پروژه‌ی دوم درس ارزهای رمزگذاری‌شده

استاد درس: دکتر بهرک

مهلت تحویل:

دستیار مربوطه: محمد محمودیان mahmoodian@ut.ac.ir

در این تمرین کامپیوتری هدف این است که شما با زبان برنامه‌نویسی Solidity و همچنین یک API[1] به نام web3.js آشنا شوید به طوریکه بتوانید یک نرم‌افزار غیرمتمرکز[2] ساده را پیاده سازی کنید. در نرم‌افزارهای غیرمتمرکز نیز، همانند سایر نرم-افزارها، لازم است یک قسمت سمت کاربر[3] و یک قسمت سمت سرور[4] پیاده سازی شود، که در این تمرین شما با پیاده سازی هر دو سمت آشنا می‌شوید.

1. دفترچه‌حساب زنجیره‌ی بلوکی[5]

هدف این است که یک سیستم غیرمتمرکز طراحی شود که بتواند بدهی و طلب افراد را ثبت کند؛ درواقع نمونه‌ی غیرمتمرکز از یک دفترچه حساب. برای شفاف شدن بیشتر، یک مثال از نحوه‌ی کار این سیستم در ادامه آورده می‌شود.

فرض کنید که علیرضا، الهام و شبنم سه دوست هستند که معمولا با یکدیگر بیرون می‌روند و غذا می‌خورند. آخرین بار که علیرضا و الهام با یکدیگر ناهار خورده‌اند، هزینه آن را الهام پرداخت کرده است. بنابراین علیرضا 30 هزار تومان به الهام بدهی دارد. همچنین یک بار که الهام و شبنم برای شام بیرون رفته‌اند شبنم هزینه آن را پرداخت کرده است، پس الهام 30 هزار تومان به شبنم بدهکار است.

حال فرض کنید که شبنم به پول نیاز دارد و مبلغ 30 هزار تومان از علیرضا قرض بگیرد؛ در این لحظه هر سه نفر می‌توانند توافق کنند که هیچ یک از آن‌ها به دیگری بدهی ندارد و از تبادل پول خودداری کنند. درواقع هر زمانی که یک دور[6] از بدهی‌ها وجود داشته باشد می توان آن دور را حذف کرد و از جابجا شدن پول جلوگیری کرد.

ما قرار است که یک سیستم غیرمتمرکز طراحی و پیاده سازی کنیم که مشخص کند هر شخص به چه کسی و چه مبلغی بدهکار است، بدون اینکه نیاز باشد یک سازمان سومی[7] در میان باشد. به صورتی‌که مثلا وقتی الهام هزینه ناهار علیرضا را پرداخت می‌کند از او بخواهد که در شبکه زنجیره‌ی بلوک بدهکار بودنش را تصدیق کند. نحوه‌ی تصدیق کردن بدین شکل است که فرد بدهکار یک تابع را صدا می‌زند و در آن شخص طلبکار و مبلغ بدهی را مشخص می‌کند؛ در کدِ برنامه این تصدیق‌نامه(تاییدیه) را

---

[1] Application Protocol Interface
[2] Decentralized Application(DApp)
[3] Client side
[4] Server side
[5] Blockchain Splitwise
[6] Cycle
[7] Third party

IOU[8] می‌نامیم. فضای موجود و عمومی و همچنین قابل دسترسِ شبکه‌ی زنجیره‌ی بلوک می‌تواند به عنوان محل ذخیره سازی و یا به عبارتی سمت سرور برای لیست بدهی‌های هر شخص استفاده شود. برای راحتی کار با این سرور لازم است که یک محیط کاربری نیز طراحی شود که جلوتر در مورد آن صحبت خواهیم کرد. ☺

2. آماده‌سازی جهت شروع

1) نصب نرم‌افزارهای پیشنیاز: لازم است که بر روی سیستم خود Node.js را نصب کنید. می توانید از این لینک استفاده کنید. لازم است که علاوه بر Node.js ، npm نیز بر روی سیستم شما نصب شود. به کمک npm می‌توانید سایر پکیج‌های مورد نیاز را نصب کنید. نسخه‌ی ویندوز Node.js همراه خود npm را نیز دارد، اما در سیستم‌عامل‌های دیگر ممکن است لازم باشد آن را جداگانه نصب کنید.
2) نصب شبیه‌ساز شبکه اتریوم و اجرای آن: برای اینکار در ترمینال خود از دستور npm install –g ganache-cli استفاده کنید. با این کار یک سرور به نام ganache بر روی سیستم شما نصب می شود که با اجرا شدن آن تعدادی اکانت اتریوم(پیشفرض 10 عدد) با موجودی های تعیین شده(100 اتر) ایجاد می شود که از آن‌ها می‌توانید برای انجام این تمرین کامپیوتری استفاده کنید. می‌توان از طریق پروتکل http به این سرور وصل شد، آدرس پیشفرض آن localhost یا همان 127.0.0.1 است و پورت پیشفرض آن نیز 8545 است. برای اجرا شدن این سرور کافیست دستور ganache-cli را در ترمینال خود وارد کنید. برای متوقف کردن سرور نیز می‌توانید از Ctrl+C استفاده کنید.
3) از صفحه‌ی درس پوشه ItCProject2Starter را دانلود کنید.
4) در جستجوگر وب خود صفحه‌ی مقابل را باز کنید. https://remix.ethereum.org . در قسمت 'Run' محیط اجرایی[9] را بر روی 'Web3 Provider' تنظیم کنید. و آدرس آن را نیز بر روی http://localhost:8545 که آدرس پیشفرض نیز هست، تنظیم کنید. محیطی که در آن هستید جایی است که لازم است شما قرارداد هوشمند[10] خود را بنویسید و آن را کامپایل کنید. و در شبکه‌ی شبیه‌سازی شده مستقر(ثبت)[11] کنید.
5) در پوشه ای که از سایت درس دانلود کرده اید یک فایل به نام index.html وجود دارد. آن را در مرورگر خود باز کنید. باید صفحه ای را تحت عنوان 'Blockchain Splitwise' بتوانید ببینید. توصیه می‌شود که قسمت javascript console مرورگر خود را نیز باز کنید و خطاهای احتمالی را مشاهده کنید. (برای نحوه‌ی باز کردن این قسمت می-توانید از این لینک کمک بگیرید) در صورتیکه همه چیز به خوبی پیش رفته باشد. شما نباید خطایی را مشاهده کنید. البته احتمالا هشدارهایی را می‌بینید که می‌توانید آن‌ها را نادیده بگیرید.
6) فایل‌های درون پوشه‌ی دانلود شده را در محیط برنامه نویسی مورد علاقه خود باز کنید. (مانند Atom, Sublime,..) شما برای کامل شدن سمت کاربر لازم است که فایل script.js را کامل کنید. توصیه می‌شود که سایر فایل های موجود را نیز بررسی کنید.
7) قبل از شروع توصیه می‌شود که راجع به web3.js و برنامه نویسی solidity اطلاعات بیشتری کسب کنید. همچنین فکر کردن راجع به نحوه‌ی کار سیستم کلی و همچنین نحوه ی ذخیره سازی اطلاعات و ساختارهای داده‌ای که قرار

[8] I Owe You
[9] Environment
[10] Smart contract
[11] Deploy

است استفاده کنید، قبل از کدزنی می‌تواند مفید باشد. درحال حاضر یک نسخه‌ی تحت توسعه از web3.js وجود دارد(نسخه 1.0) که می‌توانید راهنمای آن را از اینجا ببینید، اما در این تمرین کامپیوتری از نسخه‌های قدیمی‌تر(0.x) استفاده می‌شود که راهنمای آن را در این لینک می‌توانید ببینید.

8) در قسمت بعد کارهایی که قرار است انجام دهید با جزئیات بیشتری توضیح داده می‌شود، بعد از اینکه قرارداد هوشمند خود را نوشتید و آن را بر روی شبکه ثبت کردید لازم است که آدرس[12] آن و همچنین چکیده[13] آن را در فایل script.js به‌روزرسانی کنید. چکیده را می‌توانید از قسمت 'Compile' و آدرس را نیز می‌توانید از قسمت 'Deployed Contracts' در بخش 'Run' در صفحه Remix بیابید و کپی کنید.

3. نیازمندی‌ها

همانطور که قبلا توضیح داده شد، این تمرین کامپیوتری از دو بخش مهم تشکیل می‌شود: قرارداد هوشمند که به زبان solidity نوشته می‌شود و در شبکه‌ی آنلاین زنجیره‌ی بلوک ثبت می‌شود و قسمت سرور شما را تشکیل می‌دهد و همچنین یک محیط کاربری که به زبان JavaScript نوشته می‌شود و درمرورگر شما اجرا می‌شود و از طریق web3.js به سرور شما وصل می‌شود و اطلاعاتی را که میخواهید نمایش می‌دهد.

1) توابع سمت کاربر

I. getUsers() : این تابع یک لیست از آدرس‌های شبکه‌ی اتریوم را برمی‌گرداند. شما می‌توانید آدرس تمام افراد موجود در شبکه را برگردانید و یا اینکه فقط افرادی را برگردانید که تا به حال فعالیت مالی(طلب یا بدهی) داشته‌اند. این تابع می‌تواند به عنوان تابع کمکی در دیگر توابع مورد استفاده قرار بگیرد.

II. getTotalOwed(user) : این تابع آدرس یک کاربر را می‌گیرد و کل مبلغی را که آن کاربر بدهکار است برمی‌گرداند.

III. getLastActive(user) : این تابع زمان آخرین فعالیت کاربر را برمی‌گرداند. منظور از آخرین فعالیت، فرستادن IOU و یا ذکر شدن به عنوان بستانکار در یک IOU توسط یک کاربر دیگر است. فرمت زمانی باید به صورت ثانیه‌های گذشته از مبدأ تاریخ (1 ژانویه 1970) باشد.

IV. add_IOU(creditor, amount) : این تابع به عنوان تاییدیه فرستاده می‌شود که نشان می‌دهد شخصی که آن را صدا می‌زند مبلغی معادل amount را به فرد creditor بدهکار است. این تابع چیزی را برنمی‌گرداند. در این تابع لازم است که مشکل وجود حلقه نیز برطرف شود که جلوتر در مورد آن توضیح خواهیم داد.

2) توابع سمت سرور

یک فایل به نام BlockchainSplitwise.sol در پوشه‌ی starter وجود دارد که لازم است همه‌ی آن را خودتان بنویسید و شامل توابع زیر باشد.

I. lookup(address debtor, address creditor) public view returns (unit32 ret) : این تابع مبلغی را که فرد debtor به شخص creditor بدهکار است، برمی‌گرداند. برای این کار کافی است که در بدنه‌ی

[12] Contract address
[13] ABI

تابع متغیر ret را مقداردهی کنید و نیازی به نوشتن return در انتهای تابع نیست. کلمه view یعنی اینکه این تابع قرار نیست متغیرهای موجود در شبکه را تغییر دهد، بلکه صرفا قرار است مقادیر این متغیرها را بخواند.

II. add_IOU(address creditor, uint32 amount, ...) : این تابع تاییدیه نهایی در سرور را انجام می‌دهد و نشان می‌دهد که msg.sender مبلغ amount را به فرد creditor بدهکار است. amount لازم است که عدد مثبتی باشد. شما می‌توانید بسته به نیازتان آرگومان های دیگری برای این تابع تعریف کنید.

شما می‌توانید بسته به نیازتان توابع دلخواه دیگری را در هریک از دوسمت کاربر و سرور تعریف کنید و از آن‌ها استفاده کنید. در سمت کاربر می‌توانید توابع سمت سرور را به صورت BlockchainSplitwise.functionname(arguments) صدا بزنید.

4. حل مشکل وجود حلقه‌ی بدهی‌ها

برای تحلیل راحت‌تر، از مدل گرافِ جهت‌دارِ وزندار برای مسئله‌ی خود استفاده می‌کنیم، بدین صورت که هر یک از کاربرها یک گره محسوب می‌شوند و هریک از یال‌ها وزنی دارد که مقدار وزن آن نشان‌دهنده‌ی مبلغ بدهی است. جهت یال ها از بدهکار به سمت بستانکار است. مثلا وقتی که شخص A مبلغ X را به شخص B بدهکار است آنگاه: $A \overset{X}{\rightarrow} B$. نرم‌افزار ما لازم است به‌گونه‌ای کار کند که اگر حلقه‌ای ایجاد شد آن را از بین ببرد برای این‌کار لازم است که کوچکترین یال در حلقه پیدا شود و مقدار آن از تک‌تک یال‌ها کم شود به صورتیکه یکی از یال‌ها به ارزش صفر برسد.

به عنوان مثال اگر $A \overset{15}{\rightarrow} B$ و $B \overset{11}{\rightarrow} C$ آنگاه اگر C بخواهد تراکنش $C \overset{16}{\rightarrow} A$ را اضافه کند لازم است که نرم‌افزار ما گراف را به صورت زیر تغییر دهد.

به عنوان یک مثال دیگر اگر C بخواهد تراکنش $C \overset{9}{\rightarrow} A$ را اضافه کند آنگاه گراف جدید به صورت زیر خواهد بود.

لازمه‌ی انجام این‌کار این است که شما در تابع add_IOU در قسمت کاربر قبل از فرستادن دستور به سمت سرور چک کنید که آیا با اضافه شدن این تراکنشِ جدید حلقه‌ای به وجود می آید یا نه و درصورت وجود، حداقل یکی از آن‌ها را حذف کنید. نگران موارد پیچیده نباشید و یا اینکه لازم نیست به دنبال پیدا کردن راه‌های بهینه برای حذف حلقه باشید. بنابراین لازم است در توابع سمت سرور این فرض را درنظر بگیرید که هیچ حلقه‌ای وجود ندارد و قسمت حذف حلقه در سمت کاربر انجام می‌گیرید.

در کدهای سمت کاربر یک تابع کمکی به نام doBFS(start, end, getNeigbors) برای شما قرار داده شده است که الگوریتم bread-first search را انجام می‌دهد و خروجی آن لیستی از گره‌ها ایست که گره start را به گره end وصل می-کنند، البته درصورتیکه مسیری بین این دو گره وجود داشته باشد. دقت شود که آرگومان سوم، خود یک تابع است که یک گره را می‌گیرد و همسایه‌های آن را به صورت یک لیست برمی‌گرداند و لازم است خودتان آن را بنویسید. در صورتیکه مسیری بین دوگره وجود نداشته باشد، خروجی تابع یک لیست خالی خواهد بود. استفاده از این تابع اختیاری است و شما می‌توانید از هر الگوریتم دیگری برای پیدا کردن مسیر استفاده کنید.

آخرین چیزی که به نظر می‌رسد باید انجام دهیم، پرداخت بدهی است به این صورت که فرض کنید آخرین بار علیرضا مبلغ 40 هزار تومان به شبنم بدهکار است حال اگر پول خود را به شبنم پرداخت کند لازم است که شبنم در شبکه یک تاییدیه بفرستد که مبلغ 40 هزار تومان به علیرضا بدهکار است با این کار یک حلقه‌ی کوچک به وجود می‌آید که انتظار می‌رود نرم‌افزار ما آن را تشخیص دهد و حذف کند و نتیجه‌ی آن این باشد که هیچ یک از علیرضا و شبنم به دیگری بدهکار نیست. در واقع برای این حالت خاص لازم نیست که شما کار خاصی را انجام دهید، درصورتیکه قسمت قبل را به خوبی پیاده‌سازی کرده باشید، این حالت نیز خودبه‌خود انجام می‌شود.

5. توصیه های نهایی

1) دست شما باز است که هر طور که می‌خواهید قراداد هوشمند خود را بنویسید فقط لازم است که توابع خواسته شده را حتما پیاده‌سازی کنید. همچنین سعی کنید قرارداد هوشمند شما کمترین میزان حافظه و کمترین میزان انجام محاسبات را به خود اختصاص دهد، چرا که اینکار باعث می‌شود که استفاده از قرارداد شما ارزان‌تر تمام شود.
2) از بدهی منفی حتما خودداری کنید زیرا این‌کار پیچیدگی را افزایش می‌دهد.
3) توصیه می‌شود که ابتدا قرارداد هوشمند خود را کامل کنید، سپس سراغ بخش کاربری بروید. در سایت Remix می-توانید تمام توابع قرارداد را صدا بزنید و آن‌ها را تست کنید. (از طریق پنل سمت راست و پایین صفحه) و همچنین می‌توانید از قسمت 'Accounts' بین آدرس‌های مختلف جابجا شوید. برای کپی کردن هریک از آدرس‌هایی که فکر می‌کنید لازم دارید، می‌توانید از آیکن کنار آن‌ها استفاده کنید. همچنین از طریق قسمت 'debug' می‌توانید تراکنش‌های انجام شده را دیباگ کنید و با جزئیات بیشتری اتفاقات درون آن را ببینید.(توجه کنید که برای استفاده از این قسمت لازم است که از قسمت Environment اتصال خود به سرور محلی اتریوم را قطع کنید و به JavaScript VM وصل شوید)
4) هر بار که سرور شبیه‌ساز شبکه اجرا می‌شود اکانت‌های متفاوتی ساخته می‌شود، در صورتیکه علاقه دارید هربار اکانت-های یکسانی ساخته شود می‌توانید از دستور ganache-cli -d استفاده کنید.
5) برای انجام این تمرین کامپیوتری به حجم زیادی از کدزنی نیاز ندارید. کد نوشته شده‌ی ما شامل حدود 40 خط در سمت سرور و 70 خط در سمت کاربر است.

6) در قسمت سمت کاربر هنگامی‌که شما از تابع lookup استفاده می‌کنید مقدار بازگشتی آن به فرمت BigNumber است که به فرم استاندارد با سه پارامتر c, e, s است، برای تبدیل آن به عدد معمولی می‌توانید از دستور bn.toNumber() و یا از عملگر + قبل از bn استفاده کنید که منظور از bn مقدار بازگشتی از تابع lookup است.
7) برای دیباگ کردن سمت کاربر می‌توانید از تابع آماده‌ی log و یا از تابع console.log استفاده کنید که تابع اول خروجی را در صفحه‌ی اصلی سایت نشان می‌دهد و تابع دوم خروجی را در قسمت console مرورگرتان نشان می-دهد.
8) در صورت مشاهده‌ی خطایی مبنی بر عدم اتصال به localhost:8545 ، از درحال اجرا بودن شبیه‌ساز شبکه اتریوم (ganache-cli) مطمئن شوید.
9) در زبان solidity یک تابع کارآمد به نام require وجود دارد که می‌تواند برای چک‌کردن‌های مربوط به قسمت‌های امنیتی از آن استفاده کنید.

6. بخش امتیازی

در بخش قبل برای کامپایل کردن و ثبت کردن قرارداد بر روی شبکه‌ی بلاک‌چین از سایت remix استفاده کردیم، در این بخش قرار است ابزاری را معرفی کنیم که این کار را هوشمندانه‌تر و کاربردی‌تر انجام دهد. پکیجی به نام truffle. برای نصب آن کافیست که در ترمینال خود دستور روبرو را وارد کنید. npm install –g truffle . سه تا از دستورهای ترافل که در این بخش از آن استفاده می‌کنیم، عبارتند از:

- truffle compile
- truffle migrate
- truffle test

نحوه‌ی کار ترافل بدین شکل است که لازم است سه پوشه با نام‌های contracts، migrations و test در پوشه‌ی اصلی که پروژه ما در آن است موجود باشد. در پوشه اول(contracts) کدهای solidity قرار می‌گیرد، قراردادهای اصلی که لازم است کامپایل و ثبت شوند. در پوشه دوم و سوم فایل‌های به فرمت js. قرار می‌گیرند که اصطلاحا اسکریپت نام دارند و به زبان Node.js هستند که شباهت زیادی به JavaScript دارد. فایل‌های پوشه‌ی migrations لازم است به صورتی باشند که اسم آن‌ها با عدد شروع شود.

با اجرای دستور اول، قراردادهای موجود در پوشه اول کامپایل می‌شوند و یک پوشه به نام build ساخته می‌شود و چکیده‌ی قراردادها به فرمت json. در آن ثبت می‌شود، می‌توانید abi هر قرارداد را آن‌جا ببینید و درصورت لزوم آن را کپی کنید.

با اجرای دستور دوم، اسکریپت‌های موجود در پوشه دوم به ترتیب اسم توسط Node.js کامپایل و اجرا می‌شوند، این اسکریپت‌ها به نحوی نوشته می‌شوند که با اجرا شدن آن‌ها، قراردادهای اصلی(موجود در پوشه اول) کامپایل و درصورت درستی در شبکه ثبت شوند. درواقع هدف اسکریپت‌های موجود در پوشه دوم انجام کاری است که ما تا الان با remix انجام می‌دادیم.

اجرای دستور سوم باعث می‌شود که تمام فایل های موجود درپوشه تست کامپایل و اجرا شوند. هدف از فایل‌های موجود در این پوشه این است که قراردادهای اصلی ما را در شبکه ثبت کند و سپس توابع آن‌ها را صدا بزند و خروجی های آن‌ها را نمایش دهد، در واقع با این کار به راحتی می‌توان چک کرد که آیا توابعی که ما در قراردادمان نوشته‌ایم به درستی کار می‌کنند یا نه.

چند نکته.

I. لازم است در هنگام استفاده از truffle سرور ganache-cli درحال اجرا باشد.

II. برای آشنایی بیشتر با truffle می‌توانید از دستور truffle --help استفاده کنید و با سایر قابلیت‌های آن آشنا شوید از جمله می‌توان به truffle console اشاره کرد که برای استفاده‌ی developers مناسب است.

III. اگر در یک پوشه خالی دستور truffle unbox را اجرا کنید یک نمونه از یک پروژه انجام شده را همراه با سه پوشه اصلی و فایل های درون آن می‌توانید مشاهده کنید. حتما توصیه می‌شود که از این فایل‌ها کمک بگیرید.

IV. با انجام نکته سوم خواهید دید که در پوشه migrations دوفایل با نام‌های 1_initial_migration.js و 2_deploy_contracts.js وجود دارد فایل اول را می‌توانید دقیقا به همان صورت استفاده کنید اما لازم است که در فایل دوم تغییراتی ایجاد کنید که با قرارداد هوشمند شما سازگار باشد. همچنین برای دیدن آدرس قرارداد ثبت شده (که در مراحل قبل آدرس آن را از remix کپی می‌کردید و به فایل script.js انتقال می‌دادید) می‌توانید از قطعه کد زیر کمک بگیرید.

```
deployer.deploy(BlockchainSplitwise).then(function() {

    console.log("BlockchainSplitwise ADDRESS: " + BlockchainSplitwise.address);
    console.log("\n\n---------------\nDEPLOYMENT DONE!\n---------------\n\n");
});
```

V. با انجام قسمت سوم همچنین خواهید دید که در پوشه‌ی contracts سه فایل موجود است، لازم است که فایل Migrations.sol را به همان شکل به پوشه‌ی متناظر در پروژه‌ی خود انتقال دهید.

VI. در آخر نیز اگر بعداز انجام قسمت سوم محتویات پوشه test را ببینید. دو فایل وجود دارد که شما تنها لازم است فایلی همانند فایل metacoin.js در پروژه خود بسازید. توجه شود که در این اسکریپت از مفهومی به نام promise که یکی از مفاهیم کدنویسی به زبان nodejs است، استفاده‌ی زیادی دارد، همچنین از دستور ()it استفاده می‌شود، توصیه می‌شود برای درک بهتر کدها، مقداری با این مفاهیم آشنا شوید.

در این قسمت از شما خواسته می‌شود که سه پوشه اصلی را بسازید و فایل‌های لازم را در آن بنویسید. در مورد فایل موجود در پوشه تست، لازم است که یک فایل به نام test.js بسازید و آن را کامل کنید در این فایل لازم است که کلیه توابعی که در قرارداد خود آورده اید تست شوند و از صحت اجرای آن‌ها اطمینان حاصل شود. میزان پیچیدگی تست ها به خودتان بستگی دارد، اما انتظار می‌رود که تست‌ها هوشمندانه و قابل اطمینان باشند. از آنجاییکه این بخش به صورت امتیازی است، نمره‌ی اختصاص داده شده به آن به صورت نسبی و در مقایسه با عملکرد بقیه دانشجویان خواهد بود.

7. بارگزاری فایل ها

کلیه فایل‌ها و پوشه‌های خود را در قالب یک فایل فشرده به صورت StudentID1-StudentID2.zip آپلود کنید، یکی از افراد گروه آپلود کند، کافیست.