

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده مهندسی برق و کامپیوتر
گروه نرم افزار



کاپو، پلتفرم ساخت و انتقال توکن‌های داده‌ای

پایان‌نامه برای دریافت درجهٔ کارشناسی در رشتهٔ مهندسی کامپیوتر
گرایش نرم افزار

امین بشیری

استاد راهنما

دکتر احسان خامس‌پناه

خرداد ۱۴۰۱

سُبْحَانَ رَبِّ الْجَمَلِ



دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده مهندسی برق و کامپیوتر
گروه نرم‌افزار



کاپو، پلتفرم ساخت و انتقال توکن‌های داده‌ای

پایان‌نامه برای دریافت درجهٔ کارشناسی در رشتهٔ مهندسی کامپیوتر
گرایش نرم‌افزار

امین بشیری

استاد راهنما

دکتر احسان خامس‌پناه

خرداد ۱۴۰۱



دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده مهندسی برق و کامپیوتر



گواهی دفاع از پایان‌نامه کارشناسی

هیأت داوران پایان‌نامه کارشناسی آقای / خانم امین بشیری به شماره دانشجویی ۸۱۰۱۹۶۴۲۵ در رشته مهندسی کامپیوتر - گرایش نرم‌افزار را در تاریخ با عنوان «کاپو، پلتفرم ساخت و انتقال توکن‌های داده‌ای»
به عدد به حروف

با نمره نهایی

ارزیابی کرد.

و درجه

ردیف.	مشخصات هیأت داوران	نام و نام خانوادگی	مرتبه دانشگاهی	دانشگاه یا مؤسسه	امضا
۱	استاد راهنما	دکتر احسان خامس‌پناه	استاد	دانشگاه تهران	<input type="text"/>
۲	استاد داور داخلی	دکتر داور داخلی	دانشیار	دانشگاه تهران	<input type="text"/>
۳	استاد مدعو	دکتر داور خارجی	دانشیار	دانشگاه داور خارجی	<input type="text"/>
۴	نماینده تحصیلات تکمیلی دانشکده	دکتر نماینده	دانشیار	دانشگاه تهران	<input type="text"/>

نام و نام خانوادگی معاون آموزشی و تحصیلات

تکمیلی پردیس دانشکده‌های فنی:

تاریخ و امضا:

نام و نام خانوادگی معاون تحصیلات تکمیلی و

پژوهشی دانشکده / گروه:

تاریخ و امضا:

تعهدنامه اصالت اثر

با اسمه تعالیٰ

اینجانب امین بشیری تأیید می‌کنم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب است و به دستاوردهای پژوهشی دیگران که در این نوشته از آن‌ها استفاده شده است مطابق مقررات ارجاع گردیده است.
این پایان‌نامه قبل‌اً برای احراز هیچ مدرک هم‌سطح یا بالاتری ارائه نشده است.

نام و نام خانوادگی دانشجو: امین بشیری

تاریخ و امضای دانشجو:

چکیده

اینترنت غیر مرکزی^۱ یا Web3 به عنوان مهمترین تغییر بعد از به وجود آمدن اینترنت^۲ در نظر گرفته می‌شود. با به وجود آمدن رمزارزها^۳ و الگوریتم‌های اجماع^۴ کارآمد، و فراهم شدن زمینه اجرای برنامه‌ها و انجام تراکنش‌های مالی به صورت غیر مرکزی، عصر اینترنت غیر مرکز فرا رسیده است.

در این میان یکی از اصلی‌ترین مزایای برنامه‌های غیر مرکز مالکیت واقعی دارایی است. به این معنی که یک شخص یا یک نهاد نمی‌تواند دارایی‌های کس دیگری را مسدود یا مصادره کند. از طرفی مالکیت‌های معنوی نیز به صورت واضح و شفاف می‌توانند مشخص شوند. برای مثال یک هنرمند به وضوح صاحب اثرش است و هر چند که دیگر افراد می‌توانند اثر او را کپی کنند اما همیشه مشخص است که صاحب اصلی اثر کیست.

به این ترتیب توکن‌های تعویض‌ناپذیر^۵ با قابلیت‌های مالکیت بسیار زیادی که فراهم می‌کنند مورد استقبال فراوان مردم واقع شدند. قابلیت‌هایی مانند ساخت، نگهداری، فروش و انتقال فوق العاده راحت و سریع نیز در این سرعت فرآگیری تاثیر بسزایی داشته‌اند. برای بازاری به این تازگی و وسعت، تکنولوژی‌ها، استانداردها و پلتفرم‌های زیادی ساخته شده‌اند و همچنان نیز در حال توسعه هستند.

در این پروژه سعی بر ساخت پلتفرمی داریم که هر شخص یا شرکتی بتواند با عضویت در آن، به آسان‌ترین روش ممکن توکن‌های تعویض‌ناپذیر بسازد و به دیگران انتقال دهد. کاربردهای این پلتفرم ساده بی‌شمار است. دارایی‌هایی مانند بلیت سینما، ژتون‌های غذا، وقت گرفتن از دکتر، قراردادها و ... همه می‌توانند به آسانی در این پلتفرم به توکن تعویض‌ناپذیر تبدیل شوند، به دیگران انتقال یابند و در بازار خرید و فروش شوند.

گرچه می‌توان کاربردهای فراوانی را برای این پلتفرم در نظر داشت اما همچنان یکی از اهداف انجام این پروژه آشنایی با نحوه ساخت، تست و بارگذاری یک قرارداد هوشمند، ساخت واسطه کاربری، اتصال آن به قرارداد ERC1155 و همچنین شناخت استانداردهای معروف قراردادهای توکن‌های تعویض‌ناپذیر مانند ERC721 و ERC770 است.

لازم به ذکر است که تمامی کدهای کاپو به صورت متن باز^۶ در گیت‌هاب^۷ قابل دسترس برای عموم هستند.

¹Decentralized Web

²Word Wide Web

³CryptoCurrencies

⁴Consensus Algorithms

⁵Dapps

⁶Non-fungible tokens

⁷Open Source

⁸<https://github.com/bshramin/cappu>

واژگان کلیدی کاپو، توکن، داده، سالیدیتی، قرارداد هوشمند، توکن غیرقابل تعویض، ترافل

فهرست مطالب

۱	فصل ۱: مقدمه و بیان مسئله
۱	۱ مقدمه
۲	۲ شرح مسئله و روش انجام آن
۳	۳ اهداف کلی تحقیق
۳	۳.۱ گسترش کاربردهای توکن‌های تعویض‌ناپذیر
۳	۳.۲ ساخت روند توسعه ^۹ قرارداد هوشمند
۴	۴.۱ ۴ پادگیری
۴	۴.۲ ساختار پایان‌نامه
۵	فصل ۲: مفاهیم اولیه و پیش‌زمینه
۵	۵.۱ دلایل و برتری‌های متن‌باز بودن قراردادهای هوشمند
۷	۷.۱ آشنایی با مفهوم توکن تعویض‌ناپذیر
۸	۸.۱ کاربردها، حال و آینده
۹	۹.۱ قراردادهای هوشمند و استانداردسازی
۹	۹.۲ استاندارد ERC20
۱۰	۱۰.۲ استاندارد ERC721
۱۰	۱۰.۲ استاندارد ERC1155
۱۳	فصل ۳: آشنایی با ابزارهای توسعه
۱۳	۱۳.۱ ابزارهای ساده

⁹Development Flow

۱۴	۲.۳ کیف پول متمامسک
۱۴	۳.۳ چارچوب‌ها و کتابخانه‌ها
۱۵	۱.۳.۳ چارچوب ترافل
۱۶	۲.۳.۳ کتابخانه اپن‌زپلین
۱۷	۳.۳.۳ کتابخانه Web3JS
۱۸	۴.۳ شبکه محلی برای توسعه
۲۳	فصل ۴: پیاده‌سازی
۲۳	۱.۴ نوشتن کد قرارداد
۲۳	۱.۱.۴ نیازمندی‌های قرارداد هوشمند
۲۴	۲.۱.۴ ارثبری
۲۴	۳.۱.۴ توجه به هزینه تراکنش و نوع توابع
۲۵	۴.۱.۴ جزئیات فنی پیاده‌سازی
۲۸	۲.۴ نوشتن و اجرای تست‌ها
۳۰	۳.۴ بارگذاری قرارداد روی شبکه تستی راپستن ^{۱۰}
۳۰	۱.۳.۴ یافتن آدرس یکی از نودهای شبکه برای ارسال تراکنش بارگذاری قرارداد به آن
۳۰	۲.۳.۴ اضافه شدن اطلاعات شبکه مورد نظر به تنظیمات ترافل
۳۱	۳.۳.۴ آماده شدن نمونیکز ^{۱۱}
۳۱	۴.۳.۴ استفاده از کیف پول ایجاد شده در تنظیمات ترافل
۳۲	۵.۳.۴ نصب کیف‌پول hdwallet
۳۲	۶.۳.۴ انتخاب شبکه اضافه شده
۳۳	۷.۳.۴ بررسی آدرس کیف‌پول و موجودی آن
۳۴	۸.۳.۴ بارگذاری قرارداد هوشمند روی شبکه بلاکچین
۳۴	۹.۳.۴ اطمینان از صحت بارگذاری قرارداد هوشمند
۳۴	۴.۴ توسعه واسط کاربری، اتصال به قرارداد هوشمند و فرآیند بارگذاری

¹⁰Ropsten¹¹Mnemonics

۳۶	۵.۴ داکرایز شدن، پایپلاین‌ها و گیت
۳۶	۱.۵.۴ داکرایز شدن تست‌های قرارداد هوشمند
۳۷	۲.۵.۴ اجرای خودکار تست‌های قرارداد
۳۷	۳.۵.۴ بارگذاری خودکار واسط کاربری
۳۹	فصل ۵: دستآوردها، پیشنهادها، محدودیت‌ها
۳۹	۱.۵ دستآوردها
۳۹	۱.۱.۵ پلتفرم ایجاد شده
۳۹	۲.۱.۵ ساخت محیط توسعه سریع و خودکار
۴۰	۳.۱.۵ ورود به جامعه توسعه‌دهندگان ^{۱۲}
۴۱	۴.۱.۵ یادگیری
۴۱	۲.۵ پیشنهادها
۴۱	۱.۲.۵ استفاده از استانداردها
۴۲	۲.۲.۵ استفاده از ERC1155 به جای ERC721
۴۲	۳.۲.۵ ساخت محیط توسعه از شروع کار
۴۲	۳.۰.۵ محدودیت‌ها
۴۲	۱.۳.۵ استفاده از دریزل ^{۱۳}
۴۳	۲.۳.۵ ورژن‌های مختلف ابزارها
۴۳	۳.۳.۵ هزینه تراکنش‌های شبکه اصلی اتریوم
۴۳	۴.۳.۵ عدم وجود راهنمای مستندات کافی

اول

مراجع

¹²Community¹³Drizzle

فصل ۱

مقدمه و بیان مسئله

۱.۱ مقدمه

در یک دهه اخیر محبوبیت رمزارزها در میان مردم به شدت افزایش یافته است. رمزارزها توکن‌هایی تعویض‌پذیر هستند، به این معنی که تفاوتی میان دو توکن یک رمزارز وجود ندارد، مانند پول فیزیکی^۱ که ارزش یک هزار تومانی با یک هزار تومانی دیگر تفاوتی ندارد.

اما در دنیای واقعی تنها مالکیت پول نیست که اهمیت دارد. یک فرد می‌تواند خودرو، خانه، بلیت هوایپیما و دیگر دارایی‌هایی داشته باشد که یکتا هستند و با هیچ دارایی دیگری دقیقاً یکسان نیستند. مثلاً یک بلیت هوایپیما برای تاریخ و ساعتی خاص برای شماره پروازی خاص از یک مبدأ مشخص به یک مقصد مشخص است و شماره صندلی یکتایی نیز دارد. پس هیچ دو بلیت هوایپیمایی دقیقاً یکسان نیستند. برخلاف دو بیتکوین که کاملاً یکسان هستند، ارزش برابری دارند، و تعویض‌پذیر هستند.

کاربردهای توکن‌های تعویض‌ناپذیر بیشمار است. در حال حاضر فقط قسمت اندکی از کاربردهایی که می‌توانند داشته باشند را پاسخ گفته‌اند. در این پژوهه یک پلتفرم^۲ ساخته می‌شود که ساخت^۳ و انتقال توکن‌های تعویض‌ناپذیر را برای عموم در دسترس‌تر و آسان‌تر می‌کند. همچنین یکی از اهداف انجام این پژوهه آشنایی با تکنولوژی‌ها، استانداردها و فرایندهای توسعه این توکن‌هاست.

¹Fiat Money

²Platform

³Mint

۲.۱ شرح مسئله و روش انجام آن

پروژه تعریف شده، توسعه یک پلتفرم برای ساخت و انتقال توکن‌های تعویض‌ناپذیر به آسان‌ترین روش ممکن است. به نحوی که برای هر کسی به راحتی در دسترس باشد. نکته‌ی قابل توجه این است که در مسیر انجام این پروژه با تکنولوژی‌های موجود در این زمینه، چارچوب‌ها^۴، استانداردها و فرایند تست و بارگذاری آشنا شویم.

برای انجام این مراحل در قدم اول نحوه توسعه اپلیکیشن‌های غیرمت مرکز و برتری‌های نوشتن پروژه به صورت متن‌باز ذکر می‌شود. سپس چارچوب‌ها و ابزارهایی که برای ساخت یک اپلیکیشن غیرمت مرکز به توسعه دهنده کمک می‌کنند معرفی شده و نحوه استفاده از آن‌ها شرح داده می‌شود.

در گام بعد فرایند توسعه آغاز می‌شود، استانداردهای موجود برای نوشتن یک قرارداد برای توکن‌های تعویض‌ناپذیر شرح داده می‌شود و کاپو تا جای ممکن مطابق آن‌ها توسعه می‌یابد. برای قرارداد هوشمند نوشته شده تست می‌نویسیم و آن را روی شبکه تستی^۵ انتشار می‌دهیم. در گام بعد برای پلتفرم، واسط کاربری ساده‌ای نوشته می‌شود که با قرارداد هوشمند و همچنین کیف پول دیجیتال کاربر ارتباط برقرار می‌کند و سپس به کمک صفحات گیت‌هاب^۶ بارگذاری می‌شود تا در دسترس عموم کاربرها قرار بگیرد.

برای داکرایز^۷ کردن تست‌های قرارداد هوشمند، یک ایمیج داکر^۸ ترافل^۹ نوشته می‌شود. در قدم بعد هر دو بخش واسط کاربری و قرارداد هوشمند داکرایز می‌شوند و فرایند اجرای تست‌های قرارداد هوشمند و بارگذاری شدن واسط کاربری به صورت خودکار به کمک پایپلاین‌های گیت‌هاب پیاده‌سازی می‌شود.

⁴Frameworks

⁵Testnet

⁶Github Pages

⁷Dockerize

⁸Docker Image

⁹Truffle

۳.۱ اهداف کلی تحقیق

۱.۳.۱ گسترش کاربردهای توکن‌های تعویض‌ناپذیر

این توکن‌ها در همین مدت کوتاهی که به وجود آمده‌اند کاربردهای فراوانی را پوشش داده‌اند. اما همچنان قسمت بزرگی از این کاربردها صرفاً ثبت مالکیت آثار هنری دیجیتال است. در حالی که توکن‌های داده‌ای می‌توانند وسعت بسیار عظیم‌تری از کاربردها را پوشش دهند. از کاربری‌های روزانه مانند بلیت سینما و هوایپیما، تا مالکیت هر نوع دارایی واقعی یا مجازی.

با توجه به نحوه کار اکثر قراردادهای توکن‌های تعویض‌ناپذیر، معمولاً فقط مالک قرارداد می‌تواند توکن ایجاد کند، یا در قرارداد برای ایجاد توکن شرط‌هایی مانند حداکثر تعداد ممکن گذاشته می‌شود. این موضوع به این معنی است که اگر شخصی بخواهد خودش توکن‌هایی ایجاد کند و به دیگران انتقال دهد احتمالاً مجبور است که قرارداد هوشمند خودش را بنویسد و بارگذاری کند. این فرآیند نیاز به دانش فنی، آشنایی کامل با این زمینه و پرداخت هزینه‌های بارگذاری قرارداد روى شبکه بلاکچین دارد.

کاپو به هر آدرسی اجازه می‌دهد که به راحت‌ترین حالت ممکن و به هر تعداد که مورد نیاز است توکن تعویض‌ناپذیر روی این قرارداد ایجاد کند. به این ترتیب استفاده از کاپو برای عموم مردم آسان‌تر، ارزان‌تر و در دسترس‌تر است.

۲.۳.۱ ساخت روند توسعه قرارداد هوشمند

یکی از اهداف انجام این پروژه این است که پس از آشنایی با ایزارهای موجود، یک ساختار برای روند توسعه قرارداد هوشمند ایجاد شود. این ساختار به نحوی خواهد بود که توسعه قرارداد سریع‌تر و با اطمینان خاطر بیشتری انجام شود. در ساخت روند توسعه تاکید بر راحتی اضافه کردن تست‌ها و اجرای خودکار تست‌ها، آسان و خودکار بودن روند دیپلوی و اتصال بی‌دردسر فرانت‌اند به قرارداد هوشمند است.

۳.۳.۱ یادگیری

هدف دیگر انجام این پروژه یادگیری است. با توجه به رشد سریع و تازگی استفاده از تکنولوژی‌های بلاکچین و توکن‌های تعویض‌ناپذیر، با وجود تلاش برای ایجاد منابع یادگیری مناسب، همچنان فضاهای خالی، کمبودها و نیازمندی‌هایی وجود دارد که باید پاسخ گفته شوند. در طی انجام این پروژه با ابزارها، کتابخانه‌ها، چارچوب‌ها و استانداردهای نوشتمند آشنایی شویم، می‌آموزیم که هر یک چطور کار می‌کنند و چگونه می‌توانند به توسعه‌دهنده کمک کنند.

۴.۱ ساختار پایان‌نامه

پس از این مقدمه، در فصل ۲ مفاهیم اولیه توسعه قرارداد هوشمند بر بستر بلاکچین، کاربردها، مفاهیم و استانداردها توضیح داده می‌شود. در فصل ۳ ابزارهای توسعه قراردادهای هوشمند معرفی می‌شوند، مزایا و معایب هر یک بیان شده و نحوه استفاده از آن‌ها توضیح داده می‌شود. در فصل ۴ روند پیاده‌سازی شرح داده می‌شود. بررسی می‌شود که در هر مرحله از پیاده‌سازی چه کارهایی به چه ترتیبی انجام شده است. در فصل پنجم نیز نتایج توضیح داده می‌شوند و جمع‌بندی صورت می‌گیرد.

فصل ۲

مفاهیم اولیه و پیش زمینه

۱.۲ دلایل و برتری های متن باز بودن قراردادهای هوشمند

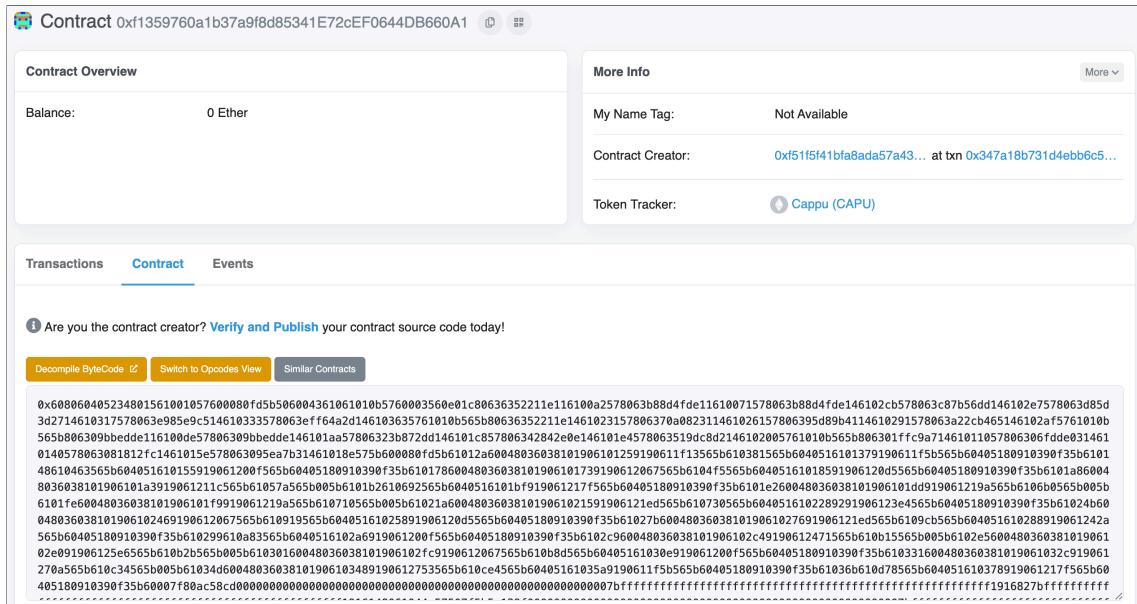
دلایل زیادی برای متن باز نوشتگی قراردادهای هوشمند وجود دارد، در ادامه تعدادی از این دلایل توضیح داده می شود.

دلیل اول، بلاکچین ها محروم‌انگی^۱ ندارند. همهی نودهای شبکه برای اجرای کد قرارداد هوشمند باید حداقل به بایت کدهای^۲ قرارداد هوشمند دسترسی داشته باشند. این بایت کدها در کاوشگرهای بلاکچین نیز قابل مشاهده هستند. همچنین دیکامپایلر^۳ هایی وجود دارند که از بایت کدهای قرارداد هوشمند کد سالیدیتی آن را به دست می آورند. پس در نتیجه تلاش برای مخفی کردن کدهای قرارداد هوشمند بیهوده خواهد بود. در تصویر ۱.۲ می توان بایت کدهای کاپو در اتراسکن را مشاهده کرد. همچنین دکمه دیکامپایل کردن بایت کدها نیز در تصویر مشخص است.

¹Confidentiality

²bytecodes

³Decompiler



شکل ۱.۲: بایت‌کدهای قرارداد کاپو

دلیل دوم، اصلی‌ترین مزیت اپلیکیشن‌های غیرمت مرکز نسبت به اپلیکیشن‌های مت مرکز عدم نیاز به اعتماد است. کاربرها می‌توانند کدهای قرارداد هوشمند را بخوانند و به کد نوشته شده اعتماد کنند. در حالی که اگر کد پرنامه برای همه کاربران قابل مشاهده نباشد کاربرها باید به سازندگان آن پرنامه اعتماد کنند.

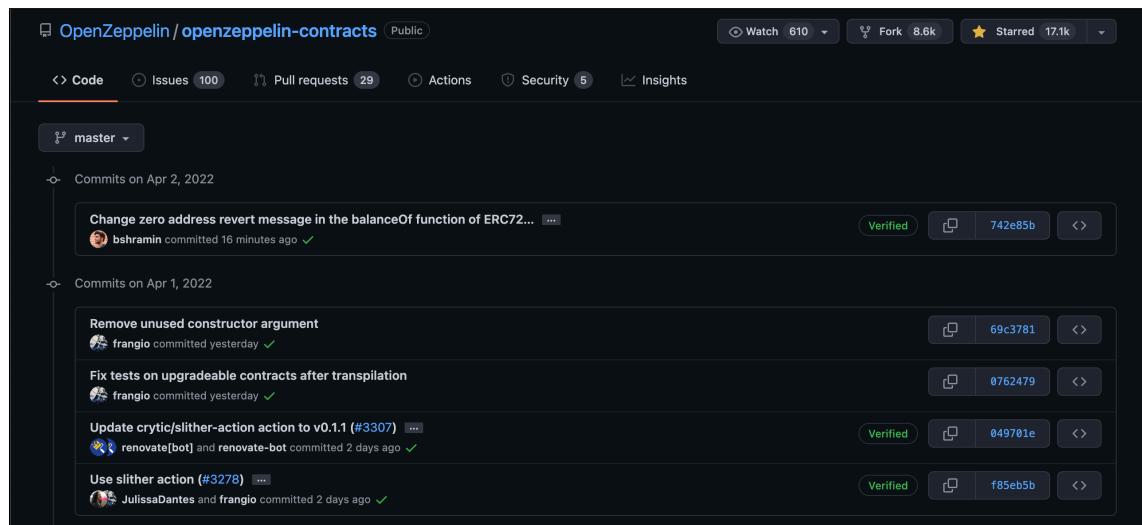
دلیل سوم، بارگذاری کردن قراردادهای هوشمند معمولاً آسان نیست و سرعت تغییرات پایین‌تر از اپلیکیشن‌های مرکز هست. پس امکان این که با پیدا شدن هر مشکل بتوان به سرعت آن را درست کرد کمتر وجود دارد و مسئله امنیت بسیار با اهمیت می‌شود. متن باز نوشتن قرارداد هوشمند باعث می‌شود چشم‌های بیشتری کدهای قرارداد را بخوانند و مشکلات احتمالی سریعتر مشخص و رفع شوند. تعداد زیادی از پروژه‌ها مانند کاپو از همان روز اول قرارداد هوشمند را به صورت متن باز توسعه می‌دهند، بعضی نیز ترجیح میدهند که پروژه به مرحله‌ای از توسعه برسد و سپس آن را متن باز می‌کنند.

در این حوزه سرعت پیشرفت و توسعه به دلیل متن باز بودن به شدت بالاست به نحوی که در طی اجرای این پروژه مرج ریکوئستی روی کتابخانه اپن زپلین⁴ زده شد که در همان روز مرج گردید. این موضوع علاوه بر این که نشان‌دهنده سرعت پیشرفت بسیار بالاست، این واقعیت را نیز نشان میدهد که در یک جامعه متن باز هر توسعه دهنده می‌تواند به پیشرفت جامعه به هر شکلی که می‌پسندد کمک کند، اشکالاتی که مشاهده می‌کند را گزارش

⁴OpenZeppelin

دهد یا تصحیح کند.

برای مثال همکاری^۵ نویسنده‌ی این پایان‌نامه در پروژه اپن‌زپلین را می‌توان در تصویر ۲.۲ مشاهده کرد. این مثال نشان‌دهنده این است که حتی در پروژه بسیار بزرگی مانند اپن‌زپلین نیز از کمک عموم توسعه‌دهنگان به راحتی پذیرش می‌شود.



شکل ۲.۲: در طی انجام پروژه مرج ریکوئستی روی اپن‌زپلین باز شد که در همان روز مرج گردید.

۲.۲ آشنایی با مفهوم توکن تعویض‌ناپذیر

شروع رمزارزها با توکن‌های تعویض‌پذیر بود. مفهوم تعویض‌پذیری به این معنی است که یک توکن با توکن دیگر تفاوتی ندارد و با جابه‌جا شدن آن‌ها تغییری ایجاد نمی‌شود. برای مثال یک بیت‌کوین با بیت‌کوین دیگر هیچ تفاوتی ندارد.

اما توکن‌های تعویض‌ناپذیر این‌گونه نیستند، هر یک از آن‌ها منحصر به فرد است و نمی‌توان آن‌ها را به جای یکدیگر به کار برد. در دنیای واقعی خانه می‌تواند مثال خوبی از یک دارایی تعویض‌ناپذیر باشد. هیچ دو خانه‌ای دقیقاً شبیه به هم، در یک مکان، در طبقه یکسان و دارای پلاک مشترک نیستند.

پس مثلاً به عنوان یک کاربرد، شهرداری می‌تواند یک قرارداد هوشمند ایجاد کند و به هر خانه یک توکن تعویض‌ناپذیر اختصاص دهد. به این صورت صاحب خانه به جای سند یک توکن تعویض‌ناپذیر دارد که مشخص

⁵<https://github.com/OpenZeppelin/openzeppelin-contracts/pull/3314>

می‌کند آن خانه متعلق به اوست. فروش خانه به راحتی انتقال آن توکن تعویض‌ناپذیر به شخص دیگری خواهد بود.

از نظر فنی هر توکن به این صورت یکتاست که یک Token ID یکتا در قراردادش دارد و هر قرارداد هم دارای یک آدرس یکتا در شبکه بلاکچین است. پس ترکیب Contract Address و Token ID باعث می‌شود که هر توکن در کل شبکه بلاکچین یکتا باشد.

۳۰.۲ کاربردها، حال و آینده

کاربرد توکن‌های تعویض‌ناپذیر تا به حال در دو دسته خلاصه می‌شود. دسته اول به عنوان صاحب یک اثر دیجیتال، مانند یک تصویر یا یک موسیقی. دسته دوم به عنوان یک جواز یا بلیت برای ورود به جایی یا دریافت کالایی برای مثال همایشی برگزار می‌شود که فقط دارندگان توکن‌های یک قرارداد هوشمند می‌توانند به آن وارد شوند.

معروف‌ترین پلتفرم معاملاتی این توکن‌ها اپن‌سی^۶ است که می‌توان در آن توکن‌های موجود را مشاهده کرد و یک توکن را توسط مزایده خرید و یا به فروش گذاشت. اپن‌سی در حال حاضر از قراردادهای شبکه‌های اتریوم و سولانا پشتیبانی می‌کند. دیگر شبکه‌ها نیز برای این منظور پلتفرم‌های خود را دارند، مانند شبکه اتم^۷ که در آن از پلتفرم استارگیز^۸ برای معامله توکن‌های تعویض‌ناپذیر استفاده می‌شود.

کاربردهای توکن‌های تعویض‌ناپذیر در آینده می‌تواند بسیار وسیع باشد. دارایی‌های فیزیکی دنیای واقعی، بلیت‌های ورود به یک مکان یا یک همایش، دارایی‌های دنیای مجازی مانند یک موسیقی یا دارایی در یک بازی و حتی دامنه‌های اینترنتی همه می‌توانند به توکن‌های تعویض‌ناپذیر تبدیل شوند. مزایای تبدیل این موارد به توکن‌های تعویض‌ناپذیر قابلیت نگهداری آسان‌تر، فروش و انتقال راحت‌تر، امنیت بیشتر، آزادی در تراکنش‌ها و آشکار بودن مالکیت دارایی بر همگان است.

⁶OpenSea

⁷Atom

⁸Stargaze

۴.۲ قراردادهای هوشمند و استانداردسازی

اکثر قراردادهای هوشمند قابلیت‌هایی مشابه با یکدیگر دارند. برای مثال گروهی از قراردادهای هوشمند توکن‌های تعویض‌پذیر دارند و گروهی توکن‌های تعویض‌ناپذیر. از طرفی اپلیکیشن‌هایی مانند کیف‌پول‌های دیجیتال، پلتفرم‌های معاملاتی و صرافی‌ها نیاز دارند که بتوانند دارایی‌های کاربر اعم از توکن‌های تعویض‌پذیر و تعویض‌ناپذیر را ببینند، به همین دلیل باید از نحوه صحبت کردن با قراردادهای هوشمند مطلع باشند.

برای ساده‌تر شدن این فرایند و همسان‌سازی اینترفیس این قراردادهای هوشمند استانداردهایی تعریف شده است که با استفاده از این استانداردها، هم فرایند توسعه قرارداد هوشمند آسان‌تر خواهد شد و هم ارتباط میان قرارداد هوشمند و اپلیکیشن‌های دیگر مانند کیف‌پول‌ها، پلتفرم‌های معاملاتی و ... آسان‌تر برقرار خواهد شد. از نمونه‌های معروف این استانداردها ERC20 برای قراردادهایی با توکن‌های تعویض‌پذیر و ERC721 برای قراردادهایی با توکن‌های تعویض‌ناپذیر است. در این پژوهه از استاندارد ERC721 استفاده می‌شود اما در مورد ERC1155 هم مطالعه شده و توضیح داده می‌شود. به طور خلاصه ERC1155 قابلیت‌های بیشتری از ERC721 دارد. قراردادی با این استاندارد می‌تواند هم توکن‌های تعویض‌پذیر و هم تعویض‌ناپذیر داشته باشد. جزئیات بیشتر هر یک از این قراردادها را می‌توان در مستندات اتریوم [۱] مطالعه کرد.

برای استفاده از این استانداردها از پکیج‌های متن‌بازی استفاده می‌شود که این استانداردها را پیاده‌سازی کرده‌اند و از آن‌ها در قرارداد مورد نظر ارجحیت می‌شود. یکی از بهترین پیاده‌سازی‌های این استانداردها توسط اپن‌زپلین [۲] انجام شده است که در این پژوهه نیز از همین پیاده‌سازی استفاده می‌شود.

۱.۴.۲ استاندارد ERC20

این استاندارد مناسب توکن‌های تعویض‌پذیر است. اینترفیسی تعریف می‌کند که نیازهای قراردادهایی با توکن‌های تعویض‌پذیر برطرف شود و نحوه تعامل برقرار کردن با آن‌ها یکسان گردد. در این استاندارد فقط می‌توان یک نوع توکن تعویض‌پذیر به تعداد دلخواه داشت. این استاندارد متدهایی برای تعریف حداکثر تعداد توکن‌های موجود، گرفتن موجودی یک آدرس، و انتقال توکن‌ها دارد. توضیحات دقیق‌تر در مورد این استاندارد را می‌توان در وبسایت اتریوم^۹ یا اپن‌زپلین^{۱۰} مشاهده کرد.

⁹<https://ethereum.org/en/developers/docs/standards/tokens/erc-20>

¹⁰<https://docs.openzeppelin.com/contracts/4.x/api/token/erc20>

۲.۴.۲ استاندارد ERC721

استفاده از استاندارد ERC721 برای توکن‌های تعویض‌ناپذیر بسیار مرسوم است. در این استاندارد متدها و ایونت‌هایی برای بکسان‌سازی اینترفیس قراردادهای دارای توکن‌های تعویض‌ناپذیر تعریف شده است. در این نوع قراردادها می‌توان به تعداد دلخواه توکن‌های متفاوت با یکدیگر داشت، هر توکن یک شناسه یکتا دارد که می‌تواند به صورت ترتیبی یا غیر ترتیبی ایجاد شود.

همچنین متدهای وجود دارد که می‌تواند شناسه یک توکن را به آدرسی تبدیل کند که اطلاعات آن توکن در آنجا موجود است. کاربرها می‌توانند توکن‌هایی که دارند را مشاهده کنند، به یکدیگر ارسال کنند یا به آدرس دیگری وکالت بدھند که توکن‌ها را به شخص دیگری ارسال کند.

تنها قابلیتی که به طور مشخص در این قرارداد معین نشده است که چگونه باید انجام شود، قابلیت ساخت توکن‌ها است. اکثر قراردادهای هوشمندی که توکن‌های تعویض‌ناپذیر دارند به کاربران اجازه ساخت توکن‌ها را نمی‌دهند و ساخت توکن‌ها فقط به آدرس صاحب قرارداد محدود می‌شود. اما در کاپو اینگونه نیست و هر کسی می‌تواند برای خودش توکن بسازد.

اطلاعات دقیق‌تر در مورد این استاندارد را نیز می‌توان در وبسایت اتریوم^{۱۱} یا اپن‌زپلین^{۱۲} مشاهده کرد.

۳.۴.۲ استاندارد ERC1155

تا اینجا با معروف‌ترین استانداردهای موجود برای قراردادهایی که توکن‌های تعویض‌پذیر یا تعویض‌ناپذیر دارند آشنا شدیم. اما همچنان نیازمندی‌هایی وجود دارند که توسط هیچ‌یک از این استانداردها برطرف نمی‌شوند. نیازمندی‌هایی مانند:

- داشتن توکن‌های تعویض‌ناپذیر با تعداد محدود به جای فقط یکی.
- داشتن همزمان چندین نوع توکن مختلف در یک قرارداد.
- انتقال همزمان چند توکن از انواع مختلف از کاربری به کاربر دیگر.

یک مثال از کاربردی که به این قابلیت‌ها نیاز دارد یک بازی مثل مونوپولی^{۱۳} باشد که در آن هر کاربر

¹¹<https://ethereum.org/en/developers/docs/standards/tokens/erc-721>

¹²<https://docs.openzeppelin.com/contracts/4.x/api/token/erc721>

¹³Monopoly

مقداری پول دارد که در واقع یک توکن تعویض‌پذیر هست، به عنوان دارایی چند خانه دارد که به عنوان توکن‌های تعویض‌ناپذیری هستند که از هرکدام فقط یکی وجود دارد و ممکن است چند کارت خروج از زندان داشته باشد که یکتا نیستند اما تعداد محدودی در بازی وجود دارد. استاندارد ERC1155 همه‌ی این نیازها را برطرف می‌کند. همه‌ی این چند نوع توکن می‌توانند همزمان در یک قرارداد هوشمند وجود داشته باشند.

در این استاندارد متدهایی برای تعریف نوعی توکن با تعداد مشخص وجود دارد. اگر نیاز به توکنی تعویض‌ناپذیر باشد تعداد آن یک قرارداده می‌شود. همچنین متدهایی برای ارسال تعداد مشخص از چند نوع توکن مختلف در یک تراکنش، دادن وکالت توکن‌ها به آدرس دیگر و گرفتن موجودی یک آدرس در این استاندارد وجود دارد. اطلاعات دقیق‌تر در مورد این استاندارد را نیز می‌توان در وبسایت اتریوم^{۱۴} یا اپن‌زپلین^{۱۵} مشاهده کرد.

¹⁴<https://ethereum.org/en/developers/docs/standards/tokens/erc-1155>

¹⁵<https://docs.openzeppelin.com/contracts/4.x/api/token/erc1155>

فصل ۳

آشنایی با ابزارهای توسعه

در تمام ابزارهای ذکر شده در ادامه این متن حتما باید به ورژن هر کدام دقت شود، ورژن‌ها باید با یکدیگر همخوانی داشته باشند در غیر این صورت مشکلاتی در کامپایل و اجرای برنامه به وجود می‌آید که به راحتی قابل رفع کردن نیستند. در انجام این پروژه عدم همخوانی ورژن‌های مختلف ابزارها با یکدیگر باعث ایجاد مشکلات فراوانی شد، به همین دلیل ورژن مورد نیاز هر ابزار در توضیحات پروژه ذکر شده است.

۱.۳ ابزارهای ساده

• ویرایشگر

برای برنامه نویسی این قرارداد هوشمند از ویرایشگر VSCode با نصب پلاگین مربوط به^۱ Solidity استفاده شده است. این پلاگین با یافتن اشتباه‌ها پیش از کامپایل و راهنمایی در نوشتن کد قرارداد کمک شایانی به افزایش سرعت توسعه می‌کند.

• ورژن‌کنترل

این پروژه از روز نخست به صورت متن‌باز توسعه یافته، برای توسعه یک پروژه به صورت متن‌باز اولین ابزار مورد نیاز یک برنامه ورژن‌کنترل است که نسخه‌های متفاوت و تغییر یافته کدها را به صورت مرتب

¹<https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity>

نگهداری کند. برای این منظور از گیت‌هاب استفاده شده.

• پکیج‌های Node و NPM

از آنجایی که کدهای سالیدیتی در واقع جاواسکریپت هستند، به ابزارهای توسعه اپلیکیشن‌های جاواسکریپت برای توسعه سالیدیتی نیاز است. ابزارهایی مانند Node برای کامپایل کردن برنامه‌های جاواسکریپت و npm که مدیریت پکیج‌های جاواسکریپتی که نصب می‌شود را به عهده دارد.

۲.۳ کیف پول متامسک

کیف پول دیجیتال متامسک از پرکاربردترین کیف پول‌ها برای ارتباط برقرار کردن با اپلیکیشن‌های غیرمت مرکزی و Web3 است. کاپو نیز برای امضای تراکنش‌ها و ایجاد ارتباط با شبکه بلاکچین از کیف پول متامسک استفاده می‌کند. برای انجام صحیح این عملیات کاربر باید از پیش کیف پول متامسک را نصب کرده باشد و سپس با انتخاب گزینه Connect Wallet، کاپو درخواست اتصال به کیف پول و دریافت آدرس کاربر را به متامسک ارسال می‌کند، متامسک نیز پس از دریافت درخواست کاپو از کاربر اجازه اتصال به اپلیکیشن را می‌گیرد و در صورت تایید کاربر آدرس کیف پول را به کاپو می‌دهد.

از این پس هرگاه که کاربر بخواهد در کاپو تراکنشی از جمله ساخت توکن جدید یا انتقال یک توکن به آدرس دیگر را انجام دهد کاپو از متامسک درخواست می‌کند که با کلید خصوصی^۲ کاربر آن تراکنش را امضا کند، متامسک از کاربر تایید تراکنش را می‌گیرد و امضا را انجام می‌دهد و تراکنش به شبکه بلاکچین ارسال می‌شود.

۳.۳ چارچوب‌ها و کتابخانه‌ها

به دلیل تازگی بحث توسعه اپلیکیشن‌های غیرمت مرکز ابزارهای کمی در این زمینه وجود دارند و همین ابزارها هم معمولاً مشکلاتی دارند و به بلوغ کامل نرسیده‌اند. اما با توجه به این که اکثر ابزارها و چارچوب‌ها و کتابخانه‌های توسعه اپلیکیشن‌های غیرمت مرکز متن‌باز هستند، سرعت رشد و تکامل بالایی دارند و به کمک توسعه‌دهندگان^۳ این

²Private key

³Developers

حوزه، هر روز نسبت به روز گذشته پیشرفت می‌کند.

برای توسعه این پروژه از چارچوب ترافل^۴، کتابخانه‌ی اپن‌زپلین^۵ و کتابخانه‌ی Web3JS^۶ استفاده شده است. در این قسمت به توضیح هر یک از این موارد پرداخته می‌شود.

۱.۳.۳ چارچوب ترافل

این چارچوب ابزارهای اولیه برای ساخت، کامپایل، تست، بارگذاری و مایگریشن قراردادهای هوشمند به زبان سالیدیتی را فراهم می‌کند. پس از نصب این ابزار با اجرای دستور truffle init می‌توان یک پروژه جدید ترافل ساخت ۱.۳، همچنین می‌توان با استفاده از دستور truffle unbox از یکی از تمپلیت‌های آماده ترافل استفاده کرد.

```
→ new-project truffle init
Starting init...
=====
> Copying project files to /Users/AminBSHR/Desktop/Thesis/new-project
Init successful, sweet!
Try our scaffold commands to get started:
$ truffle create contract YourContractName # scaffold a contract
$ truffle create test YourTestName          # scaffold a test
http://trufflesuite.com/docs
```

شکل ۱.۳: اجرای دستور truffle init

پس از ساخت پروژه با اجرای دستور truffle develop می‌توان وارد خط فرمان ترافل شد.

⁴<https://trufflesuite.com>

⁵<https://openzeppelin.com/contracts>

⁶<https://github.com/ChainSafe/web3.js>

```
+ back git:(main) ✘ truffle develop
Truffle Develop started at http://127.0.0.1:9545/
Accounts:
(0) 0x00eae7f45de5837119f5dd65fe63e58dcf8f7138
(1) 0x09ef5651770dccb33d926a1b75e10b2944924736
(2) 0x3e6cfcf6153d6dd9e2654afc8cfda499818c3e9
(3) 0xcefd7ce63b03aaccacd420828abe96acbe968d7
(4) 0xa84ab5be39670309d305e7aad3bcc347e75e9537
(5) 0x318913b83fe0c2d63db29bb84e39b6e39cccd93ef
(6) 0xde2602e64a047482e4606af26c89abb97afe6bd9
(7) 0x4d85fb20085db61ab33e8159bf88f3e1a7a1279
(8) 0xa031538284a6910cf832e25b2ec8105f52a87b13
(9) 0x3973a0026d8e807d0b9b5fb91c35a16ea990063a

Private Keys:
(0) 9453db0e1f3c1034f80d01b335b141e0d519f21aced3dbb7d2da54f37d773d6d
(1) bf85b2b427c669793da84d9c98e78d0fabd8676938d5c87cc01de50d42e235b7
(2) eff37f33a88597e7434be9def2e22594047c678a451a7f9581e8378a839e19c0
(3) fdd92e2593b15465bb7d6b61b3894c9fe50acf4e60327e7c1c2f378664dc8
(4) 0c50a6d545d747b33db3744321eea3e212400c4cd9497a0211fe4bd729a90c9f
(5) 28f97ce0d95990b234e50438ee46fd27461cf2ecf429131c9aa4176c1c85a0ab
(6) e6eb577c2aa69993aeab80ad275c346c6cf9b8506e8ab29cc69234744593a622
(7) 039c1150cf5eb82756320f0c904035cf0017da2a87c4c1d2dd057a39b3f0c452
(8) a78d582b2fad08f58f59680ade3de3e8a9e09fe101c8df850e5444c1b59c0
(9) 77e77e250d6551b2d7af2b2088e3508e84577ca39debd66c41136ab7a90e69f76

Mnemonic: attack left advance palm leader coconut doll enroll gorilla outdoor indoor erupt
⚠ Important ⚠ : This mnemonic was created for you by Truffle. It is not secure.
Ensure you do not use it on production blockchains, or else you risk losing funds.

truffle(develop)>
```

شکل ۲.۳: اجرای دستور truffle develop

دستورات لازم برای اجرای تست‌ها، کامپایل کردن قرارداد هوشمند یا بارگذاری آن روی شبکه مورد نظر از طریق این خط فرمان قابل اجرا هستند. این پلتفرم ابزارهای فراوانی را در اختیار توسعه دهنده قرار می‌دهد که با تعداد بیشتری از آن‌ها در بخش پیاده‌سازی و بارگذاری کاپو آشنا می‌شویم. همچنین از بزرگترین مزایای استفاده از این چارچوب برقراری ارتباط بسیار آسان با ابزارهای دیگر مانند گاناچه^۷ و دریزل است.

۲.۳.۳ کتابخانه اپن زپلین

یکی از معروف‌ترین کتابخانه‌های قراردادهای هوشمند و استانداردهای موجود در این کتابخانه کاملاً تست شده، داکیومنت شده، ایمن و پایه بسیاری از قراردادهای هوشمند بر بستر بلاکچین هستند. استانداردهای ذکر شده در این متن مانند، ERC1155، ERC721، ERC20، ERC155 به همراه تعداد زیادی استانداردهای دیگر در این کتابخانه پیاده‌سازی شده‌اند.

در کاپو نیز از استاندارد ERC721 پیاده‌سازی شده در این کتابخانه استفاده شده است. برای استفاده از قراردادهای اپن زپلین در قدم اول باید این کتابخانه به کمک دستور

⁷Ganache

نصب شود. پس از نصب کتابخانه، می‌توان از قراردادهای آن ارث‌بری کرد، در تصویر ۳.۳ مشاهده می‌شود که کاپو چگونه از قرارداد ERC721 موجود در اپن زپلین و همچنین یک قرارداد هوشمند به اسم Helper که در همین پروژه نوشته شده ارث‌بری کرده است.

```

2 pragma solidity >=0.4.22 <0.9.0;
3 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
4 import "./Helper.sol";
5
6 contract Cappu is ERC721, Helper {
7     constructor() ERC721("Cappu", "CAPU") {}
8 }
```

شکل ۳.۳: ارث‌بری از استاندارد ERC721 پیاده‌سازی شده توسط اپن زپلین

۳.۳.۳ کتابخانه Web3JS

تراکنش‌های با یک قرارداد هوشمند می‌تواند به ۲ حالت باشد. در حالت اول فقط اطلاعات شبکه بلاکچین خوانده می‌شود و حالت^۸ آن تغییری داده نمی‌شود، متدهای از این جنس از نوع view یا pure هستند. حالت دوم تراکنش‌هایی هستند که باعث تغییر اطلاعات شبکه بلاکچین می‌شوند.

واسط کاربری یک اپلیکیشن غیرمت مرکز برای انجام نوع اول تراکنش‌های نهایتاً فقط به آدرس کاربر نیاز دارد که اطلاعات مربوط به او را از قرار داد بگیرد. در حالت دوم نیاز است که تراکنشی بر روی شبکه ثبت شود که نیازمند امضای تراکنش توسط کلید خصوصی کاربر، پرداخت کارمزد تراکنش و ارسال آن به نودهای شبکه است.

کتابخانه‌ی Web3JS به توسعه دهنده کمک می‌کند که واسط کاربری اپلیکیشن را به کیف پول دیجیتال کاربر و شبکه بلاکچین متصل کند. با ایجاد این اتصال آدرس کاربر توسط کیف پول دیجیتال در اختیار واسط کاربری قرار می‌گیرد و هرگاه که واسط کاربری بخواهد تراکنشی را روی شبکه ارسال کند نیز از کیف پول کاربر می‌خواهد که با داشتن کلید خصوصی کاربر آن تراکنش را امضای روی شبکه ارسال کند. طبیعتاً کیف پول کاربر برای انجام هر یک از این مراحل از کاربر درخواست تاییدیه می‌کند.

⁸State

۴.۳ شبکه محلی برای توسعه

برای توسعه یک قرارداد هوشمند نیاز است که پس از هر تغییر کامپایل و روی یک شبکه بلاکچین بارگذاری شود، به نحوی که بواسطه کاربری اپلیکیشن و همچنین کیف پول متماسک بتوانند به آن متصل شوند. از شبکه اصلی نمی‌توان استفاده کرد زیرا هر بارگذاری روی شبکه اصلی هزینه‌ای خواهد داشت و بارگذاری‌های پیاپی روی شبکه امکان پذیر نخواهد بود. اگر بخواهیم برای توسعه از شبکه تستی هم استفاده کنیم گرچه هزینه‌ای نخواهد داشت اما بسیار زمان بر خواهد بود، گرچه انجام تراکنش‌ها روی شبکه تستی معمولاً سریعتر از شبکه اصلی انجام می‌شود اما همچنان توسعه دهنده زمان زیادی را برای هر بارگذاری صرف خواهد کرد.

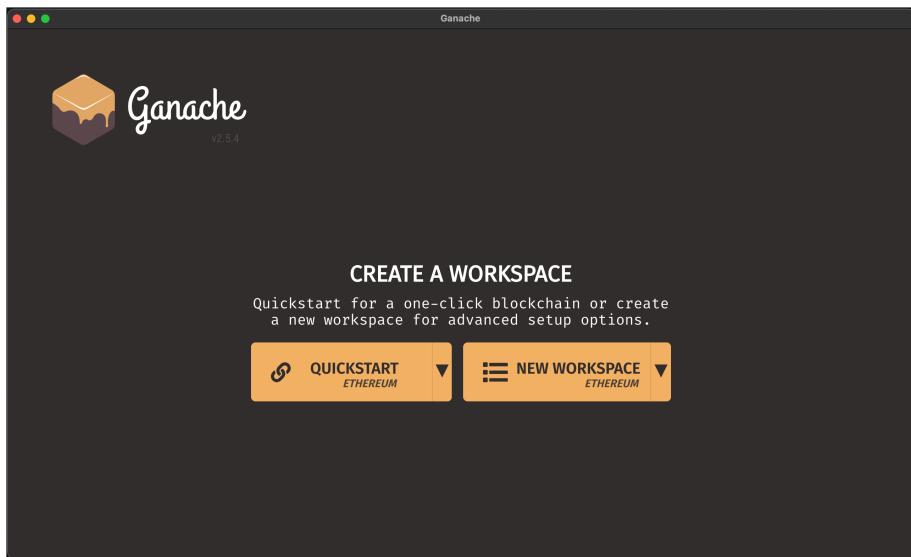
راه حل این مشکل این است که توسعه دهنده روی ماشین خودش یک شبکه محلی داشته باشد که بتواند بلافاراصله پس از ایجاد یک تغییر روی قرارداد هوشمند آن را کامپایل و بارگذاری کند. ترافل باید بتواند به این شبکه محلی متصل شود و قرارداد را روی آن بارگذاری کند. بواسطه کاربری و متماسک نیز باید بتوانند به این شبکه متصل شوند که با قرارداد هوشمند ارتباط برقرار کنند.

اگرچه ابزارهای زیادی برای ساخت این شبکه محلی وجود دارند، اما یکی از بهترین و راحت‌ترین ابزارها برای این منظور برنامه‌ی گاناچه هست. این ابزار با توجه به این که متعلق به اکوسیستم ترافل هست به آسانی به آن متصل می‌شود و با اضافه کردن آدرس آن به شبکه‌های متماسک، این کیف پول هم به شبکه محلی متصل می‌شود. جزئیات ساخت شبکه محلی و اتصال ترافل و متماسک به آن به ترتیب زیر است.

پس از نصب برنامه گاناچه باید یک محیط کار^۹ اتریوم^{۱۰} ساخته شود. برای انجام این کار گزینه New workspace (Ethereum) انتخاب می‌شود. این دکمه در تصویر ۴.۳ قابل مشاهده است.

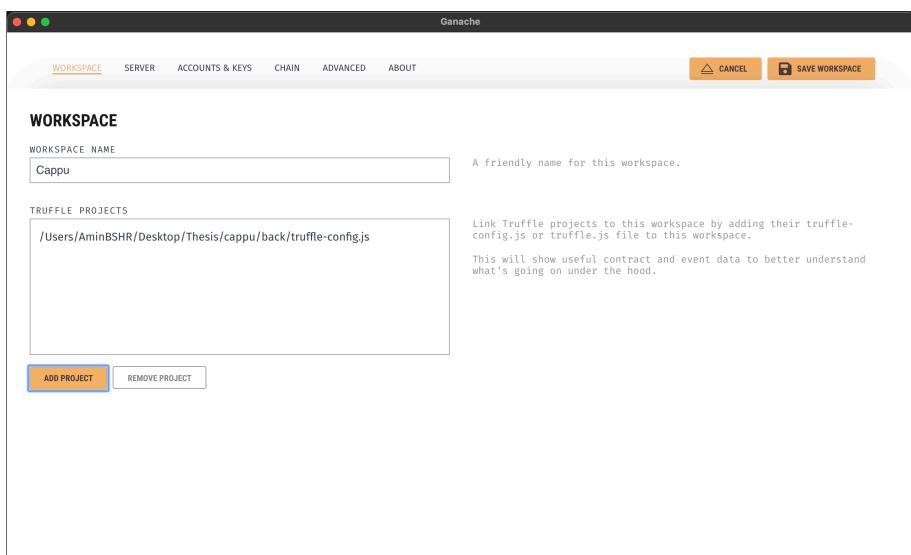
⁹Workspace

¹⁰Ethereum



شکل ۴.۳: صفحه اول گاناچه

سپس در صفحه باز شده نام محیط توسعه وارد، فایل truffle-config.js مربوط به پروژه مورد نظر انتخاب و دکمه save workspace زده می‌شود. انجام این مرحله در تصویر ۵.۲ قابل مشاهده است.



شکل ۵.۳: ساخت شبکه جدید در گاناچه

پس از انجام این مراحل محیط توسعه ساخته شده است و می‌توان جزئیات شبکه محلی را مشاهده کرد.

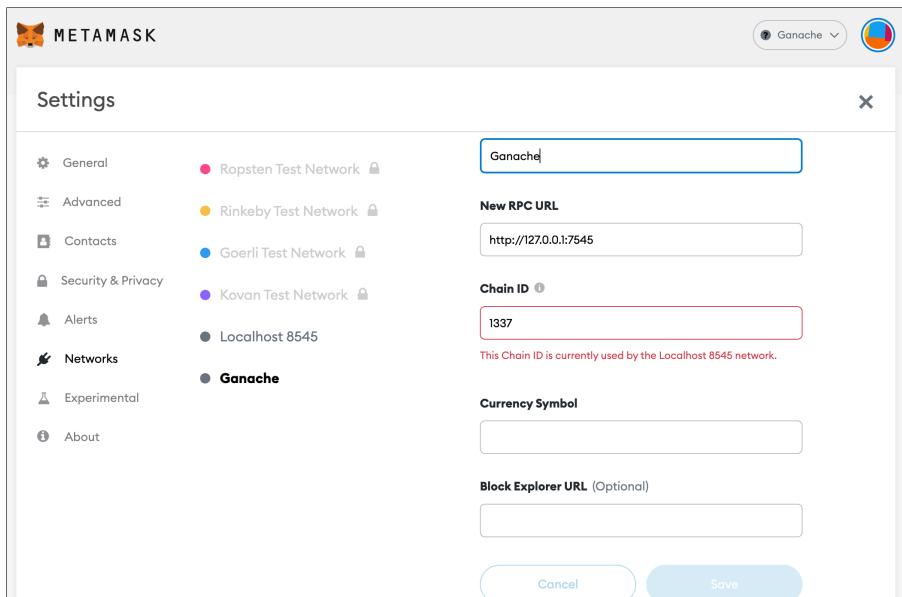
MNEMONIC	HD PATH			
camera expose urban enter absorb crime sleep kiwi neutral vault course wonder	m/44'/60'/0'/0/account_index			
ADDRESS	BALANCE	TX COUNT	INDEX	
0xBF88E8776e189E418431fEa09d97f51d21985Bad	100.00 ETH	0	0	🔗
0x06863839Ec88c65Bc166304c58AD57895b39Ad1f	100.00 ETH	0	1	🔗
0x734F0e8aD593507769AA99C7C4F868C070422BFD	100.00 ETH	0	2	🔗
0xEEd4C8CC08Db0aA0A61591496da50D87a0Ae0264	100.00 ETH	0	3	🔗
0xF1E00BA373215bEC62253d64613bE2a9F8649107	100.00 ETH	0	4	🔗
0x18b9946415a6Fc6a4ef0A552ADa0B8094ff5e8FF	100.00 ETH	0	5	🔗
0xB97221c3b3bD88A7661c3E57A364Dd0B91935523	100.00 ETH	0	6	🔗

شکل ۳.۶: مشاهده جزئیات شبکه ساخته شده

از آنجایی که در مرحله قبل برای ساخت این محیط توسعه فایل `truffle-config.js` پروژه انتخاب شد، حال اگر دستورات `truffle console` یا هر دستور دیگری مانند `migrate` بدون انتخاب شبکه بلاکچین خاصی اجرا شود به صورت پیش‌فرض روی این شبکه محلی انجام می‌شود.

حال فقط باید متامسک نیز به این شبکه محلی متصل شود. برای انجام این کار پس از نصب افزونه‌ی متامسک روی مرورگر کروم، در قسمت تنظیمات^{۱۱} و سپس شبکه‌ها^{۱۲} یک شبکه جدید مطابق تصویر ۷.۳ ساخته می‌شود، همانطور که در تصویر ۶.۳ مشخص است اطلاعات شبکه محلی در صفحه اصلی گانچه قابل مشاهده هستند.

¹¹Settings¹²Networks



شکل ۷.۳: تنظیمات شبکه‌های متامسک

پس از ذخیره شبکه جدید کافیست که برای توسعه شبکه گاناچه انتخاب شود. همچنین باید یکی از آدرس‌هایی که در صفحه اصلی گاناچه نمایش داده می‌شوند به عنوان کیف پول در متامسک وارد شود. برای انجام این کار علامت کلید کنار یکی از آدرس‌های نمایش داده شده در صفحه اصلی گاناچه انتخاب می‌شود و به کمک کلید اختصاصی نمایش داده شده کیف پول در متامسک وارد می‌شود.

فصل ۴

پیاده‌سازی

۱.۴ نوشتن کد قرارداد

در این بخش به بررسی مراحل و نحوه نوشتن کد قرارداد هوشمند پرداخته می‌شود.

۱.۱.۴ نیازمندی‌های قرارداد هوشمند

نیازمندی‌های اصلی کاپو به ترتیب زیر است.

- هر آدرس در شبکه بتواند یک داده‌ی متنی را به آسان‌ترین و کم هزینه‌ترین روش ممکن به یک توکن NFT تبدیل کند.
- هر آدرس بتواند توکن‌های خود را به اشخاص دیگر انتقال دهد یا در بازارهای معاملات NFT بفروشد.
- در صفحه اول وبسایت تعداد کل توکن‌های ساخته شده تا به حال و تعداد کل دارندگان توکن نمایش داده شود.
- قابلیت‌های قرارداد هوشمند تست شده باشد.

۲.۱.۴ ارثبری

با توجه به مزایای ذکر شده در مورد استانداردسازی قراردادهای هوشمند، انتخاب درستی است که برای پیاده‌سازی این کاربری از یکی از استانداردها استفاده شود. ارثبری از استانداردهای یک کتابخانه متن‌باز مزایای زیر را فراهم می‌کند.

- به دلیل وجود کدهای پایه به صورت آماده سرعت توسعه پروژه افزایش می‌یابد.
- ارتباط دیگر پروژه‌ها با پروژه کاپو به راحتی انجام می‌شود.
- امنیت قرارداد و درستی آن حداقل در سطوح پایه‌ای تا حد خوبی تضمین شده است.

قرارداد هوشمند کاپو از استاندارد ERC721 پیاده‌سازی شده در کتابخانه اپن‌زپلین^۱ ارثبری می‌کند که یکی از معروف‌ترین کتابخانه‌های پیاده‌کننده استانداردهای قرارداد هوشمند است.

۳.۱.۴ توجه به هزینه تراکنش و نوع توابع

در نوشتن یک قرارداد هوشمند باید به نکات زیر توجه کنیم.

- میزان حافظه‌ای که اشغال می‌کنیم.
- حجم بایت‌کد.
- میزان عملیات هر متد، به خصوص متدهایی که مکرراً مورد استفاده کاربر قرار می‌گیرند.
- نوع هر متد، که مشخص می‌کند هر متد تا چه حد روی شبکه بلاکچین تغییر ایجاد می‌کند.

توجه نکردن به هریک از این موضوعات باعث می‌شود که قرارداد هوشمند به اندازه کافی بهینه عمل نکند و کاربر وادار به پرداخت هزینه تراکنش^۲ یا هزینه تراکنش بیشتر شود. یکی از مهمترین نکاتی که برای بهینه‌تر رفتار کردن قرارداد هوشمند باید به آن توجه کنیم نوع هر متد است.

¹ <https://github.com/OpenZeppelin/openzeppelin-contracts>

² Gas fee

اگر متدهای از نوع pure تعریف شود به این معنی است که به هیچ اطلاعاتی از شبکه بلاکچین نیاز ندارد و همه اطلاعاتی که لازم دارد را در اسکوپ^۳ خودش دارد. اگر متدهای از نوع view باشد به این معنی است که به اطلاعاتش روی شبکه بلاکچین نیاز دارد اما فقط می‌خواهد که آنها را بخواند و نمی‌خواهد تغییری در آنها ایجاد کند. این دو نوع متدهای نیازی به پرداخت کارمزد تراکنش توسط کاربر ندارند، اما اگر در تعریف متدهای ذکر نشود که یکی از این دو نوع است، اینطور در نظر گرفته می‌شود که نیاز به بروزرسانی اطلاعاتش در شبکه بلاکچین دارد و از کاربری که آن را فراخوانی کرده است هزینه تراکنش دریافت می‌شود.

۴.۱.۴ جزئیات فنی پیاده‌سازی

مینت کردن در این قرارداد به آدرس‌های مشخص محدود نیست و همه می‌توانند توکن بسازند. بسیاری از قراردادها برای صرفه‌جویی در هزینه تراکنش کاربران اکثر اطلاعات مربوط به توکن‌ها را در قرارداد نگه نمیدارند و فقط داده‌های بسیار مهم توکن را در شبکه بلاکچین نگهداری می‌کنند. از آنجایی که کاپو یک قرارداد همه منظوره است و ممکن است استفاده‌های فراوانی داشته باشد، تصمیم‌گیری این مورد به عهده کاربر قرارداد گذاشته می‌شود.

در کاپو شناسه هر توکن از هش^۴ داده‌های توکن به دست می‌آید. این نحوه عملکرد چند مزیت ایجاد می‌کند. به این ترتیب هیچ دو توکنی نمی‌توانند داده‌های یکسان داشته باشند، زیرا در این صورت شناسه آنها باید یکسان باشد و این امکان پذیر نیست زیرا شناسه توکن‌ها یکتاست. همچنین شناسه توکن‌ها دیگر ترتیبی نخواهند بود و ترتیب ساخت توکن‌ها مشخص نخواهد بود.

در یک قرارداد ERC721 استاندارد فقط شناسه توکن‌ها ذخیره می‌شود. در کاپو علاوه بر شناسه توکن‌ها یک نگاشت^۵ از شناسه توکن‌ها به داده‌ی آنها با نام tokenDatas نیز نگهداری می‌شود. همچنین در کاپو نگاشت دیگری نیز از آدرس به لیست توکن‌های آن آدرس با نام ownerTokens نگهداری می‌شود. متغیر اول کمک می‌کند که با داشتن شناسه یک توکن به راحتی داده‌های آن توکن به دست آورده شوند. متغیر دوم نیز کمک می‌کند که به راحتی بتوان توکن‌های یک آدرس را به دست آورد. دو متغیر دیگر با نام‌های numberTokenHolders و numberMintedTokens نیز در کاپو نگهداشته می‌شوند که برای نمایش آمار استفاده از قرارداد در صفحه

³Scope

⁴Hash

⁵Mapping

اصلی اپلیکیشن مورد استفاده قرار می‌گیرند.

متد mint به نحوی نوشته شده است که برای عموم قابل استفاده باشد. پس از محاسبه هش داده‌ی توکن از آن به عنوان شناسه توکن استفاده می‌کند، توکن را می‌سازد و متغیرهای tokenDatas و numberOfMintedTokens را بروزرسانی می‌کند. پیاده‌سازی این متد را می‌توان در تصویر ۱.۴ مشاهده کرد.

```

14     function mint(string memory data) public {
15         uint256 theHash = uint256(keccak256(abi.encode(data)));
16         _safeMint(msg.sender, theHash);
17         _tokenDatas[theHash] = data;
18         _numberOfMintedTokens++;
19     }

```

شکل ۱.۴: پیاده‌سازی تابع mint

متد afterTokenTransfer از استاندارد ERC721 به نحوی بازنویسی^۶ شده است که پس از هر انتقال توکن با بررسی آدرس‌های مبدا و مقصد، متغیرهای numberOfTokenHolders و numberOfMintedTokens را بروزرسانی کند. نحوه عملکرد این متد در تصویر ۲.۴ مشخص است.

⁶Overwrite

```

21     function _afterTokenTransfer(
22         address from,
23         address to,
24         uint256 tokenId
25     ) internal virtual override {
26         if (from != address(0)) {
27             _ownerTokens[from] = removeItemFromArray(
28                 tokenId,
29                 _ownerTokens[from]
30             );
31             if (_ownerTokens[from].length == 0) {
32                 _numberOfTokenHolders--;
33             }
34         }
35         if (to != address(0)) {
36             _ownerTokens[to].push(tokenId);
37             if (_ownerTokens[to].length == 1) {
38                 _numberOfTokenHolders++;
39             }
40     }
41 }
```

شکل ۲.۴: پیاده‌سازی تابع afterTokenTransfer

متدهای جدیدی با نام `getUserTokens` نیز نوشته شده است که در استاندارد ERC721 به صورت پیش‌فرض وجود ندارد. این متدهای جدید با گرفتن یک آدرس و استفاده از `ownerTokens` و `tokenDatas` دو خروجی برگردانند، لیستی از شناسه توکن‌های آدرس و لیستی از داده‌های توکن‌های آدرس. محتوای این متدهای جدید در تصویر ۳.۴ قابل مشاهده است.

```

43     function getUserTokens(address user)
44         public
45         view
46         returns (uint256[] memory, string[] memory)
47     {
48         uint256[] memory tokens = _ownerTokens[user];
49         string[] memory datas = new string[](tokens.length);
50         for (uint256 i = 0; i < tokens.length; i++) {
51             datas[i] = _tokenDatas[tokens[i]];
52         }
53         return (tokens, datas);
54     }
```

شکل ۳.۴: پیاده‌سازی تابع getUserTokens

همچنین از آنجایی که سالیدیتی به طور پیش‌فرض امکان حذف یک داده از یک آرایه با داشتن مقدار آن را ندارد، عدم وجود این قابلیت هزینه‌بر بودن آن است، در سالیدیتی توسعه دهنده‌گان به استفاده از نگاشت و دوری از آرایه‌ها تشویق می‌شوند. اما برای نمایش نحوه ارتبری از دو یا چند قرارداد پدر، برای کاپو یک قرارداد به نام Helper نوشته شد که این قابلیت را فراهم می‌کند. این قرارداد در تصویر ۴.۴ قابل مشاهده است. کاپو علاوه بر Helper از قرارداد ERC721 نیز ارتبری می‌کند.

```

4   contract Helper {
5     function removeItemFromArray(
6       uint256 valueToFindAndRemove,
7       uint256[] memory array
8     ) internal pure returns (uint256[] memory) {
9       uint256[] memory auxArray = new uint256[](array.length - 1);
10      uint8 found = 0;
11      for (uint256 i = 0; i < array.length; i++) {
12        if (array[i] != valueToFindAndRemove) {
13          auxArray[i - found] = array[i];
14        } else {
15          found = 1;
16        }
17      }
18      if (found == 0) {
19        return array;
20      }
21      return auxArray;
22    }
23  }
```

شکل ۴.۴: پیاده‌سازی قرارداد Helper

۲۰.۴ نوشت و اجرای تست‌ها

پیش‌تر اشاره شد که از مزیت‌های ارتبری از کتابخانه‌های متن‌باز معروف این است که احتمال وجود خطأ و مشکل امنیتی به شدت کمتر می‌شود. یکی از دلایل این مسئله این است که این کتابخانه‌ها پوشش تستی به شدت بالایی دارند. به همین دلیل می‌توان تا حدی به عملکرد قرارداد پدر اطمینان خاطر داشت و بیشتر روی تست کردن قابلیت‌های اضافه شده در قرارداد هوشمند فرزنده مرکز داشت.

در کاپو برای هر عملکرد قرارداد تست نوشته شده است. یکی از ساده‌ترین تست‌های نوشته شده تست فرآیند ساخت یک توکن است که در آن پس از بارگذاری قرارداد با فراخوانی متده mint یک توکن ساخته می‌شود و سپس

با فراخوانی متدهای آدرس سازنده توکن بررسی می‌شود و انتظار می‌رود که پس از ساخت یک توکن، دارایی آدرس سازنده توکن یک باشد. این تست را می‌توان در تصویر ۴.۵ مشاهده کرد.

```

3  contract("Cappu", [accounts] => {
4    it("should mint a token", async () => {
5      const cappu = await Cappu.deployed();
6
7      await cappu.mint("Hey there!", { from: accounts[0] });
8
9      const balance = await cappu.balanceOf(accounts[0], {
10        from: accounts[0],
11      });
12
13      assert.equal(balance, 1);
14    });
15  });

```

شکل ۴.۵: نمونه یکی از تست‌های قرارداد کاپو

پس از نوشته شدن تست‌ها می‌توان آنها را با اجرای دستور truffle test اجرا کرد. این دستور پس از اجرای تست‌ها نتیجه و زمان اجرای هر تست را به عنوان خروجی نمایش می‌دهد. نمونه اجرای این دستور را می‌توان در تصویر ۴.۶ مشاهده کرد.

```

→ back git:(main) ✘ truffle test
Using network 'test'.

Compiling your contracts...
=====
> Compiling ./contracts/Cappu.sol
> Artifacts written to /var/folders/rp/gbmd1rwd0p1325ck73ckz_rc0000gn/T/test--15817-mpJlgp9Ttlws
> Compiled successfully using:
  - solc: 0.8.10+commit.fc410830.Emscripten clang

Contract: Cappu
  ✓ should return correct homepage info (6505ms)

Contract: Cappu
  ✓ should mint a token (1980ms)

Contract: Cappu
  ✓ should transfer a token (2301ms)

3 passing (13s)

```

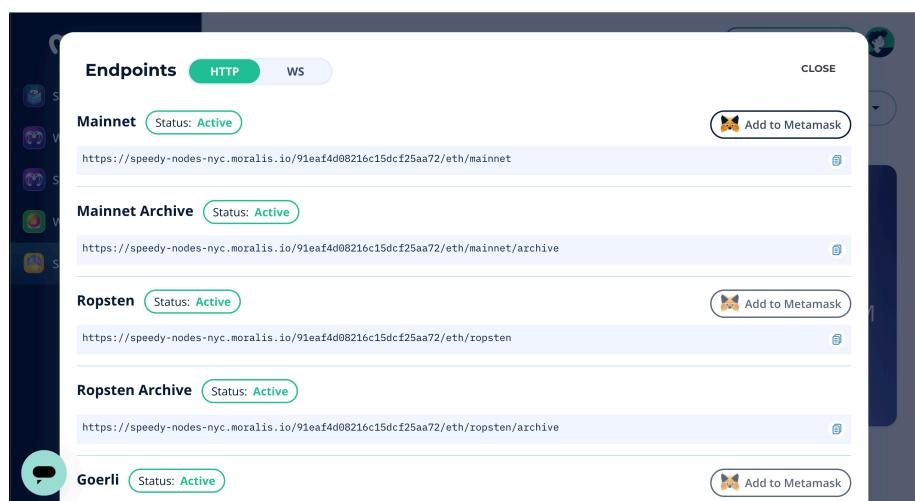
شکل ۴.۶: نمونه خروجی اجرای تست‌های قرارداد

۳.۴ بارگذاری قرارداد روی شبکه تستی راپستن

تا اینجا قرارداد هوشمند نوشته و تست شده است، در این مرحله روی شبکه تستی راپستن بارگذاری می‌شود. فرآیند بارگذاری شدن کاپو به کمک چارچوب ترافل قدم به قدم شرح داده می‌شود.

۱.۳.۴ یافتن آدرس یکی از نودهای شبکه برای ارسال تراکنش بارگذاری قرارداد به آن

آدرس نودهای یک شبکه بلاکچین همه به صورت عمومی در دسترس هستند زیرا نودها باید بتوانند یکدیگر را بینند. راههای زیادی برای به دست آوردن آدرس یک نود وجود دارد. یکی از آسان‌ترین راههای به دست آوردن آدرس یکی از نودهای شبکه مراجعه به وبسایت ماینر است. برای این پروژه مشابه تصویر ۷.۴ از وبسایت مورالیس^۷ برای پیدا کردن آدرس نود شبکه استفاده شد.



شکل ۷.۴: دریافت آدرس یکی از نودهای شبکه از وبسایت مورالیس

۲.۳.۴ اضافه شدن اطلاعات شبکه مورد نظر به تنظیمات ترافل

هنگامی که به کمک دستور `truffle init` یک پروژه ترافل ساخته می‌شود، فایلی با نام `truffle-config.js` ساخته می‌شود. تنظیمات مربوط به ترافل در این فایل نوشته شده است. برای این که ترافل شبکه مورد نظر را

⁷ <https://moralis.io>

بشناسد باید اطلاعات آن شبکه در این فایل نوشته و شبکه‌ی جدیدی تعریف شود. برای تعریف شبکه از آدرسی که در گام قبل به دست آمد استفاده می‌شود و مانند تصویر ۸.۴ شبکه‌ی جدیدی تعریف می‌شود.

```

38 networks: [
39   ropsten: {
40     provider: () =>
41       new HDWalletProvider(
42         mnemonic,
43         `https://speedy-nodes-nyc.moralis.io/91eaf4d08216c15dcf25aa72/eth/ropsten`
44       ),
45     network_id: 3, // Ropsten's id
46     gas: 8000000, // Ropsten has a lower block limit than mainnet
47     gasPrice: 10000000000,
48     confirmations: 2, // # of confs to wait between deployments. (default: 0)
49     timeoutBlocks: 200, // # of blocks before a deployment times out (minimum/default: 50)
50     skipDryRun: true, // Skip dry run before migrations? (default: false for public nets )
51   },
52 ]

```

شکل ۸.۴: اضافه کردن شبکه راپستان به شبکه‌های ترافل

۳.۳.۴ آماده شدن نمونیکز

برای انجام این پروژه به کمک دستور npm mnemonics مطابق تصویر ۹.۴ یک آدرس تستی ساخته می‌شود. این دستور، نمونیکز متناسب با این آدرس را به عنوان خروجی می‌دهد. دقت کنید که برای بارگذاری روی شبکه اصلی^۸ حتما باید از نمونیکز مربوط به یک کیف پول واقعی استفاده شود و اطلاعات آن در اختیار کسی قرار نگیرد.

```
→ cappu git:(main) ✘ npx mnemonics
flavor bleak joy tired bid habit regret prison nasty acoustic amount thought
```

شکل ۹.۴: ایجاد نمونیکز تستی

۴.۳.۴ استفاده از کیف پول ایجاد شده در تنظیمات ترافل

ترافل برای این که بتواند از کیف‌پول برای انجام تراکنش‌ها استفاده کند باید به نمونیکز یا کلید خصوصی آن دسترسی داشته باشد. به این منظور فایلی با نام secrets.json در دایرکتوری اصلی برنامه ساخته می‌شود و مطابق تصویر ۱۰.۴ نمونیکز کیف پول به شکل زیر در آن قرار داده می‌شود.

⁸Mainnets

```

1  {
2    "mnemonic": "sun seat live wealth pistol bubble ...",
3  }
4

```

شکل ۱۰.۴: قراردادن نمونیکز در فایل secrets.json

سپس در تنظیمات ترافل باید مطابق تصویر ۱۱.۴ ذکر شود که می‌تواند آدرس کیف‌پول را در این آدرس پیدا کند.

```

25 |
26   const mnemonic = require("./secrets.json").mnemonic;
27

```

شکل ۱۱.۴: معرفی فایل secrets.json در تنظیمات ترافل

۵.۳.۴ نصب کیف‌پول hdwallet

ترافل برای استفاده از نمونیکز کیف‌پول مانیاز به نصب پکیج hdwallet-provider دارد، این پکیج کاربری‌های یک کیف‌پول دیجیتال از جمله امضا و ارسال تراکنش بر روی شبکه بلاکچین را در اختیار ترافل قرار می‌دهد. این پکیج با اجرای دستور npm install –save-dev @truffle/hdwallet-provider نصب می‌شود. پس از نصب کیف‌پول در تنظیمات ترافل در فایل truffle-config.js مطابق با تصویر ۱۲.۴ ذکر می‌شود که از این کیف‌پول استفاده شود.

```

20
21   const HDWalletProvider = require("@truffle/hdwallet-provider");
22

```

شکل ۱۲.۴: استفاده از کیف‌پول hdwallet در تنظیمات ترافل

۶.۳.۴ انتخاب شبکه اضافه شده

حال هنگام ورود به خط فرمان ترافل مانند تصویر ۱۳.۴ شبکه مورد نظر انتخاب می‌شود.

```
→ back git:(main) ✘ truffle console --network ropsten
truffle(ropsten)> |
```

شکل ۱۳.۴: ورود به خط فرمان ترافل با انتخاب شبکه راپستن

۷.۳.۴ بررسی آدرس کیف‌پول و موجودی آن

برای بارگذاری یک قرارداد هوشمند باید آدرس بارگذاری کننده آن بتواند هزینه تراکنش بارگذاری را پرداخت کند. در صورتی که بارگذاری بر روی یک شبکه تستی انجام می‌شود باید با استفاده از یک faucet روی شبکه تستی به میزان کافی پول تستی دریافت شود.

برای دریافت آدرس‌های کیف‌پول مانند تصویر ۱۴.۴ از دستور `getAccounts` در خط فرمان ترافل استفاده می‌شود.

```
truffle(ropsten)> await web3.eth.getAccounts()
[
  '0xF51f5f41BfA8ADa57a43862cBc18dA4750AecB4c',
  '0x909ebC92395FC4335c35894C7DDc8bfFFDCeEF06',
  '0x48156708DF687C7a8F97C951b5E734E132e891D1',
  '0xF1C6c91D80032528e2C01F73DAd588D11DA0f17d',
  '0xA6f899d10B4E1c1195AFD1C6f29E4e539C828450',
  '0xB63191Dd13637c024C7F1F339F254F0F13d4bB34',
  '0x1699Ba468F7E5af64f510B323537bbcd107373F9',
  '0x6eDd855A6D2d3De5D96749e1bD3E9580c33468E7',
  '0x8A97C0bfC3086DFcd9E1B25D69A1A238A1290BE6',
  '0xc4838dF4d46862d1226BDC409EbE8395cA6fE703'
]
```

شکل ۱۴.۴: دریافت آدرس‌های کیف‌پول در خط فرمان ترافل

برای دریافت مانده حساب آدرس، مانند تصویر ۱۵.۴ از دستور `getBalance` در خط فرمان ترافل استفاده می‌شود.

```
truffle(ropsten)> await web3.eth.getBalance("0xF51f5f41BfA8ADa57a43862cBc18dA4750AecB4c")
'790887817599784390'
truffle(ropsten)>
```

شکل ۱۵.۴: دریافت موجودی کیف‌پول در خط فرمان ترافل

۸.۳.۴ بارگذاری قرارداد هوشمند روی شبکه بلاکچین

پس از اطمینان از توانایی پرداخت کارمزد تراکنش با استفاده از دستور `migrate` در خط فرمان ترافل قرارداد هوشمند روی شبکه بلاکچین بارگذاری می‌شود.

۹.۳.۴ اطمینان از صحبت بارگذاری قرارداد هوشمند

پس از اتمام بارگذاری قرارداد هوشمند برای اطمینان از به درستی انجام شدن فرآیند بارگذاری قرارداد، می‌توان از جستجوگرهای شبکه^۹ بلاکچین استفاده کرد. برای مثال قرارداد هوشمند کاپو بر روی شبکه راپستان بارگذاری شده است، که با رفتن به وبسایت اتراسکن^{۱۰} و قراردادن آن روی شبکه راپستان، مانند تصویر ۱۶.۴ می‌توان قرارداد بارگذاری شده و تراکنش‌های آن را مشاهده کرد.

The screenshot shows the Etherscan interface for the Ropsten Testnet Network. It displays the contract overview for the address 0xf1359760a1b37a9f8d85341E72cEF0644DB660A1. The contract name is 'Contract' and its balance is 0 Ether. The 'Transactions' tab shows the latest 13 transactions from a total of 13, all related to the minting of tokens. The transactions are listed with columns for Txn Hash, Method, Block, Age, From, To, Value, and Txn Fee. Each transaction row includes a small icon indicating the method (Safe Transfer or Mint) and the recipient's address.

Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0xd207a3f62e89fcfc054c...	Safe Transfer Fr...	12147200	7 days 19 hrs ago	0xf51f41bfa8ada57a43...	IN 0xf1359760a1b37a9f8d8...	0 Ether	0.000352610001
0xeca8b0c2be5bd13534...	Mint	12147179	7 days 19 hrs ago	0xf51f41bfa8ada57a43...	IN 0xf1359760a1b37a9f8d8...	0 Ether	0.021437500002
0xd754232c700bd4cb75...	Mint	12147178	7 days 19 hrs ago	0xf51f41bfa8ada57a43...	IN 0xf1359760a1b37a9f8d8...	0 Ether	0.000273017508

شکل ۱۶.۴: مشاهده قرارداد کاپو در اتراسکن روی شبکه راپستان

۴.۴ توسعه واسط کاربری، اتصال به قرارداد هوشمند و فرآیند بارگذاری

برای توسعه واسط کاربری اپلیکیشن، React به عنوان چارچوب مورد استفاده انتخاب شد. ترکیب این چارچوب با استفاده از کتابخانه material-ui کمک می‌کند در زمان کوتاه بتوان ظاهری زیبا و یکدست در

⁹Block Explorers

¹⁰<https://etherscan.io>

اپلیکیشن ایجاد کرد و کتابخانه Web3JS که بواسطه کاربری را به کیف پول کاربر و شبکه بلاکچین متصل می‌کند، همه‌ی قابلیت‌های مورد نیاز برای توسعه یک بواسطه کاربری زیبا و کارآمد را در اختیار توسعه دهنده قرار می‌دهد. در پوشه اصلی بواسطه کاربری فایلی با عنوان config.js وجود دارد. در این فایل علاوه بر ABI قرارداد هوشمند سایر اطلاعات مورد نیاز مانند آدرس شبکه، آدرس قرارداد در شبکه و نام شبکه مورد نظر نیز ذخیره می‌شود. هنگام توسعه باید دقیق شود که این فایل به قرارداد روی شبکه محلی متصل شود.

برای استفاده از Web3JS و اتصال به کیف پول کاربر یک فایل به نام connect.js ساخته شد، تمامی اعمال ارتباطی با کیف پول کاربر به عنوان چند تابع در این فایل جمع آوری شده‌اند، این فایل به صورت یک آداتور میان Web3JS و کد کاپو عمل می‌کند. تمامی قابلیت‌های مورد نیاز مانند اتصال به کیف پول و ورود^{۱۱} کاربر، خروج^{۱۲} کاربر، گرفتن آدرس و شبکه‌ی کیف پول و ... در این فایل انجام می‌شود.

واسطه کاربری کاپو پس از تایید کاربر و دریافت آدرس کیف پول او، آن را در sessionStorage ذخیره می‌کند، از این طریق متوجه می‌شود که آیا کاربر وارد شده است یا خیر و با چه آدرسی. کاپو پیش از اتصال به کیف پول کاربر چک می‌کند که کیف پول روی شبکه یکسانی با شبکه فعلی کاپو باشد و در غیر این صورت به کاربر هشدار می‌دهد. همچنین در واسطه کاربری کاپو برای داشتن تجربه کاربری بهتر تلاش شده است. نکاتی مانند عدم نمایش قابلیت‌هایی مانند ساخت و ارسال توکن هنگامی که کیف پول کاربر به اپلیکیشن متصل نیست، جایه‌جایی آسان میان صفحات به کمک react-router، طراحی responsive برای رایانه و گوشی موبایل، نمایش alert ها و error های مناسب به کاربر، نمایش loading هنگامی که تراکنش‌ها در حالت pending هستند و نمایش پیام‌های مناسب با توجه به نتیجه تراکنش‌های کاربر.

برای این که کاربرها بتوانند با قرارداد هوشمند ارتباط برقرار کنند نیاز است که واسطه کاربری اپلیکیشن در سروری بارگذاری شود. خوشبختانه گیت‌هاب قابلیت به نام Github Pages در اختیار کاربرانش قرار می‌دهد که به کمک آن می‌توان واسطه کاربری اپلیکیشن را در آدرسی متناسب با آدرس مخزن کد در گیت‌هاب بارگذاری کرد و کاربران با رجوع به آن آدرس می‌توانند واسطه کاربری اپلیکیشن را ببینند و از آن استفاده کنند.

این قابلیت گیت‌هاب در واقع به این صورت عمل می‌کند که یک برنج به نام repository gh-pages در پروژه می‌سازد و هر بار که دستور بارگذاری پروژه توسط گیت‌هاب اجرا می‌شود، یک بیلد از پروژه می‌گیرد و فایل‌های خروجی بیلد روی این برنج پوش می‌شوند. سپس این فایل‌ها روی آدرسی متناسب با آدرس repository بارگذاری

¹¹Login¹²Logout

می‌شوند. برای مثال آدرس ریپازیتوری و واسط کاربری اپلیکیشن کاپو به صورت زیر است:

- آدرس ریپازیتوری: <https://github.com/bshramin/cappu>

- آدرس واسط کاربری: <https://bshramin.github.io/cappu>

البته بارگذاری شدن واسط کاربری روی Github Pages با ایجاد مشکلاتی در routing همراه بود که رفع شدند.

۵.۴ داکرایز شدن، پایپلاین‌ها و گیت

اقدامات زیر به منظور سرعت بخشیدن و تسهیل فرآیندهای توسعه و بارگذاری انجام شدند.

۱.۵.۴ داکرایز شدن تست‌های قرارداد هوشمند

برای سرعت بخشیدن به توسعه قرارداد هوشمند، این نیازمندی به وجود آمد که بعد از پوش شدن هر تغییر روی گیت‌هاب تست‌های قرارداد به صورت خودکار اجرا شوند. به این منظور پیش از هر چیز تست‌های قرارداد هوشمند باید بتوانند به صورت داکرایز اجرا شوند.

برای داکرایز کردن اجرای تست‌های قرارداد هوشمند، اول سعی در این بود که یک ایمیج داکر پایه که ترافل روی آن نصب شده باشد پیدا شود، اما نسخه ترافل نمونه‌هایی که یافت شد با نسخه مورد نظر همخوانی نداشت. در نتیجه یک ایمیج پایه داکر نوشته شد که داکرفایل آن را می‌توان در گیت‌هاب^{۱۳} مشاهده کرد، همچنین این ایمیج داکر در داکرهاب^{۱۴} نیز پوش شد.

سپس داکرفایل دیگری نوشته شد که با استفاده از این ایمیج پایه تست‌های قرارداد را اجرا کند. تست‌های قرارداد در این ایمیج که ترافل بر روی آن نصب شده است با اجرای دستور `truffle test` اجرا می‌شود.

¹³ <https://github.com/bshramin/truffle-docker>

¹⁴ <https://hub.docker.com/r/aminbshir/truffle>

۲.۵.۴ اجرای خودکار تست‌های قرارداد

با داشتن داکرفایلی که با بیلد و اجرای آن تست‌های قرارداد هوشمند اجرا می‌شوند، تست‌های قرارداد هوشمند می‌توانند به عنوان یکی از مراحل پایپلاین پروژه در گیت‌هاب نیز اجرا گردند. به این صورت در هر مرحله ریکوئست به برج master و با پوشش‌شدن یک کامیت در برج master تست‌ها به صورت خودکار در پایپلاین گیت‌هاب اجرا می‌شوند. به این ترتیب سرعت توسعه و اطمینان از کدهای قرارداد بیشتر می‌شود.

۳.۵.۴ بارگذاری خودکار واسط کاربری

برای ساده‌سازی بیشتر فرآیند بارگذاری واسط کاربری و سرعت بخسیدن به توسعه آن، این قابلیت پیاده‌سازی می‌شود که پس از هر بار ایجاد تغییر در واسط کاربری، به جای این که توسعه‌دهنده با اجرای دستوراتی واسط کاربری را به کمک صفحات گیت‌هاب بارگذاری کند، واسط کاربری پس از پوشش‌شدن تغییرات جدید روی برج اصلی ریپازیتوری بارگذاری می‌شود.

برای پیاده‌سازی این قابلیت از Github Actions که در واقع پایپلاین‌های گیت‌هاب برای یک پروژه هستند استفاده می‌شود. تنها نکته‌ای که باید به آن توجه شود این است که این استیج از پایپلاین یک تفاوت اصلی با استیج‌های دیگر دارد. استیج‌های دیگر فقط می‌خواهند که کدهای ریپازیتوری را بخوانند و نمی‌خواهند چیزی را در ریپازیتوری تغییر دهند، اما این استیج می‌خواهد که کدهای واسط کاربری را بیلد کند و سپس فایل‌های بیلد شده را روی برج دیگری به نام gh-pages پوش کند. پس این استیج پایپلاین نیاز به دسترسی پوش کردن کد روی ریپازیتوری دارد.

برای پیاده‌سازی این قابلیت به این صورت عمل می‌شود که نخست یک داکرفایل نوشته می‌شود که در آن کدهای واسط کاربری بیلد و سپس به کمک صفحات گیت‌هاب روی برج gh-pages پوش و بارگذاری می‌شوند. اما این کانتینر برای این که بتواند کدها را روی ریپازیتوری پوش کند نیاز به یک توکن از گیت‌هاب دارد، به همین دلیل برای این داکرفایل یک ENV تعریف می‌شود و هنگامی که در استیج بارگذاری واسط کاربری این داکرفایل بیلد و اجرا می‌شود توکنی که از گیت‌هاب گرفته شده است به عنوان env به این کانتینر داده می‌شود. به این ترتیب این توکن درون کانتینر داکر وجود خواهد داشت و Github Pages از آن استفاده خواهد کرد.

فصل ۵

دست آوردها، پیشنهادها، محدودیت‌ها

۱.۵ دست آوردها

۱.۱.۵ پلتفرم ایجاد شده

قرارداد هوشمند نوشته شده در این پروژه، کاپو، با عملکرد کامل بر بروی شبکه تستی راپستان بارگذاری شد و امکانات لازم برای دسترسی عموم مردم به روشی آسان و ارزان به توکن‌های تعویض‌ناپذیر را فراهم می‌کند. کاربران می‌توانند در صفحه اصلی این اپلیکیشن تعداد توکن‌های ساخته شده و تعداد آدرس‌های دارای توکن را مشاهده کنند. سپس با متصل کردن کیف‌پولشان به اپلیکیشن می‌توانند توکن بسازند، دارایی‌هایشان را مشاهده کنند و توکن‌هایشان را به دیگران ارسال کنند.

۲.۱.۵ ساخت محیط توسعه سریع و خودکار

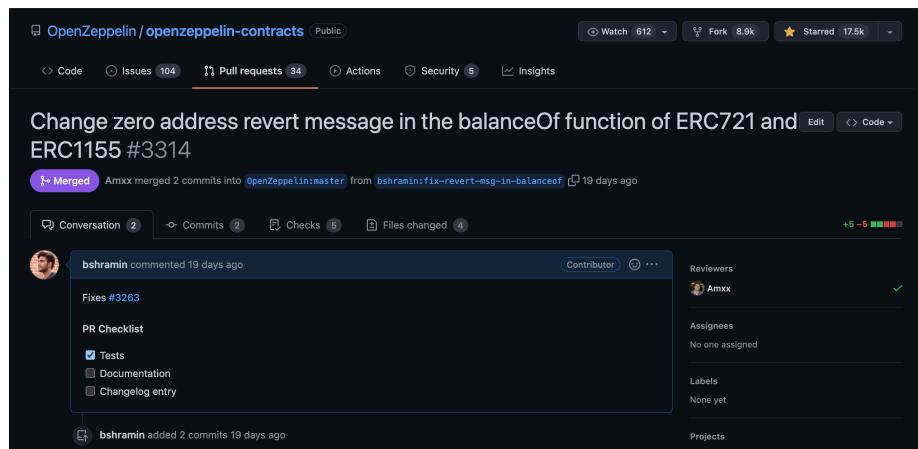
سپس آموختیم که چگونه اجرای تست‌های قرارداد هوشمند را داکرایز و به صورت خودکار در پایپ‌لاین پروژه اجرا کنیم. برای انجام این کار یک داکر ایمیج ترافل نوشته شد، که آن به صورت متن باز بر روی گیت‌هاب بارگزاری و ایمیج آن به داکرهاب اضافه شد. سپس واسط کاربری اپلیکیشن داکرایز شد و از env های داکر برای فرستادن توکن گیت‌هاب از پایپ‌لاین به کانتینر استفاده شد و در نتیجه واسط کاربری اپلیکیشن به صورت خودکار در پایپ‌لاین

پروژه روی صفحات گیت‌هاب بارگزاری می‌شود.

در نتیجه انجام این کارها یک مسیر راحت و سریع برای توسعه یک قرارداد هوشمند به همراه واسط کاربری ایجاد شد که تست‌ها و فرایند بارگذاری همه به صورت خودکار در آن اجرا می‌شوند.

۳.۱.۵ ورود به جامعه توسعه‌دهندگان

برای انجام این پروژه از قراردادهای هوشمند اپن‌زپلین استفاده شد. اپن‌زپلین یکی از پر استفاده‌ترین مخزن استانداردهای قراردادهای هوشمند است که در زمان نوشته شدن این متن در گیت‌هاب بیش از هفده هزار و پانصد ستاره و نزدیک به نه هزار فورک دارد. در حین توسعه کاپو تغییراتی در پروژه اپن‌زپلین نیز اعمال گردید. سپس یک مرج ریکوئست^۱ شامل این تغییرات ساخته شد که در همان روز توسط تیم اپن‌زپلین به مخزن کد اصلی اضافه شد. این مرج ریکوئست در تصویر ۱.۵ قابل مشاهده است. این موضوع برتری‌های توسعه متن‌باز را نشان می‌دهد. تیم‌های توسعه بزرگ، حتی به بزرگی پروژه‌ای مانند اپن‌زپلین، از کمک همه توسعه دهندگان استقبال می‌کنند، حتی کسانی که برای اولین بار می‌خواهند روی یک پروژه تغییری ایجاد کنند.



شکل ۱.۵: مرج ریکوئست باز شده بر روی اپن‌زپلین

¹Merge Requests

۴.۱.۵ یادگیری

در طی انجام این پروژه با ابزارها، چارچوب‌ها، کتابخانه‌ها و استانداردهای توسعه قراردادهای هوشمند آشنا شدیم. آموختیم که چارچوب ترافل چه ابزارهایی را در اختیار توسعه دهنده قرار می‌دهد. چگونه می‌توان یک شبکه محلی برای توسعه ایجاد کرد، قرارداد هوشمند را بر روی آن بارگذاری کرد و واسط کاربری و کیف پول را به آن متصل کرد.

آموختیم که چگونه می‌توانیم برای پیاده‌سازی قراردادهای هوشمند از استانداردهای موجود استفاده کنیم، برای آن‌ها تست بنویسیم و به کمک چارچوب ترافل این تست‌ها را اجرا کنیم. آموختیم که چگونه پس از اتمام فرآیند توسعه قرارداد هوشمند را بر روی شبکه تستی بارگذاری کنیم. همچنین واسط کاربری اپلیکیشن به کمک صفحات گیت‌هاب بارگذاری و به قرارداد هوشمند روی شبکه تست متصل شد.

۲.۵ پیشنهادها

در این قسمت با توجه به آموخته‌هایی که در طی انجام این پروژه به دست آمد، پیشنهادهایی برای توسعه پروژه‌های مشابه ذکر می‌شود. امید است که استفاده از این پیشنهادها مسیر توسعه را هموارتر کرده و سرعت بینشند.

۱۰.۲.۵ استفاده از استانداردها

خوبختانه در این پروژه از ابتدا به استفاده از استانداردهای موجود اهمیت داده شد. در صورتی که توسعه دهنده بخواهد از استانداردهای موجود استفاده نکند مزیت سازگاری قرارداد هوشمند نوشته شده با پلتفرم‌هایی که از پیش وجود دارند را از دست می‌دهد.

همچنین در صورتی که توسعه دهنده تصمیم بگیرد که از یکی از استانداردها پیروی کند بهتر است که از پیاده‌سازی‌های موجود به صورت متن‌باز استفاده کند، این تصمیم باعث رشد چشمگیر سرعت توسعه قرارداد هوشمند می‌شود، امکان وجود خطا و مشکل امنیتی در قرارداد را کم و امکان دریافت آپدیت‌های جدید را تسهیل می‌کند.

۲.۲.۵ استفاده از ERC721 به جای ERC1155

استاندارد ERC1155 برتری‌های فراوانی نسبت به استاندارد ERC721 دارد. از جمله این برتری‌ها می‌توان به قابلیت ارسال چند توکن در یک تراکنش، توانایی ساخت انواع مختلف توکن با تعداد متفاوت و پشتیبانی از توکن‌های تعویض‌پذیر و تعویض‌ناپذیر به صورت همزمان اشاره کرد. با توجه به قابلیت ارسال همزمان چند توکن و یا ساخت همزمان چندین توکن در یک تراکنش، هزینه‌ی پرداختی کاربرها نیز برای استفاده از قرارداد هوشمند به نحو شایانی کاهش می‌یابد و به این نحو قرارداد در دسترس جامعه بزرگتری قرار می‌گیرد.

۳.۲.۵ ساخت محیط توسعه از شروع کار

انجام مواردی مانند خودکار سازی اجرا شدن تست‌ها، بارگذاری رابط کاربری و استفاده از ابزارهای کنترل ورژن مانند گیت‌هاب گرچه در شروع کار ممکن است خسته‌کننده باشند و به توسعه‌دهنده حس پیشرفت در انجام پروژه را ندهند، اما این کارها هرچه زودتر و در شروع پروژه انجام شوند سرعت پیشرفت پروژه را دو چندان می‌کنند. در نتیجه پیشنهاد می‌شود که در نقطه شروع پروژه به روند توسعه توجه شود و زمانی به بهینه‌سازی این روند اختصاص داده شود.

۳.۵ محدودیت‌ها

۱.۳.۵ استفاده از دریزل

در ابتدای انجام پروژه سعی شد که برای برقراری ارتباط رابط کاربری با قرارداد هوشمند از دریزل استفاده شود. اما استفاده از این ابزار مشکلات فراوانی را به همراه داشت. با توجه به تازگی و بالغ نبودن ابزارهای موجود برای توسعه قراردادهای هوشمند، باید سعی شود که تا جای ممکن از ابزارهای پراستفاده و با جامعه توسعه‌دهندگان بزرگ استفاده شود. در این پروژه پس از مواجهه با محدودیت‌های فراوان دریزل، از کتابخانه Web3JS استفاده شد که دست توسعه دهنده را به میزان خوبی باز می‌گذارد.

۲.۳.۵ ورژن‌های مختلف ابزارها

عدم همخوانی نسخه‌های مختلف ابزارهای مورد استفاده با یکدیگر مشکلات زیادی در توسعه پروژه ایجاد کرد. در هنگام توسعه پروژه باید حتماً دقت شود که برای هر ابزار در حال استفاده از چه نسخه‌ای هستیم. همچنین برای محدود کردن این مشکل پیشنهاد می‌شود از ابزارهای کانتینر کننده مانند Docker استفاده شود.

۳.۳.۵ هزینه تراکنش‌های شبکه اصلی اتریوم

در زمان نوشته شدن این متن، هزینه انجام تراکنش روی شبکه اصلی اتریوم به شدت بالاست. این هزینه‌ی بالا باعث می‌شود که بارگذاری کردن روی شبکه اصلی برای یک پروژه آزمایش از دسترس دور باشد. اگرچه ممکن است در آینده با بروزرسانی اتریوم ۲ این هزینه به شدت کاهش یابد.

۴.۳.۵ عدم وجود راهنمای مستندات کافی

تازگی این زمینه باعث عدم وجود راهنمای مستندات کافی شده است. این موضوع از دیگر دلایل پیشنهاد به استفاده از ابزارهای با جامعه توسعه‌دهنگان بزرگ است.

مراجع

- [1] Foundation, Ethereum. Ethereum documentation. . Accessed: 2010-09-30.
- [2] Foundation, Zeppelin. Ethereum documentation. . Accessed: 2010-09-30.

Abstract

Web3 is the next step in the evolution of the internet, the most critical innovation after the word wide web. Increasingly, applications and financial transactions will be able to be decentralized with the advent of cryptocurrencies, consensus algorithms, and smart contracts.

Absolute ownership of assets is one of the main advantages of a decentralized internet. It is so that no one person or organization can control another's assets or prevent them from being transferred. The blockchain will also protect the copyright of digital assets. Even if someone copies an artist's digital art, it is still evident to everyone that the artist owns the image.

With the benefits of ownership they provide, non-fungible tokens became very popular. The ease and convenience of minting, keeping, selling, and transferring NFTs have also encouraged the widespread use of these tokens. Many technologies and platforms have been introduced and are still being developed for this new and huge market.

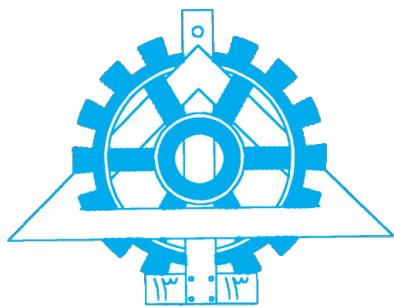
With this project, we are developing an easy-to-use platform so that any person or organization can mint and transfer NFTs. The use cases for such a platform are endless. In Cappu, assets such as cars, houses, movie tickets, and doctor appointments can be minted, sold, and transferred to others as an NFT.

Even though this platform may have a wide variety of uses, the primary objective of this project is to become familiar with the process and tools of developing and deploying a smart contract, development and connections of the front end of the smart contract, and learn about the standards of building a smart-contract, such as ERC721 and ERC1155.

Moreover, Cappu is open-source, and the code can be found on Github².

Keywords Cappu, NFT, smart-contract, solidity, ERC721

²<https://github.com/bshramin/cappu>



**University of Tehran
College of Engineering
Faculty of Electrical and
Computer Engineering
Software Department**



Cappu, a platform to mint and transfer data NFTs.

A Thesis submitted to the Undergraduate Studies Office
In partial fulfillment of the requirements for
The degree of Bachelor of Science
in Computer Engineering - Software Engineering

By:
Amin Bashiri

Supervisor:
Dr. Ehsan Khamespanah

June 2022