

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. همچنین 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 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 این بازار تازه و بزرگ، تکنولوژی‌ها، استانداردها و پلتفرم‌های زیادی ساخته شده‌اند و همچنان نیز در حال توسعه هستند.

هدف این پروژه ساخت پلتفرمی است 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 ساده‌بی‌شمار 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، آزمون و بارگذاری یک قرارداد هوشمند، ساخت واسطه کاربری، اتصال آن به قرارداد ERC1155 و همچنین شناخت استانداردهای معروف قراردادهای توکن‌های تعویض‌ناپذیر مانند ERC721 و ERC155 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 متامسک ^{۱۰}
۱۵	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 .
۲۷	۲.۴ نوشتن و اجرای آزمون‌ها .
۲۸	۳.۴ بارگذاری قرارداد روی شبکه آزمایشی راپستان ^{۱۱} .
۳۰	۱.۳.۴ یافتن آدرس یکی از ندهای 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 ^{۱۲} .
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 .
۳۲	۷.۳.۴ بررسی آدرس کیف‌پول و موجودی آن .
۳۳	۸.۳.۴ بارگذاری قرارداد هوشمند روی شبکه بلاکچین .
۳۴	۹.۳.۴ اطمینان از صحت بارگذاری قرارداد هوشمند .

¹⁰Metamask¹¹Ropsten¹²Mnemonics

۴۴	توسعه واسطکاربری	۴.۴
۱.۴.۴	اتصال به کیف پول و قرارداد کاپو	۱.۴.۴
۲.۴.۴	تجربه کاربری	۲.۴.۴
۳.۴.۴	بارگذاری واسطکاربری	۳.۴.۴
۵.۴	داکرایز شدن، پایپلاین‌ها و گیت	۵.۴
۱.۵.۴	داکرایز شدن آزمون‌های قرارداد هوشمند	۱.۵.۴
۲.۵.۴	اجرای خودکار آزمون‌های قرارداد	۲.۵.۴
۳.۵.۴	بارگذاری خودکار واسطکاربری	۳.۵.۴
فصل ۵: دستآوردها، پیشنهادها، محدودیت‌ها		
۱.۵	دستآوردها	۱.۵
۱.۱.۵	پلتفرم ایجاد شده	۱.۱.۵
۲.۱.۵	ساخت محیط توسعه سریع و خودکار	۲.۱.۵
۳.۱.۵	ورود به جامعه توسعه‌دهندگان ^{۱۳}	۳.۱.۵
۴.۱.۵	یادگیری	۴.۱.۵
۲.۰.۵	پیشنهادها	۲.۰.۵
۱.۲.۵	استفاده از استانداردها	۱.۲.۵
۲.۲.۵	استفاده از ERC1155 به جای ERC721	۲.۲.۵
۳.۲.۵	ساخت محیط توسعه از شروع کار	۳.۲.۵
۳.۰.۵	محدودیت‌ها	۳.۰.۵
۱.۳.۵	استفاده از دریزل ^{۱۴}	۱.۳.۵
۲.۳.۵	ورژن‌های مختلف ابزارها	۲.۳.۵
۳.۳.۵	هزینه تراکنش‌های شبکه اصلی اتریوم	۳.۳.۵
۴.۳.۵	عدم وجود راهنمای مستندات کافی	۴.۳.۵

¹³Community¹⁴Drizzle

مراجع

۵۱

ت

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 آزمون 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 پلتفرم توسعه یافته در این پروژه، کاپو، تا جای ممکن مطابق آن‌ها توسعه می‌یابد. برای قرارداد هوشمند نوشته شده آزمون می‌نویسیم و آن را روی شبکه آزمایشی^۵ انتشار می‌دهیم. در گام بعد برای پلتفرم، واسطکاربری 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 آزمون‌های قرارداد هوشمند، یک ایمیج داکر^۸ ترافل^۹ نوشته می‌شود. در قدم بعد هر دو بخش واسطکاربری و قرارداد هوشمند داکرایز می‌شوند و فرایند اجرای آزمون‌های قرارداد هوشمند و بارگذاری شدن واسطکاربری 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 هواپیما، 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 ابزارهای 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 آزمون‌ها و اجرای خودکار آزمون‌ها، آسان و خودکار بودن روند دیپلوی و اتصال بی‌دردسر رابط کاربری^{۱۰} به قرارداد هوشمند است.

¹⁰User Interface

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 دوم مفاهیم اولیه توسعه قرارداد هوشمند بر بستر بلاکچین، کاربردها، مفاهیم و استانداردها توضیح داده می‌شود. در فصل سوم ابزارهای توسعه قراردادهای هوشمند معرفی می‌شوند، مزایا و معایب هر یک بیان شده و نحوه استفاده از آن‌ها توضیح داده می‌شود. در فصل چهارم 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

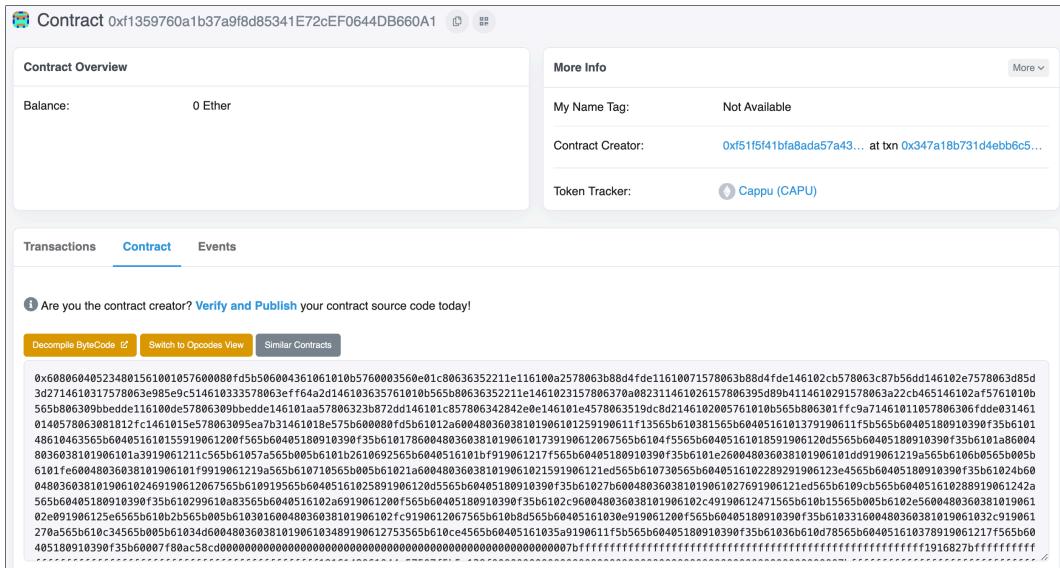
دلایل زیادی برای متن باز نوشتگان قراردادهای هوشمند وجود دارد، در ادامه تعدادی از این دلایل توضیح داده می شود.

دلیل اول، بلاکچین ها محرمانگی^۱ ندارند. همه نُدهای 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. 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: بایت کدهای قرارداد کاپو

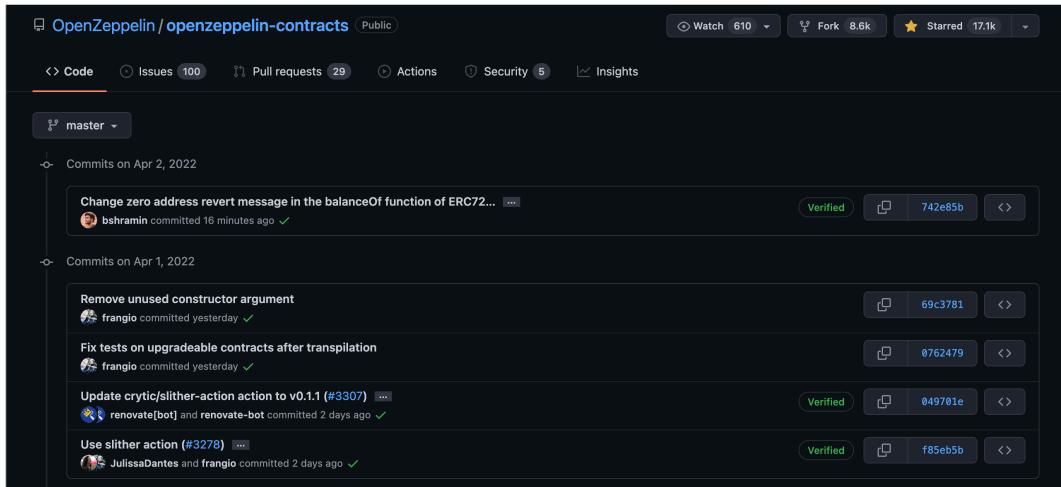
دلیل دوم، اصلی ترین مزیت اپلیکیشن های غیر مرکز نسبت به اپلیکیشن های مرکز عدم نیاز به اعتماد است. کاربرها می توانند کدهای قرارداد هوشمند را بخوانند و به کد نوشته شده اعتماد کنند. در حالی که اگر کد برنامه برای همه کاربران قابل مشاهده نباشد کاربرها باید به سازندگان آن برنامه اعتماد کنند.

دلیل سوم، بارگذاری کردن قراردادهای هوشمند معمولاً آسان نیست و سرعت تغییرات پایین تر از اپلیکیشن های 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 می شود. متن باز نوشتن قرارداد هوشمند باعث می شود چشم های بیشتری کدهای قرارداد را بخوانند و مشکلات احتمالی سریعتر مشخص و رفع شوند. تعداد زیادی از پروژه ها کاپو از همان روز اول قرارداد هوشمند را به صورت متن باز توسعه می دهند، بعضی نیز ترجیح می دهند که پروژه به مرحله ای از توسعه بررسد و سپس آن را متن باز می کنند.

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 نشان دهنده سرعت پیشرفت بسیار بالاست، این واقعیت را نیز نشان می دهد که در یک جامعه متن باز هر توسعه دهنده می تواند به پیشرفت جامعه به هر شکلی که می پسندد کمک کند، اشکالاتی که مشاهده می کند را گزارش دهد یا تصحیح کند.

⁴OpenZeppelin

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: اعمال تغییرات انجام شده روی پروژه بزرگ اپن‌زپلین

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

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

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.

در فصل دهم کتاب Mastering Ethereum [۱] انواع توکن‌ها و مفاهیم تعویض‌پذیری به تفصیل توضیح داده شده‌اند.

۳.۲ 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 یا به فروش گذاشت [۹]. 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. همچنین 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^۹ 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 [۳] مطالعه کرد.

برای استفاده از این استانداردها از بسته‌های متربازی استفاده می‌شود که این استانداردها را پیاده‌سازی کرده‌اند و از آن‌ها در قرارداد مورد نظر ارجحیت برخی می‌شود. یکی از بهترین پیاده‌سازی‌های این استانداردها توسط اپن‌زپلین [۱۰] 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 متامسک^{۱۰} هایی برای تعریف حداکثر تعداد توکن‌های موجود، گرفتن موجودی یک آدرس، و انتقال توکن‌ها دارد. توضیحات دقیق‌تر در مورد این استاندارد را

⁹Interface
¹⁰Method

می‌توان در وبسایت اتریوم^{۱۱} یا اپن‌زپلین^{۱۲} مشاهده کرد.

۲.۴.۲ استاندارد ERC721

استفاده از استاندارد ERC721 برای توکن‌های تعویض‌ناپذیر بسیار مرسوم است. در این استاندارد تابع‌ها و رخداد^{۱۳} هایی برای یکسان سازی 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 تابعی 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^{۱۴} یا اپن‌زپلین^{۱۵} 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.

¹¹<https://ethereum.org/en/developers/docs/standards/tokens/erc-20>

¹²<https://docs.openzeppelin.com/contracts/4.x/api/token/erc20>

¹³Event

¹⁴<https://ethereum.org/en/developers/docs/standards/tokens/erc-721>

¹⁵<https://docs.openzeppelin.com/contracts/4.x/api/token/erc721>

- 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 می‌تواند یک بازی مثل مونوپولی^{۱۶} باشد که در آن هر کاربر مقداری پول دارد که در واقع یک توکن تعویض‌پذیر هست، به عنوان دارایی چند خانه دارد که به عنوان توکن‌های تعویض‌ناپذیری هستند که از هر کدام 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 بر طرف می‌کند. همه‌ی این چند نوع توکن می‌توانند همزمان در یک قرارداد هوشمند وجود داشته باشند.

در این استاندارد تابع‌هایی برای تعریف نوعی توکن با تعداد مشخص وجود دارد. اگر نیاز به توکنی تعویض‌ناپذیر باشد تعداد آن یک قرارداده می‌شود. همچنین تابع‌هایی 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^{۱۷} یا اپن‌زپلین^{۱۸} مشاهده کرد.

¹⁶Monopoly

¹⁷<https://ethereum.org/en/developers/docs/standards/tokens/erc-1155>

¹⁸<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 در گیت‌هاب ذکر شده است. لیست کاملی از این ابزارها با جزئیات بیشتر را میتوان در کتاب [1] Mastering Ethereum در ضمیمه D مشاهده کرد.

۱.۳ ابزارهای ساده

• ویرایشگر

برای برنامه نویسی این قرارداد هوشمند از ویرایشگر VSCode با نصب پلاگین مربوط به سالیدیتی^۱ 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 است، برای توسعه یک پروژه به صورت متن‌باز

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

اولین ابزار مورد نیاز یک برنامه ورژن کنترل است که نسخه‌های متفاوت و تغییر یافته کدها را به صورت مرتب نگهداری کند. برای این منظور از گیت‌هاب^۲ استفاده شده است.

• بسته‌های NPM و Node

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 بسته‌های 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 مرکز 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 برنامه را می‌گیرد. در صورت تایید کاربر آدرس کیف پول به کاپو داده می‌شود.

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^۳ کاربر آن تراکنش را امضا کند. متامسک از کاربر تایید تراکنش را می‌گیرد، امضا را انجام می‌دهد و تراکنش به شبکه بلاکچین ارسال می‌شود.

جزئیات بیشتر این فرآیندها در مستندات متامسک [۷] در دسترس است.

²<https://github.com/>

³Private key

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 هنوز به بلوغ کامل نرسیده‌اند. اما با توجه به متن باز بودن اکثر ابزارها، چارچوب‌ها و کتابخانه‌های توسعه اپلیکیشن‌های غیرمت مرکز، معمولاً سرعت رشد و تکامل بالایی دارند و به کمک توسعه‌دهندگان^۴ این حوزه، هر روز نسبت به روز گذشته پیشرفت بیشتری می‌کنند.

برای توسعه این پروژه از چارچوب ترافل^۵، کتابخانه‌ی اپن‌زپلین^۶ و کتابخانه‌ی Web3JS^۷ 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، آزمودن، بارگذاری و ماイگریشن^۸ قراردادهای هوشمند به زبان سالیدیتی را فراهم می‌کند. مستندات ترافل [۱۳] توضیحات کاملی در مورد نحوه استفاده از این چارچوب ارائه می‌کند. پس از نصب این ابزار با اجرای دستور truffle init مانند تصویر ۱.۳ می‌توان یک پروژه جدید ترافل ساخت. همچنین می‌توان با اجرای 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

⁴Developers

⁵<https://trufflesuite.com>

⁶<https://openzeppelin.com/contracts>

⁷<https://github.com/ChainSafe/web3.js>

⁸Migration

پس از ساخت پروژه با اجرای دستور `truffle console` مانند تصویر ۲.۳ و یا `truffle develop` می‌توان وارد خط فرمان ترافل شد.

```
→ back git:(main) ✘ truffle develop
Truffle Develop started at http://127.0.0.1:9545/

Accounts:
(0) 0x00eae7f45de5837119f5dd65fe63e58dcf8f7138
(1) 0x09ef5651770dcbb33d926a1b75e10b2944924736
(2) 0x3e6cfcfc6153d6dd9e2654afc8cfda499818c3e9
(3) 0xcefd7ce63b03aaccacd420828abe96acbe968d7
(4) 0xa84ab5be39670309d305e7aad3bcc347e75e9537
(5) 0x318913b83fe0c2d63ab29bb84e39b6e39ccdf93ef
(6) 0xde2602e64a047482e4606af26c89abb97afe6bdf
(7) 0x4d85fb20085db61ab33e8159bfe88f3e1a7a1279
(8) 0xa031538284a6910cf832e25b2ec8105f52a87b13
(9) 0x3973a0026d8e807d0b9b5fb91c35a16ea990063a

Private Keys:
(0) 9453db0e1f3c1034f80a01b335b141e0d519f21aced3dbb7d2da54f37d773a6d
(1) bf85b2b427c669793da84d9c98e78d0fabd8676938d5c87cc01de50d42e235b7
(2) eff37f33a88597e7434be9def2e22594047c678a451a7f9581e8378a839e19c0
(3) fdd92e2593bb15465bb7d6b61b3894c9fe50acf4ec60327ec7c1c2f378664dc8
(4) 0c50a6d545d747b33ab3744321eed3e212400c4cd9497a0211fe4bd729a90c9f
(5) 28f97ce0d95990b234c50438ee46fd27461cf2ecf429131c9aa4176c1c05a0ab
(6) e6eb577c2aa69993aeab80ad275c346c6cf9b8506e8ab29cc69234744593a622
(7) 039c1150cf5eb82756320f0c904035cf0017da2a87c4c1d2dd057a39b3f0c452
(8) a78d582b2fad08f58f59680de3de3e3e8a9e609fe101c8df850e5444c1b59c0
(9) 77e77e250d6551b2d7af2b2088e350e84577ca39deb66c41136ab7a90e69f76

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 آزمون‌ها، 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 توسعه‌دهنده 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^۹ و دریزل است.

۲.۳.۳ کتابخانه اپن زپلین

یکی از معروف‌ترین کتابخانه‌های قراردادهای هوشمند و استانداردهایشان است. قراردادها و استانداردهای موجود در این کتابخانه کاملاً آزمون شده، داکیومنت شده، ایمن و پایه بسیاری از قراردادهای هوشمند بر بستر

⁹Ganache

بلاکچین هستند. استانداردهای ذکر شده در این متن مانند، ERC20، ERC721 و ERC1155 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 اپن‌زپلین در قدم اول باید این کتابخانه به کمک دستور `npm install @openzeppelin/contracts` 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 اپن‌زپلین 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 اپن‌زپلین

۳.۳.۳ کتابخانه Web3JS

تراکنش‌های با یک قرارداد هوشمند می‌تواند به دو حالت باشد. در حالت اول فقط اطلاعات شبکه بلاکچین خوانده می‌شود و در حالت^{۱۰} آن تغییری داده نمی‌شود. تابع‌های از این جنس از نوع view یا pure هستند. حالت دوم تراکنش‌هایی هستند که باعث تغییر اطلاعات شبکه بلاکچین می‌شوند.

واسطه‌کاربری یک اپلیکیشن غیرمت مرکز برای انجام نوع اول تراکنش‌ها نهایتاً فقط به آدرس کاربر نیاز دارد که اطلاعات مربوط به او را از قرارداد دریافت کند. در حالت دوم نیاز است که تراکنشی بر روی شبکه ثبت شود. این کار نیازمند امضاشدن تراکنش توسط کلید خصوصی کاربر، پرداخت کارمزد تراکنش و ارسال آن به نُدهای شبکه است.

کتابخانه‌ی Web3JS به توسعه‌دهنده کمک می‌کند که واسطه‌کاربری اپلیکیشن را به کیف پول دیجیتال کاربر و شبکه بلاکچین متصل کند. با ایجاد این اتصال آدرس کاربر توسط کیف پول دیجیتال در اختیار واسطه‌کاربری

¹⁰State

قرار می‌گیرد. هرگاه که واسطه‌کاربری بخواهد تراکنشی را روی شبکه ارسال کند نیز از کیف پول کاربر می‌خواهد که با داشتن کلید خصوصی کاربر آن تراکنش را امضا و روی شبکه ارسال کند. کیف پول کاربر برای انجام هر یک از این مراحل از کاربر درخواست ثبت تاییدیه می‌کند.

۴.۳ شبکه محلی برای توسعه

برای توسعه یک قرارداد هوشمند نیاز است که قرارداد پس از هر تغییر کامپایل و روی یک شبکه بلاکچین بارگذاری شود، به نحوی که واسطه‌کاربری اپلیکیشن و همچنین کیف پول متامسک بتوانند به آن متصل شوند. از شبکه اصلی نمی‌توان استفاده کرد زیرا هر بارگذاری روی شبکه اصلی هزینه‌ای خواهد داشت و بارگذاری‌های متعدد روی شبکه اصلی امکان پذیر نخواهد بود. اگر بخواهیم برای توسعه از شبکه آزمایشی استفاده کنیم نیز گرچه هزینه‌ای نخواهد داشت اما بسیار زمان بر خواهد بود. ثبت تراکنش‌ها روی شبکه آزمایشی معمولاً سریعتر از شبکه اصلی انجام می‌شود اما همچنان توسعه‌دهنده زمان زیادی را برای هر بارگذاری مصرف خواهد کرد.

راه حل این مشکل این است که توسعه‌دهنده روی ماشین خودش یک شبکه محلی داشته باشد، که بتواند بلافارسله پس از ایجاد یک تغییر روی قرارداد هوشمند آن را کامپایل و بارگذاری کند. ترافل باید بتواند به این شبکه محلی متصل شده و قرارداد را روی آن بارگذاری کند. واسطه‌کاربری و متامسک نیز باید بتوانند به این شبکه متصل شوند که با قرارداد هوشمند ارتباط برقرار کنند.

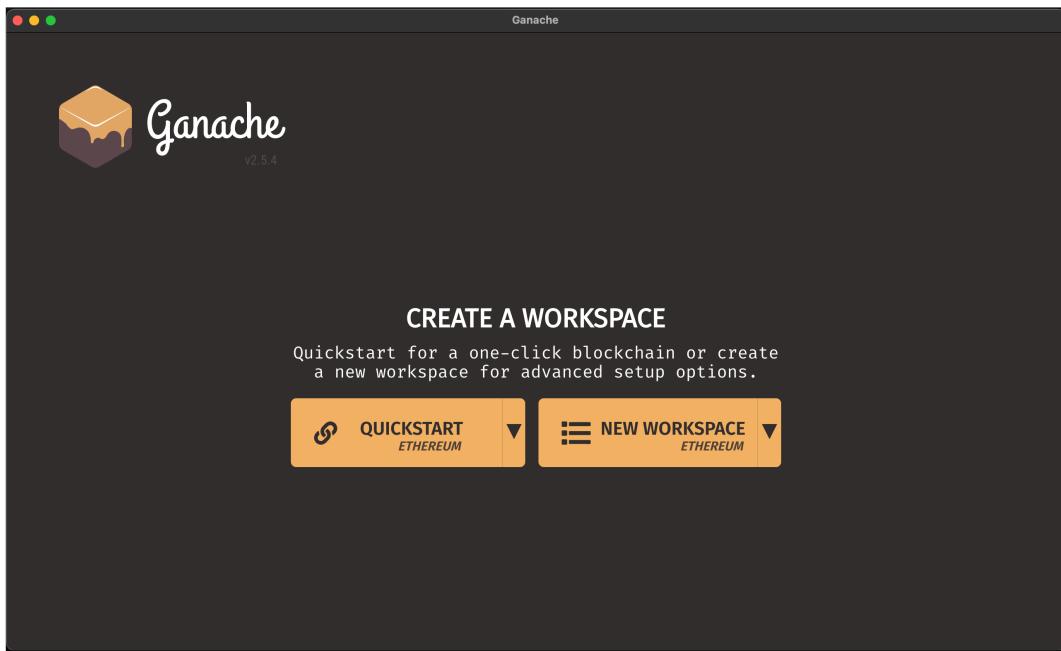
اگرچه ابزارهای زیادی برای ساخت این شبکه محلی وجود دارند، اما یکی از بهترین و راحت‌ترین ابزارها برای این منظور برنامه‌ی گاناچه هست. این ابزار با توجه به این که متعلق به اکوسیستم ترافل هست، به آسانی به آن متصل شده و با اضافه کردن آدرس آن به شبکه‌های متامسک، این کیف پول نیز به شبکه محلی اتصال می‌یابد. جزئیات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

اگرچه توضیحات کامل نحوه استفاده از گاناچه در مستندات این ابزار [۱۲] نیز آمده است، اما در این قسمت توضیحات کوتاه مورد نیاز داده می‌شود. پس از نصب برنامه گاناچه باید یک محیط کار^{۱۱} اتریوم^{۱۲} ساخته شود. برای انجام این کار گزینه

New workspace (Ethereum) انتخاب می‌شود. این دکمه در تصویر ۴.۳ قابل مشاهده است.

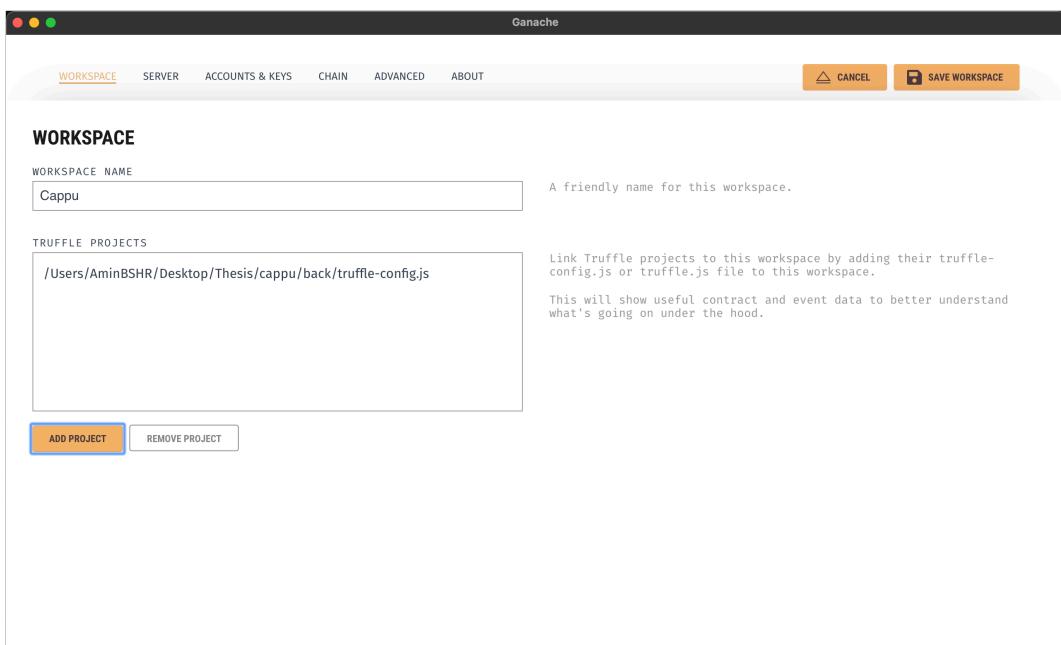
¹¹Workspace

¹²Ethereum



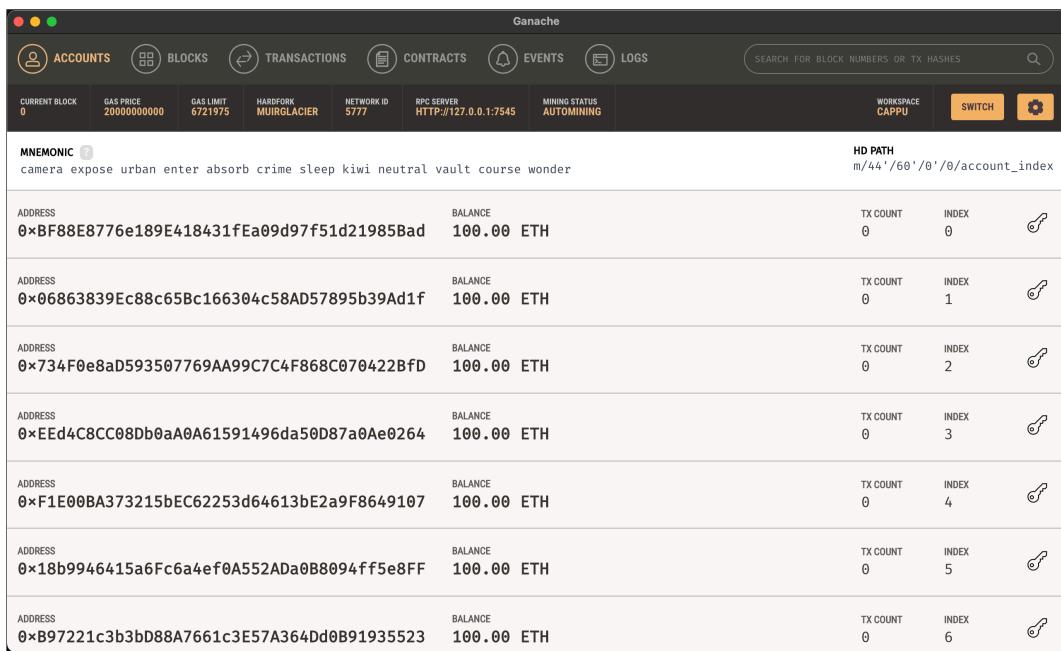
شکل ۴.۳: صفحه اول گاناچه

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 ۵.۳ 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
0xBF88E8776e189E418431fEa09d97f51d21985Bad	100.00 ETH
ADDRESS	BALANCE
0x06863839Ec88c65Bc166304c58AD57895b39Ad1f	100.00 ETH
ADDRESS	BALANCE
0x734F0e8aD593507769AA99C7C4F868C070422BFD	100.00 ETH
ADDRESS	BALANCE
0xEEd4C8CC08Db0aA0A61591496da50D87a0Ae0264	100.00 ETH
ADDRESS	BALANCE
0xF1E00BA373215bEC62253d64613bE2a9F8649107	100.00 ETH
ADDRESS	BALANCE
0x18b9946415a6Fc6a4ef0A552ADa0B8094ff5e8FF	100.00 ETH
ADDRESS	BALANCE
0xB97221c3b3bD88A7661c3E57A364Dd0B91935523	100.00 ETH

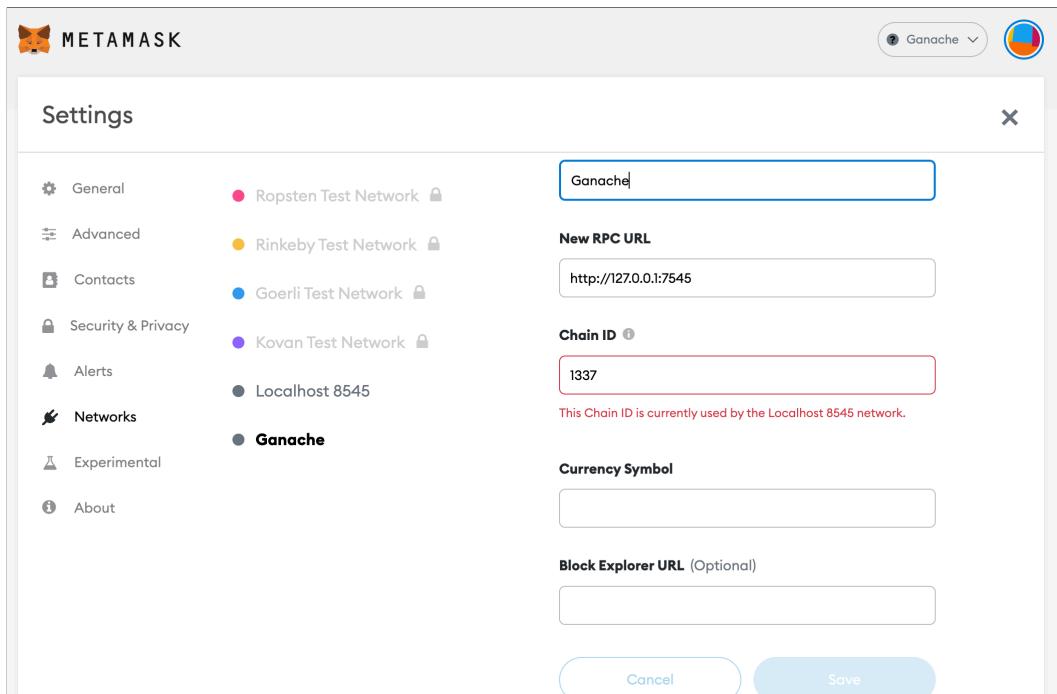
شکل ۳: مشاهده جزئیات شبکه ساخته شده

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^{۱۳} و سپس شبکه‌ها^{۱۴} یک شبکه جدید مطابق تصویر ۷.۳ ساخته می‌شود، همانطور که در تصویر ۶.۳ مشخص است اطلاعات شبکه محلی در صفحه اصلی گانache قابل مشاهده هستند.

¹³Settings

¹⁴Networks



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

۱.۰۴ نوشتن کد قرارداد

در این بخش به بررسی مراحل و نحوه نوشته شدن 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 متغیر را به آسان‌ترین و کم هزینه‌ترین روش ممکن به یک توکن تعویض‌ناپذیر تبدیل کند.
- هر آدرس بتواند توکن‌های خود را به اشخاص دیگر انتقال دهد و یا در بازارهای معاملات توکن‌های تعویض‌ناپذیر بفروشد.
- در صفحه اول وبسایت تعداد کل توکن‌های ساخته شده تا به حال و تعداد کل صاحبان توکن‌ها نمایش داده شود.
- قابلیت‌های قرارداد هوشمند آزمون شده باشد.

۲.۱.۴ ارث‌بری

با توجه به مزایای ذکر شده در مورد استاندارد سازی قراردادهای هوشمند، انتخاب درست این است که برای پیاده‌سازی این کاربری از یکی از استانداردها استفاده شود. ارث‌بری^۱ از استانداردهای یک کتابخانه متن باز 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 سطح پایه‌ای تا حد خوبی تضمین شده است.

قرارداد هوشمند کاپو از استاندارد ERC721 پیاده‌سازی شده در کتابخانه اپن زپلین^۲ 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 توسط داده‌های قرارداد روی شبکه بلاکچین اشغال می‌شود.
- حجم بایت‌کد قرارداد هوشمند.
- تعداد عملیات‌های هر تابع، به ویژه تابع‌هایی که به دفعات زیادی مورد استفاده کاربر قرار می‌گیرند.
- نوع هر تابع، که مشخص می‌کند هر تابع 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^۳ بیشتر شود. فصل ۷ کتاب [۱] Mastering Ethereum در مورد صرفه‌جویی در هزینه تراکنش‌ها توضیحات جامعی ارائه می‌دهد. یکی از مهم‌ترین نکاتی که برای بهینه‌تر رفتار کردن قرارداد هوشمند باید به آن توجه کنیم نوع هر تابع است.

¹Inheritance

²<https://github.com/OpenZeppelin/openzeppelin-contracts>

³Gas fee

اگر تابعی از نوع `pure` تعریف شود به این معنی است که به هیچ اطلاعاتی از شبکه بلاکچین نیاز ندارد و همه‌ی اطلاعاتی که لازم دارد را در اسکوپ^۴ خودش دارد. اگر تابعی از نوع `view` باشد به این معنی است که به اطلاعاتش روی شبکه بلاکچین نیاز دارد اما فقط می‌خواهد آن‌ها را بخواند و نمی‌خواهد تغییری در آن‌ها ایجاد کند. این دو نوع تابع نیازی به پرداخت کارمزد تراکنش توسط کاربر ندارند. اما اگر در تعریف تابعی 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 بیشتر دریافت می‌شود.

۴.۱.۴ جزئیات فنی پیاده‌سازی

ساخت توکن‌های تعویض‌ناپذیر در این قرارداد به آدرس‌های مشخص محدود نیست و همه می‌توانند توکن سازند. 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 مانند شناسه آن 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^۵ داده‌های توکن به دست می‌آید. این نحوه عملکرد یک مزیت بسیار مهم 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.

در یک قرارداد ERC721 استاندارد فقط شناسه توکن‌ها ذخیره می‌شود. در کاپو علاوه بر شناسه توکن‌ها یک نگاشت^۶ 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 داده آن توکن به دست آورده شود. متغیر دوم نیز کمک می‌کند که به راحتی بتوان توکن‌های یک آدرس را به دست آورد. دو متغیر دیگر با نام‌های `numberOfTokenHolders` و `numberOfMintedTokens` نیز در کاپو نگهداشته می‌شوند که برای نمایش آمار استفاده از قرارداد در صفحه اصلی اپلیکیشن مورد استفاده قرار می‌گیرند.

⁴Scope

⁵Hash

⁶Mapping

```

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

```

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

تابع 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 می‌شود، توکن ساخته می‌شود و متغیرهای tokenDatas و numberOfMintedTokens را بروزرسانی می‌گردند. پیاده‌سازی این تابع را می‌توان در تصویر ۱.۴ مشاهده کرد.

تابع afterTokenTransfer از استاندارد ERC721 به نحوی بازنویسی^۷ شده است که پس از هر انتقال توکن با بررسی آدرس‌های مبدا و مقصد، متغیرهای numberOfTokenHolders و numberOfMintedTokens را بروزرسانی کند. نحوه عملکرد این تابع در تصویر ۲.۴ مشخص است.

تابع جدیدی با نام getUserTokens نیز نوشته شده است که در استاندارد ERC721 به صورت پیش‌فرض

⁷Overwrite

```

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`

وجود ندارد. این تابع با گرفتن یک آدرس و استفاده از `ownerTokens` و `tokenDatas` دو خروجی برمی‌گرداند، لیستی از شناسه توکن‌های آدرس و لیستی از داده‌های توکن‌های آدرس. محتوای این تابع در تصویر ۳.۴ قابل مشاهده است.

زبان سالیدیتی به طور پیش‌فرض امکان حذف یک داده از یک آرایه با داشتن مقدار آن را ندارد. دلیل 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 ارث‌بری از بیش از یک قرارداد پدر، برای کاپو یک قرارداد به نام `Helper` نوشته شد که قابلیت حذف کردن یکی از اعضای آرایه با داشتن مقدارش را فراهم می‌کند. این قرارداد در تصویر ۴.۴ قابل مشاهده است. کاپو علاوه بر ERC721 از قرارداد `Helper` نیز ارث‌بری می‌کند.

۲.۴ نوشتن و اجرای آزمون‌ها

پیش‌تر اشاره شد که یکی از مزایای ارث‌بری 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 آزمونی به شدت بالایی دارند. به همین دلیل می‌توان تا حدی به عملکرد قرارداد پدر اطمینان خاطر داشت و بیشتر روی آزمون کردن قابلیت‌های اضافه شده در قرارداد هوشمند فرزنده تمرکز کرد.

در کاپو برای هر عملکرد قرارداد آزمون نوشته شده است. یکی از ساده‌ترین آزمون‌های نوشته شده آزمون فرآیند ساخت یک توکن است که در آن پس از بارگذاری قرارداد با فراخوانی تابع `mint` یک توکن ساخته می‌شود و

```

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

سپس با فراخوانی تابع `balanceOf` دارایی آدرس سازنده توکن بررسی می‌شود و انتظار می‌رود که پس از ساخت یک توکن، دارایی آدرس سازنده توکن یک باشد. این آزمون را می‌توان در تصویر ۵.۴ مشاهده کرد.

پس از نوشته شدن آزمون‌ها می‌توان آن‌ها را با اجرای دستور `truffle test` اجرا کرد. این دستور پس از اجرای آزمون‌ها نتیجه و زمان اجرای هر آزمون را به عنوان خروجی نمایش می‌دهد. نمونه اجرای این دستور را می‌توان در تصویر ۶.۴ مشاهده کرد.

۳.۴ بارگذاری قرارداد روی شبکه آزمایشی راپستن

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

```

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 ۵.۴: نمونه یکی از آزمون‌های قرارداد کاپو

```

→ 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)

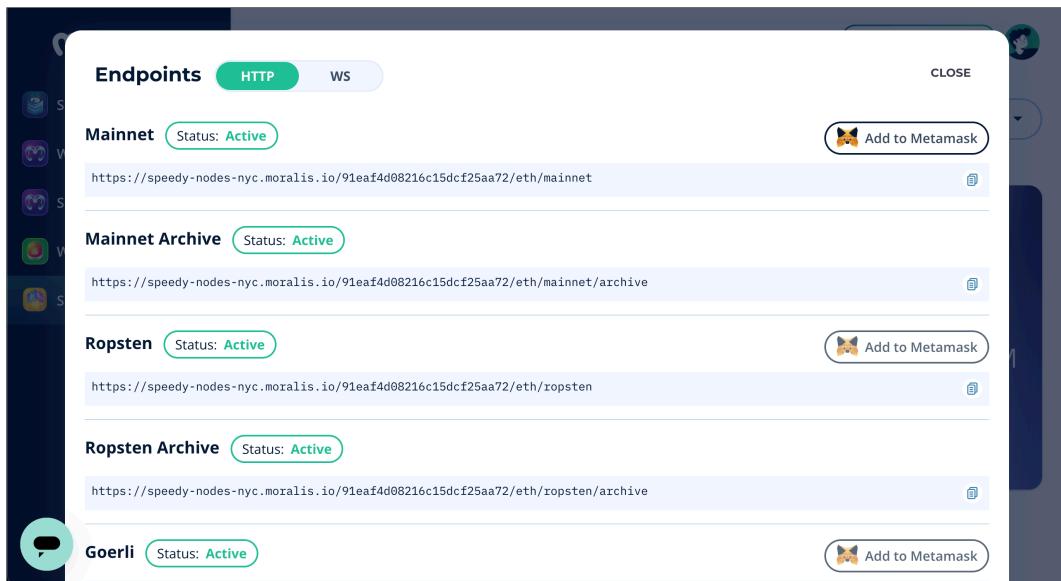
Contract: Cappu
  ✓ should mint a token (1980ms)

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

3 passing (13s)

```

شکل ۶.۴: نمونه خروجی اجرای آزمون‌های قرارداد



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

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

آدرس نُد^۸ های یک شبکه بلاکچین همه به صورت عمومی در دسترس هستند زیرا نُدها باید بتوانند یکدیگر را ببینند. راههای زیادی برای به دست آوردن آدرس یک نُد وجود دارد. یکی از آسان‌ترین راههای به دست آوردن آدرس یکی از نُدهای شبکه مراجعه به وبسایت ماینر^۹ هاست است. برای این پژوهه مشابه تصویر ۷.۴ از وبسایت مورالیس^{۱۰} برای پیدا کردن آدرس نُد شبکه استفاده شد.

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` نیز در پوششی اصلی پروژه ساخته می‌شود. تنظیمات مربوط به ترافل در این فایل قرار دارند. برای این که ترافل شبکه مورد نظر را 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.

⁸Node

⁹Miner

¹⁰ <https://moralis.io>

```

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: 100000000000,
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 راپستن به شبکه‌های ترافل

```

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

```

شکل ۹.۴: ایجاد نمونیکز آزمایشی

۳.۳.۴ آماده شدن نمونیکز

برای انجام این پروژه به کمک دستور npm mnemonics مطابق تصویر ۹.۴ یک آدرس آزمایشی ساخته می‌شود. این دستور، نمونیکز متناسب با این آدرس را به عنوان خروجی می‌دهد. دقت کنید که برای بارگذاری روی شبکه اصلی^{۱۱} حتماً باید از نمونیکز مربوط به یک کیف پول واقعی استفاده شود و اطلاعات ان در اختیار کسی قرار نگیرد.

۴.۳.۴ استفاده از کیف پول ایجاد شده در تنظیمات ترافل

ترافل برای این که بتواند از کیف پول 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.

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

¹¹Mainnet

```

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

```

شکل ۱۰.۴: قراردادن نمونیکز در فایل secrets.json

```

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

```

شکل ۱۱.۴: معرفی فایل secrets.json در تنظیمات ترافل

۵.۳.۴ نصب کیف‌پول

ترافل برای استفاده از نمونیکز کیف‌پول مانیاز به نصب بسته hdwallet-provider دارد، این بسته کاربری‌های یک کیف‌پول دیجیتال از جمله امضا و ارسال تراکنش بر روی شبکه بلاکچین را در اختیار ترافل قرار می‌دهد. این بسته 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.

۶.۳.۴ انتخاب شبکه اضافه شده

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

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 آزمایشی انجام می‌شود باید با استفاده از یک faucet روی شبکه

```

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

```

شکل ۱۲.۴: استفاده از کیف‌پول hdwallet در تنظیمات ترافل

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

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

```
truffle(ropsten)> await web3.eth.getAccounts()
[
  '0xF51f5f41BfA8ADa57a43862cBc18dA4750AecB4c',
  '0x909ebC92395FC4335c35894C7DDc8bffFDcEfF06',
  '0x48156708DF687C7a8F97C951b5E734E132e891D1',
  '0xF1C6c91D80032528e2C01F73DAd588D11DA0f17d',
  '0xA6f899d10B4E1c1195AFD1C6f29E4e539C828450',
  '0xB63191Dd13637c024C7F1F339F254F0F13d4bB34',
  '0x1699Ba468F7E5af64f510B323537bbcd107373F9',
  '0x6eDd855A6D2d3De5D96749e1bD3E9580c33468E7',
  '0x8A97C0bfC3086DFcd9E1B25D69A1A238A1290BE6',
  '0xc4838df4d46862d1226BDC409EbE8395cA6fE703'
]
```

شکل ۱۴.۴: دریافت آدرس‌های کیف‌پول در خط فرمان ترافل

آزمایشی به میزان کافی پول آزمایشی دریافت شود.

برای دریافت آدرس‌های کیف‌پول مانند تصویر ۱۴.۴ از دستور `getAccounts` در خط فرمان ترافل استفاده می‌شود. همچنین برای دریافت مانده حساب آدرس، مانند تصویر ۱۵.۴ از دستور `getBalance` در خط فرمان ترافل استفاده می‌شود.

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.

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

شکل ۱۵.۴: دریافت موجودی کیف‌پول در خط فرمان ترافل

Txn Hash	Method ⓘ	Block	Age	From	To	Value	Txn Fee
0xd207a3f62e89fcfc054c...	Safe Transfer Fr...	12147200	7 days 19 hrs ago	0xf1359760a1b37a9f8d8...	0xf1359760a1b37a9f8d8...	0 Ether	0.000352610001
0xece8b0c2be5bd13534...	Mint	12147179	7 days 19 hrs ago	0xf1359760a1b37a9f8d8...	0xf1359760a1b37a9f8d8...	0 Ether	0.021437500002
0xd754232c700bd4cb75...	Mint	12147178	7 days 19 hrs ago	0xf1359760a1b37a9f8d8...	0xf1359760a1b37a9f8d8...	0 Ether	0.000273017508

شکل ۱۶.۴: مشاهده قرارداد کاپو در اتراسکن روی شبکه راپستن

۹.۳.۴ اطمینان از صحیح بارگذاری قرارداد هوشمند

پس از اتمام بارگذاری قرارداد هوشمند برای اطمینان از به درستی انجام شدن فرآیند بارگذاری قرارداد، می‌توان از جستجوگرهای شبکه^{۱۲} بلاکچین استفاده کرد. برای مثال قرارداد هوشمند کاپو بر روی شبکه راپستن بارگذاری شده است، که با رفتن به وبسایت اتراسکن^{۱۳} و قراردادن آن روی شبکه راپستن، مانند تصویر ۱۶.۴ می‌توان قرارداد بارگذاری شده و تراکنش‌های آن را مشاهده کرد.

۴.۴ توسعه واسطکاربری

برای توسعه واسطکاربری اپلیکیشن، ری‌اکت^{۱۴} به عنوان چارچوب مورد استفاده انتخاب شده است. ترکیب این چارچوب با استفاده از کتابخانه material-ui که کمک می‌کند در زمان کوتاه بتوان ظاهری زیبا و یکدست در اپلیکیشن ایجاد کرد و کتابخانه Web3JS که واسطکاربری را به کیف پول کاربر و شبکه بلاکچین متصل می‌کند، همه‌ی قابلیت‌های مورد نیاز برای توسعه یک واسطکاربری زیبا و کارآمد را در اختیار توسعه‌دهنده قرار می‌دهد. برای استفاده از این دو کتابخانه جامع‌ترین منبع مستندات خودشان [۸][۴] است.

¹²Block Explorers

¹³<https://etherscan.io>

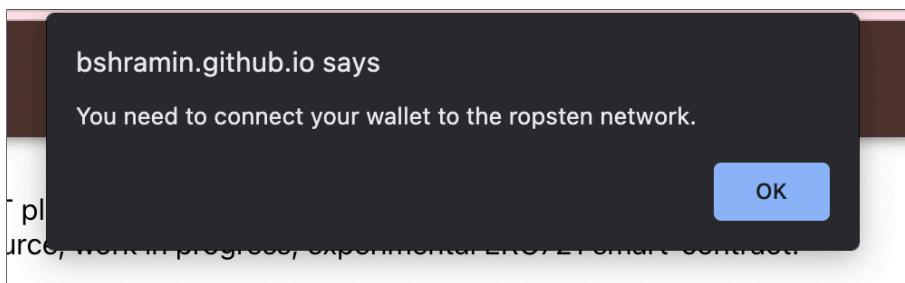
¹⁴React

۱.۴.۴ اتصال به کیف پول و قرارداد کاپو

در پوشه اصلی واسطه‌کاربری فایلی با عنوان config.js وجود دارد. در این فایل علاوه بر ABI^{۱۵} 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 اشاره کند.

برای استفاده از کتابخانه Web3JS و اتصال به کیف‌پول کاربر یک فایل به نام connect.js ساخته شد. تمامی اعمال ارتباطی با کیف‌پول 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 کیف‌پول و ورود^{۱۶} کاربر، خروج^{۱۷} کاربر، گرفتن آدرس و شبکه‌ی کیف‌پول و ... در این فایل انجام می‌شود.

واسطه‌کاربری کاپو پس از تایید کاربر و دریافت آدرس کیف‌پول او، آن را در sessionStorage ذخیره می‌کند. از این طریق متوجه می‌شود که آیا کاربر کیف‌پول خود را به برنامه متصل کرده وارد شده است یا خیر. کاپو پیش از اتصال به کیف‌پول کاربر چک می‌کند که کیف‌پول روی شبکه یکسانی با شبکه فعلی کاپو باشد و در غیر این صورت مانند تصویر ۱۷.۴ به کاربر هشدار می‌دهد که شبکه‌ی کیف‌پول را تغییر دهد.



شکل ۱۷.۴: پیام کاپو به کاربر هنگامی که کیف‌پول روی شبکه مورد نظر نباشد

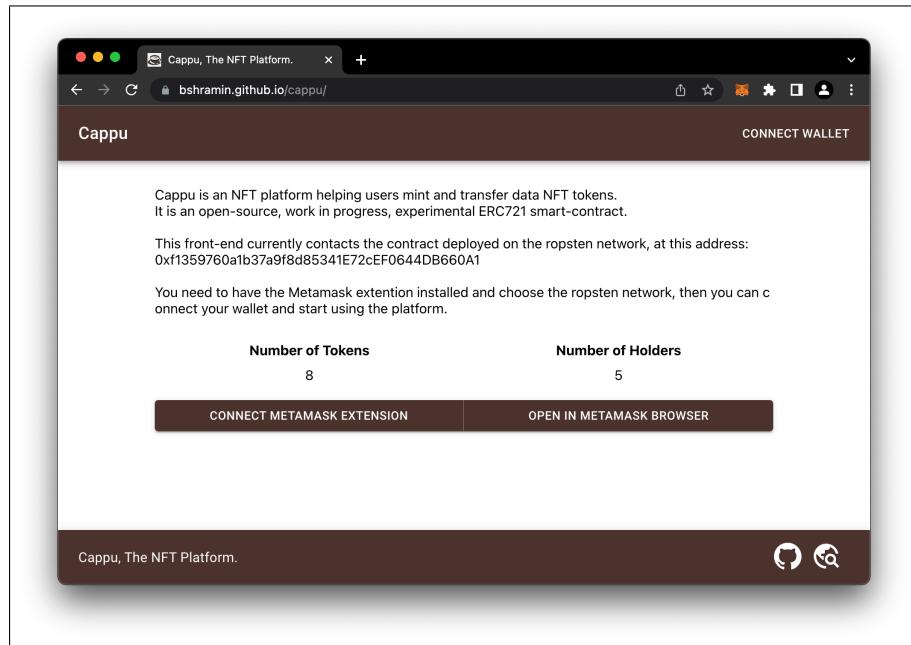
۲.۴.۴ تجربه کاربری

همچنین در واسطه‌کاربری کاپو برای داشتن تجربه کاربری بهتر تلاش شده است. نکاتی مانند عدم نمایش قابلیت‌هایی مانند ساخت و ارسال توکن هنگامی که کیف‌پول کاربر به اپلیکیشن متصل نیست، جایه‌جایی آسان

¹⁵ Application Binary Interface

¹⁶ Login

¹⁷ Logout



شکل ۱۸.۴: صفحه اصلی کاپو (بدون کیف پول متصل)

میان صفحات به کمک react-router، طراحی تعاملی^{۱۸} برای رایانه و گوشی موبایل، نمایش هشدار^{۱۹} ها و خطای^{۲۰} های مناسب به کاربر، نمایش نشانگر بارگذاری^{۲۱} هنگامی که تراکنش‌ها در حالت انتظار^{۲۲} هستند و نمایش پیام‌های مناسب با توجه به نتیجه تراکنش‌های کاربر.

در این قسمت روند تعامل یک کاربر با کاپو مورد بررسی قرار می‌گیرد. هنگامی که کاربر برنامه کاپو را باز می‌کند و هنوز کیف پولش را متصل نکرده است، مانند تصویر ۱۸.۴ توضیحاتی در مورد نحوه عملکرد کاپو و همچنین آماری از تعداد توکن‌های ساخته شده و تعداد آدرس‌های دارای نوکن را مشاهده می‌کند. کاربر هنوز گزینه‌هایی مانند ساخت توکن یا لیست توکن‌ها را نمی‌بیند زیرا به آن‌ها دسترسی ندارد. پس از اتصال کیف‌پول کاربر به کاپو گزینه‌های ساخت توکن و لیست توکن‌ها مانند تصویر ۱۹.۴ به صفحه اضافه می‌شوند.

حال کاربر که کیف پول خود را به کاپو متصل کرده است می‌تواند توکن بسازد. برای این منظور از قسمت بالای صفحه گزینه Mint را انتخاب می‌کند. در صفحه‌ای که باز می‌شود اطلاعات توکن را وارد کرده و دکمه Mint را می‌زند. حال کاپو تراکنش ساخت توکن را به کیف پول کاربر می‌فرستد که توسط کاربر تایید و روی شبکه

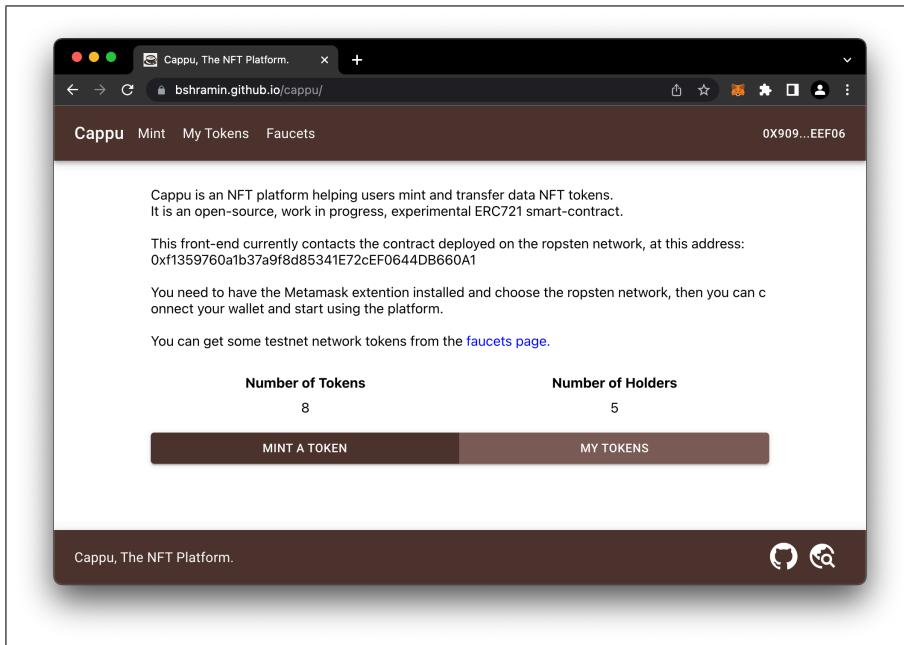
¹⁸Responsive

¹⁹Alert

²⁰Error

²¹Loading

²²Pending

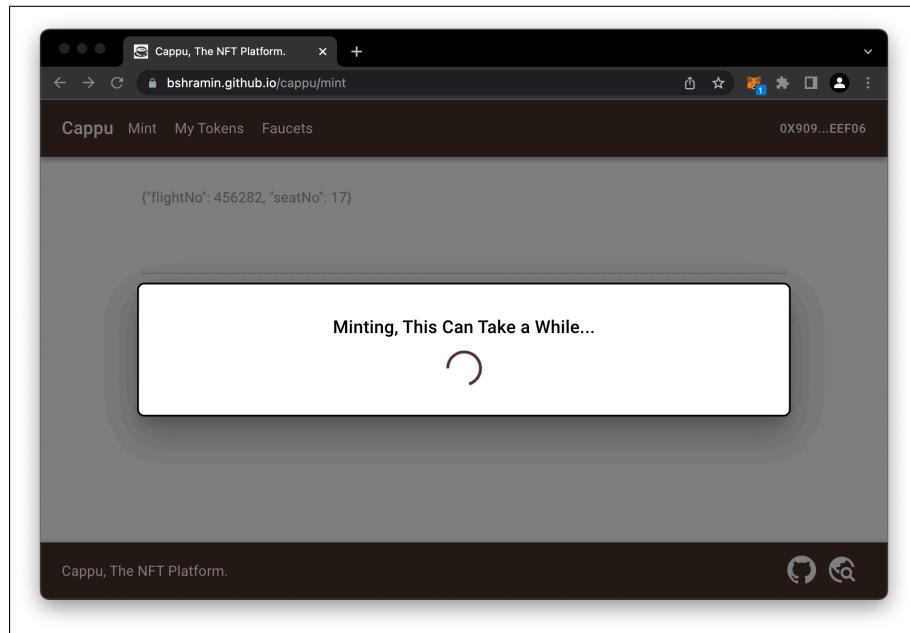


شکل ۱۹.۴: صفحه اصلی کاپو (با کیف پول متصل)

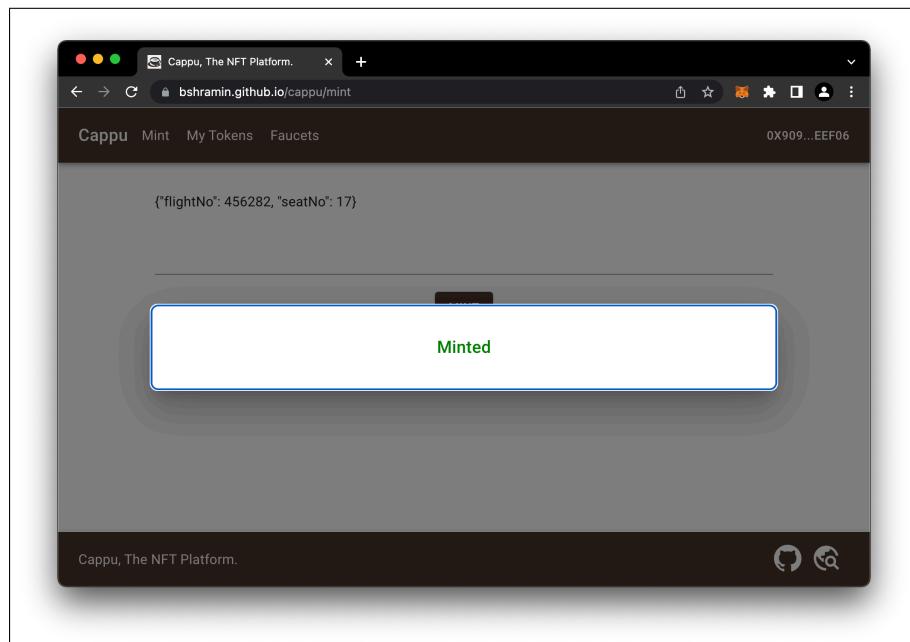
ارسال شود. تا زمانی که نتیجه‌ی تراکنش مشخص نشده است کاپو مانند تصویر ۲۰.۴ پیام انتظار به کاربر نمایش می‌دهد. سپس در صورت موفقیت آمیز بودن ساخت توکن، مانند تصویر ۲۱.۴ پیام مناسب به کاربر نمایش داده می‌شود.

پس از ساخت یک یا چند توکن، کاربر می‌تواند با انتخاب گزینه My Tokens یه صفحه لیست توکن‌ها منتقل شود. در این قسمت کاربر می‌تواند دارایی‌هایش را مشاهده کند. در صورتی که کاربر دارای توکنی نباشد پیام مناسب مانند تصویر ۲۲.۴ به کاربر نمایش داده می‌شود. در صورتی که کاربر توکن‌هایی داشته باشد نیز مانند تصویر ۲۳.۴ لیست توکن‌هایش را مشاهده می‌کند.

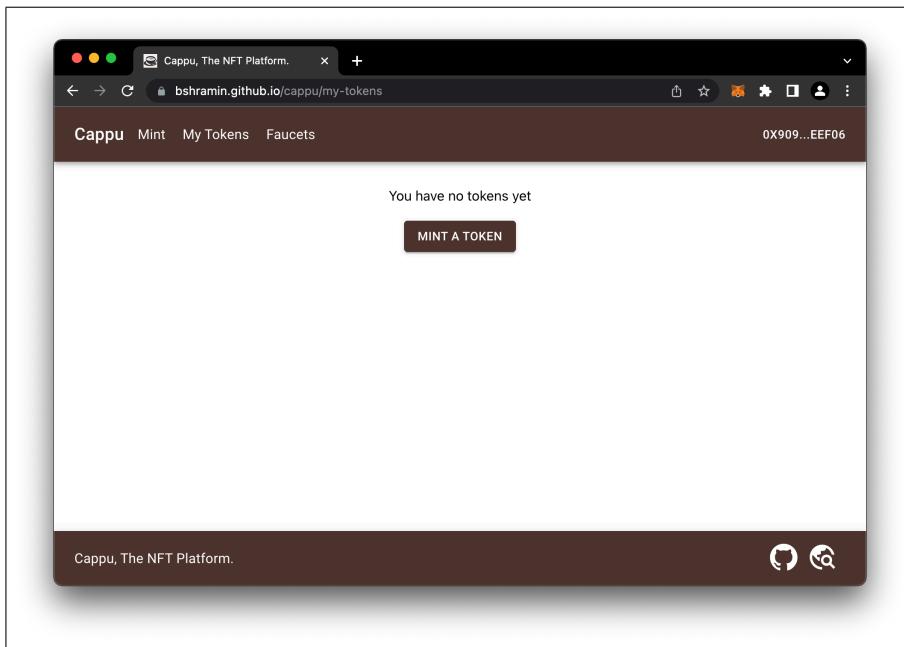
در این صفحه کاربر می‌تواند هر یک از توکن‌ها را با انتخاب دکمه Transfer به شخص دیگری ارسال کند. با انتخاب گزینه انتقال توکن، صفحه‌ی کوچکی مانند تصویر ۲۴.۴ باز می‌شود که کاربر می‌تواند در آن آدرس مقصد را وارد کرده و توکن را ارسال کند. پس از فشردن دکمه انتقال، مانند تصویر ۲۵.۴ پیام انتظار به کاربر نمایش داده می‌شود. در صورت موفق بودن انتقال کاربر مانند تصویر ۲۶.۴ پیام موفقیت را مشاهده می‌کند. همچنین اگر تراکنش با مشکلی مواجه شود نیز پیام مناسب مانند تصویر ۲۷.۴ به کاربر نمایش داده می‌شود.



شکل ۲۰.۴: کاپو در حال ساخت توکن



شکل ۲۱.۴: پیام موفقیت در ساخت توکن

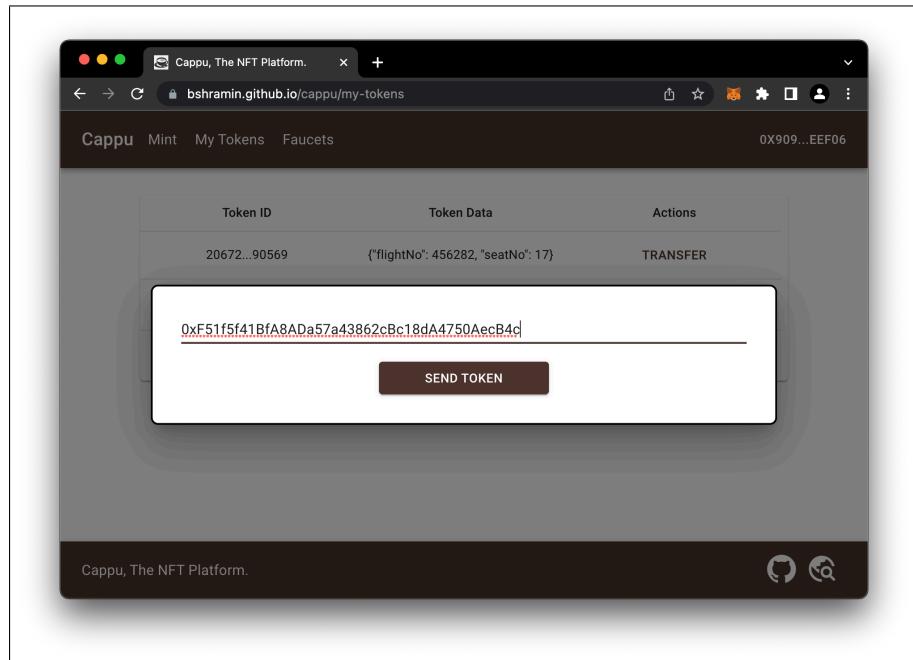


شکل ۲۲.۴: کاربر توکنی ندارد

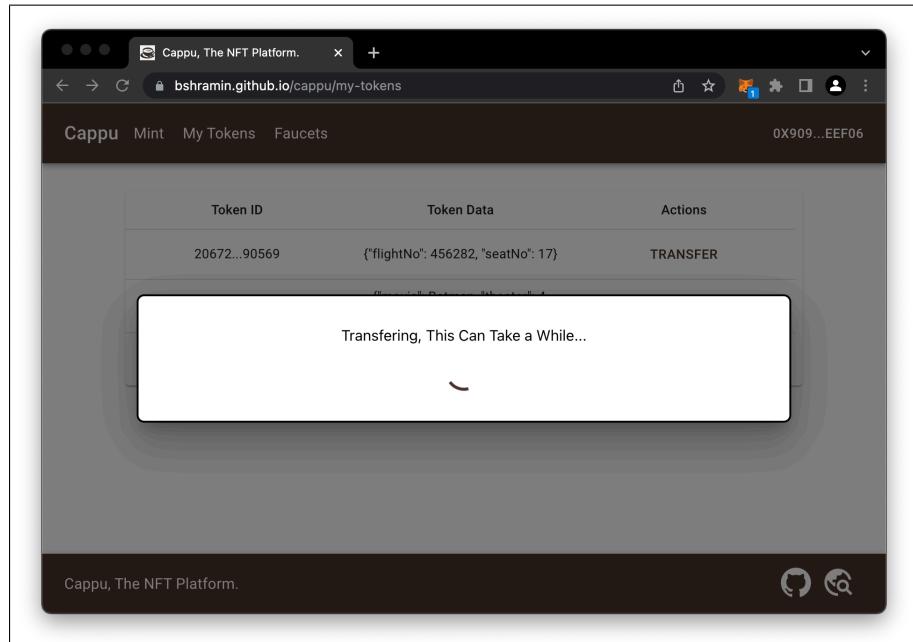
The screenshot shows a web browser window for 'Cappu, The NFT Platform.' at the URL bshramin.github.io/cappu/my-tokens. The page has a dark header with tabs for 'Cappu', 'Mint', 'My Tokens', and 'Faucets'. On the right side of the header, it displays the address '0X909...EEF06'. The main content area contains a table with three rows of token data. The table has three columns: 'Token ID', 'Token Data', and 'Actions'. The 'Actions' column for each row contains the word 'TRANSFER'. The 'Token Data' column contains JSON objects representing different types of tokens. The first row's data is: {"flightNo": 456282, "seatNo": 17}. The second row's data is: {"movie": Batman, "theater": 4, "seatNo": 17}. The third row's data is: {"type": "dentist", "date": "2022-04-22", "time": "14:30"}. The footer of the page is identical to the one in the previous screenshot.

Token ID	Token Data	Actions
20672...90569	{"flightNo": 456282, "seatNo": 17}	TRANSFER
72551...84314	{"movie": Batman, "theater": 4, "seatNo": 17}	TRANSFER
18307...56308	{"type": "dentist", "date": "2022-04-22", "time": "14:30"}	TRANSFER

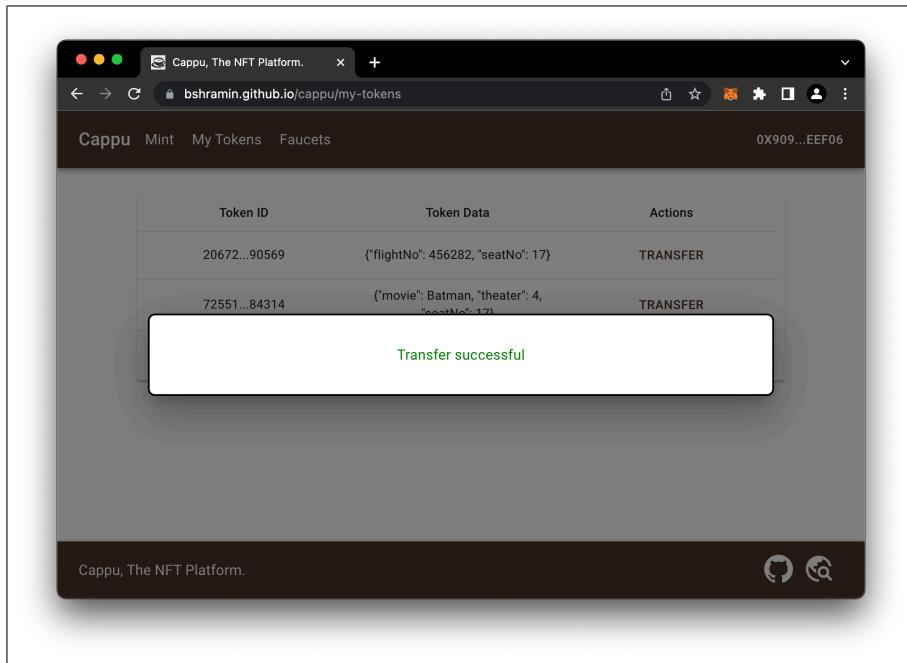
شکل ۲۳.۴: لیست توکن‌های کاربر



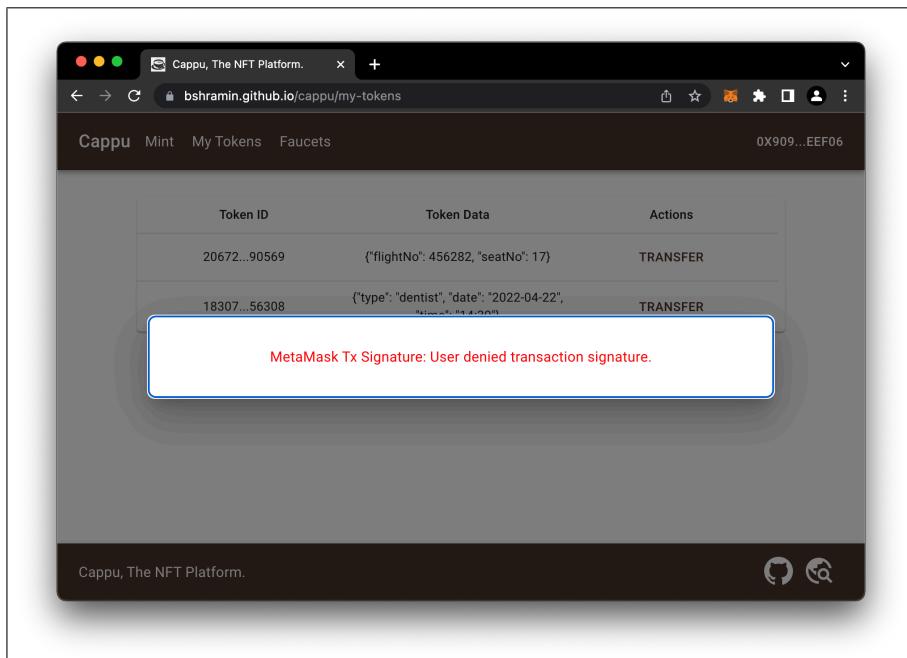
شکل ۲۴.۴: صفحه ورود آدرس مقصد و انتقال توکن



شکل ۲۵.۴: در حال ارسال توکن



شکل ۲۶.۴: ارسال موفق توکن



شکل ۲۷.۴: ارسال توکن با مشکل مواجه شده

۳.۴.۴ بارگذاری واسطکاربری

برای این که کاربرها بتوانند با قرارداد هوشمند ارتباط برقرار کنند نیاز است که واسطکاربری اپلیکیشن در سروری بارگذاری شود. خوشبختانه گیت‌هاب قابلیت به نام صفحات گیت‌هاب در اختیار کاربرانش قرار می‌دهد که به کمک آن می‌توان واسطکاربری اپلیکیشن را در آدرسی مناسب با آدرس مخزن کد در گیت‌هاب بارگذاری کرد و کاربران با رجوع به آن آدرس می‌توانند واسطکاربری اپلیکیشن را ببینند و از آن استفاده کنند. جزئیات این فرآیند در مستندات این ابزار [۶] کاملاً شرح داده شده است.

این قابلیت گیت‌هاب در واقع به این صورت عمل می‌کند که یک برنچ به نام gh-pages در مخزن^{۲۳} پروژه می‌سازد و هر بار که دستور بارگذاری پروژه توسط گیت‌هاب اجرا می‌شود، یک بیلد از پروژه می‌گیرد و فایل‌های خروجی بیلد روی این برنچ پوش می‌شوند. سپس این فایل‌ها روی آدرسی مناسب با آدرس مخزن کدهای پروژه بارگذاری می‌شوند. برای مثال آدرس ریپازیتوری و واسطکاربری اپلیکیشن کاپو به صورت زیر است:

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

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

البته بارگذاری شدن واسطکاربری روی صفحات گیت‌هاب با ایجاد مشکلاتی در مسیریابی^{۲۴} 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 آزمون‌های قرارداد هوشمند

برای سرعت بخشیدن به توسعه قرارداد هوشمند، این نیازمندی به وجود آمد که بعد از پوش شدن هر تغییر روی گیت‌هاب آزمون‌های قرارداد به صورت خودکار اجرا شوند. به این منظور نیاز است که آزمون‌های قرارداد

²³Repository

²⁴Routing

هوشمند بتوانند در یک کانتینر داکر اجرا شوند.

برای داکرایز کردن اجرای آزمون‌های قرارداد هوشمند، نخست سعی در این بود که یک ایمیچ 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 ایمیچ پایه داکر نوشته شد که داکرفایل آن را می‌توان در گیت‌هاب^{۲۵} مشاهده کرد، همچنین این ایمیچ داکر در داکرهاب^{۲۶} نیز پوشش دارد. راهنمایی چگونگی انجام این مراحل در مستندات داکر [۲] ذکر شده است.

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

۲.۵.۴ اجرای خودکار آزمون‌های قرارداد

با داشتن داکرفایلی که با بیلد و اجرای آن آزمون‌های قرارداد هوشمند اجرا می‌شوند، آزمون‌های قرارداد هوشمند می‌توانند به عنوان یکی از مراحل پایپلاین پروژه در گیت‌هاب نیز اجرا گردند. به این صورت در هر مرحله ریکوئست به برنج اصلی^{۲۷} و با پوشش دادن یک کامیت در برنج اصلی آزمون‌ها به صورت خودکار در پایپلاین گیت‌هاب اجرا می‌شوند. برای پیاده‌سازی این فرآیند به مستندات پایپلاین‌های گیت‌هاب [۵] رجوع شده است. به این ترتیب سرعت توسعه و اطمینان از کدهای قرارداد بیشتر می‌شود.

۳.۵.۴ بارگذاری خودکار واسطکاربری

برای ساده‌سازی بیشتر فرآیند بارگذاری واسطکاربری و سرعت بخشیدن به توسعه آن، این قابلیت پیاده‌سازی می‌شود که پس از هر بار ایجاد تغییر در واسطکاربری، به جای این که توسعه‌دهنده با اجرای دستوراتی واسطکاربری را به کمک صفحات گیت‌هاب بارگذاری کند، واسطکاربری پس از پوشش دادن تغییرات جدید روی برنج اصلی ریپازیتوری بارگذاری می‌شود.

برای پیاده‌سازی این قابلیت از Github Actions که در واقع پایپلاین‌های گیت‌هاب برای یک پروژه هستند

²⁵ <https://github.com/bshramin/truffle-docker>

²⁶ <https://hub.docker.com/r/aminbshr/truffle>

²⁷ Master branch

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 واسطکاربری را بیلد کند و سپس فایل‌های بیلد شده را روی برنج 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 واسطکاربری بیلد و سپس به کمک صفحات گیت‌هاب روی برنج 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 متغیر محلی²⁸ تعریف می‌شود و هنگامی که در استیج بارگذاری واسطکاربری این داکرفایل بیلد و اجرا می‌شود توکنی که از گیت‌هاب گرفته شده است به عنوان متغیر محلی به این کانتینر داده می‌شود. به این ترتیب این توکن درون کانتینر داکر وجود خواهد داشت و Github Pages از آن استفاده خواهد کرد.

²⁸Environment variable

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 آزمایشی را پستن 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 کیف پولشان به اپلیکیشن می‌توانند توکن بسازند، دارایی‌هایشان را مشاهده کنند و توکن‌هایشان رو به دیگران ارسال کنند.

۲.۱.۵ ساخت محیط توسعه سریع و خودکار

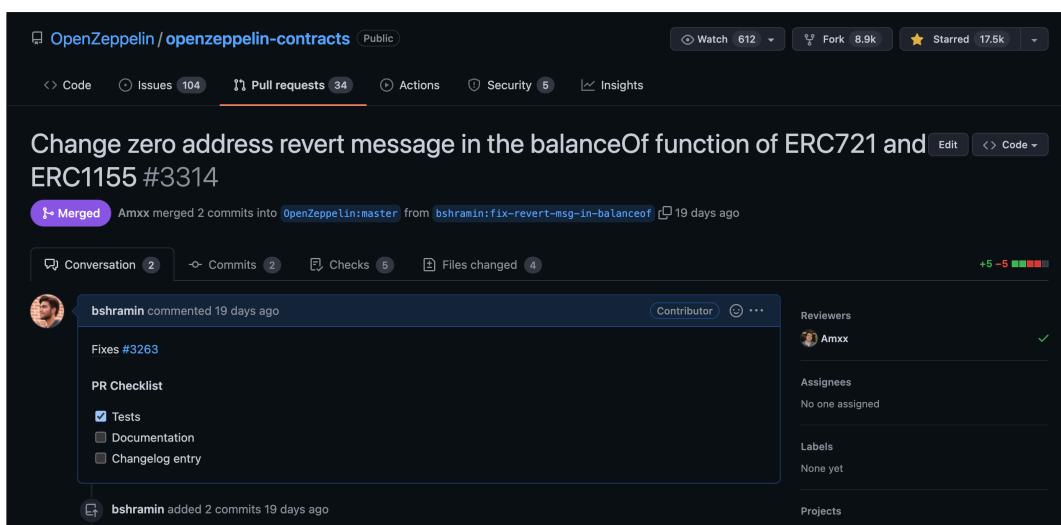
سپس آموختیم که چگونه اجرای آزمون‌های قرارداد هوشمند را داکرایز و به صورت خودکار در پایپ‌لاین پروژه اجرا کنیم. برای انجام این کار یک داکر ایمیج ترافل نوشته شد، که آن به صورت متن باز بر روی گیت‌هاب بارگزاری و ایمیج آن به داکرهاب اضافه شد. سپس واسطه کاربری اپلیکیشن داکرایز شد و از متغیر محلی‌های داکر برای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

در پایلین پروژه روی صفحات گیت‌هاب بارگزاری می‌شود.

در نتیجه انجام این کارها یک مسیر راحت و سریع برای توسعه یک قرارداد هوشمند به همراه واسطه‌کاربری ایجاد شد که آزمون‌ها 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 توسعه‌دهندگان 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 توسعه‌دهنده قرار می‌دهد. چگونه می‌توان یک شبکه محلی برای توسعه ایجاد کرد، قرارداد هوشمند را بروی آن بارگذاری کرد و واسطکاربری و کیف پول را به آن متصل کرد.

آموختیم که چگونه می‌توانیم برای پیاده‌سازی قراردادهای هوشمند از استانداردهای موجود استفاده کنیم، برای آن‌ها آزمون بنویسیم و به کمک چارچوب ترافل این آزمون‌ها را اجرا کنیم. آموختیم که چگونه پس از اتمام فرآیند توسعه قرارداد هوشمند را بروی شبکه آزمایشی بارگذاری کنیم. همچنین واسطکاربری اپلیکیشن به کمک صفحات گیت‌هاب بارگذاری و به قرارداد هوشمند روی شبکه آزمون 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.

۱.۲.۵ استفاده از استانداردها

خوبی‌ختنانه در این پروژه از ابتدا به استفاده از استانداردهای موجود اهمیت داده شد. در صورتی که توسعه‌دهنده 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 توسعه‌دهنده 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 آزمون‌ها، 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 توسعه‌دهنده را به میزان خوبی باز می‌گذارد. پیشنهاد می‌شود حتماً پیش از استفاده از این کتابخانه مستندات دریزل [۱۱] مشاهده شود تا با اطمینان خاطر از رفع نیازها از آن استفاده شود.

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] Antonopoulos, Andreas. *Mastering Ethereum*. O'Reilly Media, Inc., 2019.
- [2] DockerInc. Docker documentation. <https://docs.docker.com/>. Accessed: 2022-01-20.
- [3] EthereumFoundation. Ethereum documentation. <https://ethereum.org/en/developers/docs/>. Accessed: 2022-01-20.
- [4] EthereumFoundation. Web3js documentation. <https://web3js.readthedocs.io/en/v1.7.3/>. Accessed: 2022-01-20.
- [5] GitHub. Github actions documentation. <https://docs.github.com/en/actions>. Accessed: 2022-01-20.
- [6] GitHub. Github pages documentation. <https://docs.github.com/en/pages>. Accessed: 2022-01-20.
- [7] Metamask. Metamask documentation. <https://docs.metamask.io/guide/>. Accessed: 2022-01-20.
- [8] MUI. Material ui documentation. <https://mui.com/material-ui/>. Accessed: 2022-01-20.
- [9] OpenSea. Opensea documentation. <https://docs.opensea.io/docs>. Accessed: 2022-01-20.
- [10] OpenZeppelin. Zeppelin contracts. <https://docs.openzeppelin.com/>. Accessed: 2022-01-20.
- [11] TruffleFramework. Drizzle documentation. <https://trufflesuite.com/docs/drizzle/>. Accessed: 2022-01-20.
- [12] TruffleFramework. Ganache documentation. <https://trufflesuite.com/docs/ganache/>. Accessed: 2022-01-20.

[13] TruffleFramework. Truffle suite documentation. <https://trufflesuite.com/docs/truffle/>. Accessed: 2022-01-20.

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.

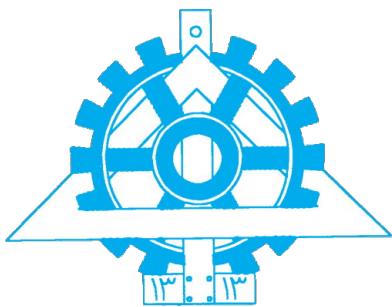
With this project, we are developing an easy-to-use platform so that any person or organization can mint and transfer NFTs. The use cases for such a platform are endless. In Cappu, assets such as cars, houses, movie tickets, and doctor appointments can be minted, sold, and transferred to others as an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