt đầu và nút kết thúc, cùng với một hàm để lấy 'các nút láng giềng' của bất kỳ nút nào. Bạn có tự do không sử dụng cài đặt này nếu muốn.

Việc thực hiện việc giải quyết này một cách an toàn là trách nhiệm của bạn. Không thể cho phép một khách hàng xấu tính 'xóa sạch' nợ của họ sau khi đã đăng tải nó.

Giờ chúng ta có thể minh hoạ cách bạn có thể trả lại một IOU trong hệ thống này. Giả sử Alice mượn $10 từ Bob; bây giờ, cô ấy muốn trả lại Bob bằng tiền mặt. Khi Alice trả Bob $10 bằng tiền mặt, Bob sẽ thêm một IOU với số tiền $10 và người cho vay là Alice. Điều này sẽ tạo ra một chu trình: cụ thể là A −! 10 B và B −! 10 A. Theo yêu cầu giải quyết chu trình ở trên, điều này sẽ kết thúc với A −!0 B và B −!0 A.

5. Overall Requirements

Bạn có thể viết hợp đồng của mình bằng bất kỳ cách nào bạn muốn, miễn là nó có các hàm lookup và add\_IOU được chỉ định. Mục tiêu của bạn là viết một hợp đồng giảm thiểu lượng lưu trữ và tính toán được sử dụng bởi cả hai hàm hợp đồng. Điều này sẽ giảm thiểu chi phí gas.

Bạn có thể giả định rằng khối lượng giao dịch nhỏ đến mức khả thi để tìm kiếm toàn bộ blockchain trên khách hàng, nhưng bạn không nên giả định rằng chỉ có những người dùng trong ví của bạn là những người dùng duy nhất của hệ thống - nói cách khác, provider.listAccounts() không chứa tất cả người dùng có thể có của hệ thống.

6. Nộp mã của bạn:

Chúng tôi sẽ sử dụng Gradescope để nộp bài. Vui lòng chỉ nộp các tệp mycontract.sol và script.js DUY NHẤT! Bài nộp của bạn sẽ được đánh giá dựa trên việc câu trả lời câu truy vấn một cách chính xác, việc chi trả một lượng gas hợp lý và bảo mật hợp đồng.

7. Ghi chú:

Chúng tôi sẽ đăng một danh sách các lời giải thích và lời khuyên được cập nhật trên Ed. Vui lòng theo dõi bài viết đó để nhận thông tin mới nhất khi chúng tôi xử lý bất kỳ vấn đề nào liên quan đến nhiệm vụ.

7.1 Kiến trúc hệ thống:

* Bạn nên quyết định cấu trúc dữ liệu được lưu trữ trên blockchain trước. Hãy cân nhắc kỹ về thông tin mà bạn cần cung cấp cho khách hàng. Bạn không cần sử dụng bất kỳ cấu trúc dữ liệu phức tạp nào. Quyết định của bạn có thể làm cho việc triển khai trở nên khó khăn hơn, vì vậy bạn nên sẵn sàng để quay lại và thay đổi kiến trúc của mình.
* Chúng tôi chưa đề cập đến trường hợp xảy ra chu trình giữa hai người. Chúng tôi đề xuất thiết kế hệ thống của bạn sao cho đây không phải là một trường hợp đặc biệt - khi người mượn đã khi người mượn đã "trả lại" cho người cho vay, người cho vay đơn giản chỉ cần thử thêm một IOU theo hướng ngược lại, kích hoạt quá trình giải quyết chu trình và kết thúc với cả hai không nợ gì cho nhau. Chúng tôi cũng đề xuất bạn tránh bất kỳ khái niệm nợ "âm" nào, vì điều này có thể làm cho mọi thứ phức tạp hơn.
* Hãy nhớ rằng khi tối ưu hóa chi phí gas, các chức năng chạy trên khách hàng là miễn phí - chúng không tốn phí gì.
* Chúng tôi đề nghị sau khi thiết kế hệ thống của bạn, bạn nên bắt đầu bằng việc viết và gỡ lỗi cẩn thận hợp đồng.
* Bạn không cần một lượng mã lớn để hoàn thành nhiệm vụ. Giải pháp của chúng tôi gồm khoảng 40 dòng Solidity và khoảng 70 dòng JavaScript mới (không bao gồm ABI).

7.2 Phát triển và gỡ lỗi thực tế:

* Để gỡ lỗi mã phía khách hàng, hãy sử dụng console.log một cách rộng rãi. Bạn sẽ thấy kết quả của các cuộc gọi và số dòng chúng được gọi từ trong cửa sổ JavaScript console của trình duyệt.
* Có thể bỏ qua các cảnh báo về synchronous XMLHttpRequest.
* Solidity có một hàm rất hữu ích là require, cho phép bạn kiểm tra các tiền điều kiện.
* Trình tự giải mã ABI sẽ giải mã đầu vào của chức năng thành các chữ cái thường. Để thích ứng với điều này, mã khởi đầu cố gắng trả về tất cả các giá trị dưới dạng phiên bản chữ cái thường của chúng bằng cách sử dụng hàm toLowerCase(). Hãy nhớ giữ cho các giá trị chữ in hoa và chữ thường đúng. Mỗi chức năng nên hoạt động mà không phụ thuộc vào chữ hoa hay chữ thường của các ký tự chữ cái được truyền vào dưới dạng số hexa.
* Vui lòng lưu ý rằng chúng tôi đang sử dụng phiên bản Solidity 0.8.17, vì vậy không phụ thuộc vào tài liệu của các phiên bản cũ hơn vì chúng có thể hoạt động khác nhau.