



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



## کاپو، پلتفرم ساخت و انتقال NFT های داده‌ای

پایان‌نامه برای دریافت درجه کارشناسی ارشد در رشته مهندسی کامپیوتر  
گرایش نرم‌افزار

امین بشیری

استاد راهنما

دکتر احسان خامس‌پناه

خرداد ۱۴۰۱









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



## کاپو، پلتفرم ساخت و انتقال NFT های داده‌ای

پایان نامه برای دریافت درجه کارشناسی ارشد در رشته مهندسی کامپیوتر  
گرایش نرم افزار

امین بشیری

استاد راهنما

دکتر احسان خامس پناه

خرداد ۱۴۰۱





دانشگاه تهران  
پردیس دانشکده‌های فنی  
دانشکده مهندسی برق و کامپیوتر



## گواهی دفاع از پایان‌نامه کارشناسی ارشد

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

به عدد	به حروف
<input type="text"/>	<input type="text"/>

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	
۲	استاد داور داخلی	دکتر داور داخلی	دانشیار	دانشگاه تهران	
۳	استاد مدعو	دکتر داور خارجی	دانشیار	دانشگاه داور خارجی	
۴	نماینده تحصیلات تکمیلی دانشکده	دکتر نماینده	دانشیار	دانشگاه تهران	

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

باسمه تعالی

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 متمرکز»<sup>۱</sup> یا Web<sup>۳</sup> به عنوان مهمترین تغییر بعد از به وجود آمدن اینترنت<sup>۲</sup> در نظر گرفته می شود. با به وجود آمدن «رمزارها»<sup>۳</sup> و «الگوریتم های اجماع»<sup>۴</sup> کارآمد، و فراهم شدن زمینه اجرای برنامه ها و انجام تراکنش های مالی به صورت غیر متمرکز، عصر اینترنت غیر متمرکز فرا رسیده است.

در این میان یکی از اصلی ترین مزایای «برنامه های غیر متمرکز»<sup>۵</sup> مالکیت واقعی دارایی است، به این معنی که یک شخص یا یک نهاد نمی تواند دارایی های کس دیگری را مسدود یا مصادره کند. از طرفی مالکیت های معنوی هم به صورت واضح و شفاف می توانند مشخص شوند. برای مثال یک هنرمند به وضوح صاحب اثرش است و هرچند که دیگر افراد می توانند اثر او را کپی کنند اما همیشه مشخص است که صاحب اصلی اثر کیست.

به این ترتیب «توکن های تعویض ناپذیر»<sup>۶</sup> با قابلیت های مالکیت بسیار زیادی که فراهم می کنند مورد استقبال فراوان مردم واقع شدند. قابلیت هایی مانند ساخت، نگهداری، فروش و انتقال فوق العاده راحت و سریع نیز در این سرعت فراگیری تاثیر بسزایی داشته اند. برای بازاری به این تازگی و وسعت، تکنولوژی ها، استانداردها و پلتفرم های زیادی ساخته شده اند 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 آسان ترین روش ممکن توکن های تعویض ناپذیر بسازد و به دیگران انتقال دهد. کاربردهای این پلتفرم ساده بی شمار است. دارایی هایی مانند بلیت سینما، ژتون های غذا، وقت گرفتن از دکتر، قراردادهای ... همه می توانند به آسانی در این پلتفرم به توکن تعویض ناپذیر تبدیل شوند، به دیگران انتقال یابند و در بازار خرید و فروش شوند.

اگرچه کاربردهای فراوانی برای این پلتفرم می توان در نظر داشت اما همچنان هدف اصلی از انجام این پروژه آشنایی با نحوه ساخت، تست و بارگذاری یک اسمارت کانترکت، ساخت واسط کاربری، اتصال آن به اسمارت کانترکت و همچنین شناخت استانداردهای معروف قراردادهای توکن های تعویض ناپذیر مانند ERC721 و ERC1155 است.

لازم به ذکر است که تمامی کدهای کاپو به صورت متن باز در گیت هاب<sup>۷</sup> قابل دسترس برای عموم هستند.

---

<sup>1</sup>Decentralized Web

<sup>2</sup>Word Wide Web

<sup>3</sup>CryptoCurrencies

<sup>4</sup>Consensus Algorithms

<sup>5</sup>Dapps

<sup>6</sup>Non-fungible tokens

<sup>7</sup><https://github.com/bshramin/cappu>

واژگان کلیدی    کاپو، توکن، داده، سالی‌دیتی، اسمارت کانترکت، قرارداد هوشمند، توکن غیرقابل تعویض،  
non-fungible, ERC1155, ERC721, solidity, cappu

# فهرست مطالب

فصل ۱: مقدمه و بیان مسئله	۱
۱.۱ مقدمه	۱
۲.۱ شرح مسئله و روش انجام آن	۲
۳.۱ اهداف کلی تحقیق	۲
۱.۳.۱ گسترش کاربردهای توکن‌های تعویض ناپذیر	۳
۲.۳.۱ یادگیری	۳
۴.۱ ساختار پایان‌نامه	۴
فصل ۲: مفاهیم اولیه و پیش‌زمینه	۵
۱.۲ دلایل و برتری‌های متن‌باز بودن قراردادهای هوشمند	۵
۲.۲ آشنایی با مفهوم توکن تعویض ناپذیر	۶
۳.۲ کاربردها، حال و آینده	۷
۴.۲ قراردادهای هوشمند و استانداردسازی	۸
۱.۴.۲ استاندارد ERC20	۸
۲.۴.۲ استاندارد ERC721	۹
۳.۴.۲ استاندارد ERC1155	۹
فصل ۳: آشنایی با ابزارهای توسعه	۱۱
۱.۳ ابزارهای ساده	۱۱
۲.۳ کیف پول متامسک	۱۲

۳.۳	چارچوب‌ها و کتابخانه‌ها	۱۲
۱.۳.۳	چارچوب Truffle	۱۳
۲.۳.۳	کتابخانه OpenZeppelin	۱۴
۳.۳.۳	کتابخانه Web3JS	۱۵
۴.۳	شبکه محلی برای توسعه	۱۵
<b>فصل ۴: پیاده‌سازی</b>		
۱.۴	نوشتن کد قرارداد	۱۹
۱.۱.۴	نیازمندی‌های قرارداد هوشمند	۱۹
۲.۱.۴	ارث‌بری	۲۰
۳.۱.۴	توجه به هزینه تراکنش و نوع توابع	۲۰
۴.۱.۴	جزئیات فنی پیاده‌سازی	۲۱
۲.۴	نوشتن و اجرای تست‌ها	۲۳
۳.۴	بارگذاری قرارداد روی شبکه تستی راپستن <sup>۸</sup>	۲۵
۱.۳.۴	یافتن آدرس یکی از نودهای شبکه برای ارسال تراکنش بارگذاری قرارداد به آن	۲۵
۲.۳.۴	اضافه شدن اطلاعات شبکه مورد نظر به تنظیمات ترافل	۲۵
۳.۳.۴	آماده شدن mnemonics	۲۶
۴.۳.۴	استفاده از کیف پول ایجاد شده در تنظیمات ترافل	۲۷
۵.۳.۴	نصب کیف پول hdwallet	۲۷
۶.۳.۴	انتخاب شبکه اضافه شده	۲۸
۷.۳.۴	بررسی آدرس کیف پول و موجودی آن	۲۸
۸.۳.۴	بارگذاری قرارداد هوشمند روی شبکه بلاکچین	۲۹
۹.۳.۴	اطمینان از صحت بارگذاری قرارداد هوشمند	۲۹
۴.۴	توسعه واسط کاربری، اتصال به قرارداد هوشمند و فرآیند بارگذاری	۲۹
۵.۴	داکرایز شدن، پایپلاین‌ها و گیت	۳۱

<sup>۸</sup>Ropsten

۱.۵.۴	داکرایز شدن تست‌های قراردادهوشمند	۳۱
۲.۵.۴	اجرای خودکار تست‌های قرارداد	۳۲
۳.۵.۴	بارگذاری خودکار واسط کاربری	۳۲
<b>فصل ۵: دست‌آوردها، پیشنهادها، محدودیت‌ها</b>		
۱.۵	دست‌آوردها	۳۵
۱.۱.۵	یادگیری	۳۵
۲.۱.۵	پلتفرم ایجاد شده	۳۶
۳.۱.۵	ساخت محیط توسعه سریع و خودکار	۳۶
۲.۵	پیشنهادها	۳۶
۱.۲.۵	استفاده از استانداردها	۳۷
۲.۲.۵	استفاده از ERC1155 به جای ERC721	۳۷
۳.۲.۵	ساخت محیط توسعه از شروع کار	۳۷
۳.۵	محدودیت‌ها	۳۸
۱.۳.۵	استفاده از Drizzle	۳۸
۲.۳.۵	ورژن‌های مختلف ابزارها	۳۸
۳.۳.۵	هزینه تراکنش‌های شبکه اصلی اتریوم	۳۸
۴.۳.۵	عدم وجود راهنما و مستندات کافی	۳۸

مراجع

اول

# فصل ۱

## مقدمه و بیان مسئله

### ۱.۱ مقدمه

در یک دهه اخیر محبوبیت رمزارزها در میان مردم به شدت افزایش داشته است. رمزارزها توکن هایی تعویض پذیر هستند به این معنی که تفاوتی میان دو توکن یک رمزارز وجود ندارد، مانند پول فیزیکی<sup>۱</sup> که ارزش یک هزار تومانی با یک هزار تومانی دیگر تفاوتی ندارد.

اما در دنیای واقعی تنها مالکیت پول نیست که اهمیت دارد، بلکه یک فرد می تواند خودرو، خانه، بلیت هواپیما و دیگر دارایی هایی داشته باشد که یکتا هستند و با هیچ دارایی دیگری دقیقاً یکسان نیستند. مثلاً یک بلیت هواپیما برای تاریخ و ساعتی خاص برای شماره پروازی خاص از یک مبدا مشخص به یک مقصد مشخص است و شماره صندلی یکتایی نیز دارد. پس هیچ دو بلیت هواپیمایی دقیقاً یکسان نیستند، بر خلاف دو بیتکوین که کاملاً یکسان هستند، ارزش برابری دارند، و تعویض پذیر هستند.

کاربردهای توکن های تعویض ناپذیر بیشمار است و در حال حاضر فقط قسمت اندکی از کاربردهایی که می توانند داشته باشند را پاسخ گفته اند. در این پروژه یک پلتفرم<sup>۲</sup> می سازیم که ساخت و انتقال توکن های تعویض ناپذیر را برای عموم در دسترس تر و آسان تر می کند. همچنین یکی از اهداف انجام این پروژه آشنایی با تکنولوژی ها، استانداردها و فرایندهای توسعه این توکن هاست.

---

<sup>۱</sup>Fiat Money

<sup>۲</sup>Platforms



## ۲.۱ شرح مسئله و روش انجام آن

پروژه تعریف شده توسعه یک پلتفرم برای ساخت<sup>۳</sup> و انتقال توکن‌های تعویض ناپذیر 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<sup>۴</sup>، استانداردها و فرایند تست و بارگذاری آشنا شویم. برای انجام این مراحل در قدم اول نحوه توسعه اپلیکیشن‌های غیر متمرکز و برتری‌های نوشتن پروژه به صورت متن باز ذکر می‌شود، سپس چارچوب‌ها و ابزارهایی که برای ساخت یک اپلیکیشن غیر متمرکز به توسعه دهنده کمک می‌کنند معرفی می‌شوند و نحوه استفاده از آن‌ها شرح داده می‌شود.

سپس فرایند توسعه آغاز می‌شود، استانداردهای موجود برای نوشتن یک قرارداد برای توکن‌های تعویض ناپذیر 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<sup>۵</sup> 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<sup>۶</sup> بارگذاری می‌شود تا در دسترس عموم کاربرها قرار بگیرد.

برای داکرایز<sup>۷</sup> کردن تست‌های قرارداد هوشمند<sup>۸</sup> یک ایمیج داکر<sup>۹</sup> ترافل<sup>۱۰</sup> 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.

## ۳.۱ اهداف کلی تحقیق

اهداف این تحقیق را می‌توان به دو دسته تقسیم‌بندی نمود.

<sup>3</sup>Mint

<sup>4</sup>Frameworks

<sup>5</sup>Testnets

<sup>6</sup>Github Pages

<sup>7</sup>Dockerize

<sup>8</sup>Smart Contracts

<sup>9</sup>Docker Images

<sup>10</sup>Truffle

### ۱.۳.۱ گسترش کاربردهای توکن‌های تعویض ناپذیر

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 کاربردهای روزانه مانند بلیت سینما و هواپیما، تا مالکیت هر نوع دارایی واقعی یا مجازی.

با توجه به نحوه کار اکثر قراردادهای توکن‌های تعویض ناپذیر، 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 روی شبکه بلاکچین دارد.

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

### ۲.۳.۱ یادگیری

هدف دیگر انجام این پروژه یادگیری است. با توجه به رشد سریع و تازگی استفاده از تکنولوژی‌های بلاکچین و توکن‌های تعویض ناپذیر، با وجود تلاش برای ایجاد منابع یادگیری مناسب همچنان فضاهاى خالی، کمبودها و نیازمندی‌هایی وجود دارد که باید پاسخ گفته شوند. در طی انجام این پروژه با ابزارها، کتابخانه‌ها، چارچوب‌ها و استانداردهای نوشتن قراردادهای هوشمند آشنا می‌شویم، می‌آموزیم که هر یک چگونه کار میکنند و چگونه می‌توانند به توسعه دهنده کمک کنند.

## ۴.۱ ساختار پایان‌نامه

پس از این مقدمه، در فصل ۲ مفاهیم اولیه توسعه اپلیکیشن 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 می‌شود 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 برتری‌های متن‌باز بودن قراردادهای هوشمند

دلایل زیادی برای متن‌باز نوشتن قراردادهای هوشمند وجود دارد، در ادامه تعدادی از این دلایل توضیح داده می‌شود.

دلیل اول، بلاک‌چین‌ها محرمانگی<sup>۱</sup> ندارند، همه‌ی نودهای شبکه برای اجرای کد قرارداد هوشمند باید حداقل به بایت‌کدها<sup>۲</sup> قرارداد هوشمند دسترسی داشته باشند و این بایت‌کدها در کاوشگرهای بلاکچین نیز وجود دارند، همچنین دیکامپایلر<sup>۳</sup>هایی وجود دارند که از بایت‌کدهای قرارداد هوشمند کد سالی‌دیتی آن را به دست می‌آورند. پس در نتیجه تلاش برای مخفی کردن کدهای قرارداد هوشمند بیهوده خواهد بود.

دلیل دوم، اصلی‌ترین مزیت اپلیکیشن‌های غیرمتمرکز نسبت به اپلیکیشن‌های متمرکز عدم نیاز به اعتماد است، کاربرها می‌توانند 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 پایین‌تر از اپلیکیشن‌های متمرکز هست، پس امکان این که با پیدا شدن هر مشکل بتوان به سرعت آن را درست کرد کمتر وجود دارد و مسئله امنیت بسیار اهمیت دارد. متن‌باز نوشتن قرارداد هوشمند باعث می‌شود چشم‌های بیشتری کدهای قرارداد را

---

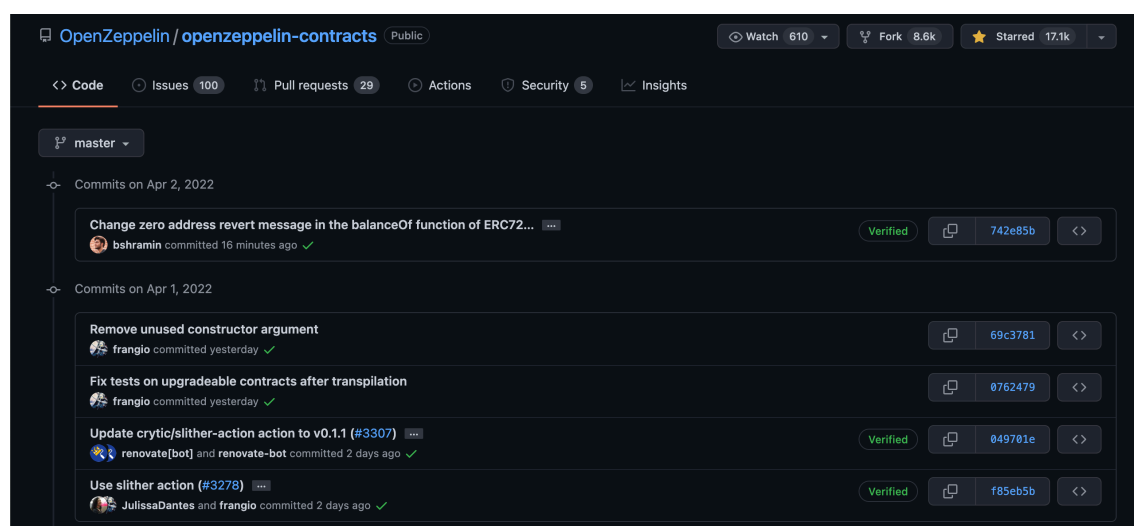
<sup>1</sup>Confidentiality

<sup>2</sup>bytecodes

<sup>3</sup>

بخوانند و مشکلات احتمالی سریعتر مشخص و رفع شوند. تعداد زیادی از این پروژه‌ها 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.

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



شکل ۱.۲: در طی انجام پروژه مرج ریکوئستی روی OpenZeppelin باز شد که در همان روز مرج شد.

## ۲.۲ آشنایی با مفهوم توکن تعویض ناپذیر

شروع رمزارزها با توکن‌های تعویض پذیر بود، مفهوم تعویض پذیری 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 یک بیت‌کوین دیگر هیچ تفاوتی ندارد.

اما توکن‌های تعویض ناپذیر اینگونه نیستند، هر یک منحصر به فرد است و جابه‌جا کردن آن‌ها با یکدیگر

تغییر ایجاد می‌کند، در دنیای واقعی خانه می‌تواند مثال خوبی از یک دارایی تعویض ناپذیر 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 NFT اختصاص دهد. به این صورت صاحب خانه به جای سند یک توکن NFT دارد که مشخص می‌کند که دارایی متعلق به اوست، و فروش خانه به راحتی انتقال آن NFT به شخص دیگری است.

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 Address Contract و Token ID باعث می‌شود که هر توکن یکتا باشد.

## ۳.۲ کاربردها، حال و آینده

کاربرد های NFT 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 چیزی، برای مثال همایشی برگزار می‌شود که فقط دارندگان های NFT یک قرارداد هوشمند می‌توانند به آن وارد شوند.

معروف‌ترین پلتفرم معاملاتی این توکن‌ها OpenSea است که می‌توان در آن توکن‌های موجود را مشاهده کرد و یک توکن را توسط مزایده خرید یا به فروش گذاشت. OpenSea در حال حاضر از قراردادهای شبکه‌های اتریوم و سولانا پشتیبانی می‌کند. دیگر شبکه‌ها نیز معمولاً پلتفرم‌های خود را دارند، مانند شبکه Atom که در آن از پلتفرم Stargaze برای معامله های NFT استفاده می‌شود.

کاربردهای NFT ها 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 آیتمی در یک بازی و حتی دامنه‌های اینترنتی همه می‌توانند به NFT تبدیل شوند. مزایای تبدیل این موارد به NFT قابلیت نگهداری آسان‌تر، قابلیت فروش و انتقال راحت‌تر، امنیت بیشتر، آزادی در تراکنش‌ها و آشکار بودن مالکیت دارایی بر همگان است.

## ۴.۲ قراردادهای هوشمند و استانداردسازی

اکثر قراردادهای هوشمند قابلیت‌هایی مشابه با یکدیگر دارند، برای مثال گروهی از قراردادهای هوشمند توکن‌های تعویض پذیر دارند و گروه توکن‌های تعویض ناپذیر. از طرفی اپلیکیشن‌هایی مانند کیف پول‌های دیجیتال، پلتفرم‌های معاملاتی و صرافی‌های نیاز دارند که بتوانند دارایی‌های کاربر اعم از توکن‌های تعویض پذیر و تعویض ناپذیر را ببینند، به همین دلیل باید نحوه صحبت کردن با قراردادهای هوشمند را بدانند.

برای ساده‌تر کردن این فرایند و همسان‌سازی اینترفیس این قراردادهای هوشمند استانداردهایی تعریف شده است که با استفاده از این استانداردها هم فرایند توسعه اسمارت کانترکت آسان‌تر خواهد شد و هم ارتباط میان قراردادهای هوشمند و اپلیکیشن‌های دیگر مانند کیف پول‌ها، پلتفرم‌های معاملاتی و ... آسان‌تر خواهد شد.

از نمونه‌های معروف این استانداردها ERC20 برای قراردادهایی با توکن‌های تعویض پذیر و ERC721 برای قراردادهایی با توکن‌های تعویض ناپذیر است. در این پروژه 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 و یک قرارداد با این استاندارد می‌تواند هم توکن‌های تعویض پذیر و هم تعویض ناپذیر داشته باشد.

برای استفاده از این استانداردها از پکیج‌های متن بازی استفاده می‌شود که این استانداردها را پیاده‌سازی کرده‌اند و از آن‌ها در قراردادی که نوشته می‌شود ارث‌بری می‌شود، یکی از بهترین پیاده‌سازی‌های این استانداردهای توسط اپن‌زپلین<sup>۴</sup> انجام شده است که در این پروژه نیز از همین پیاده‌سازی استفاده می‌شود.

### ۱.۴.۲ استاندارد ERC20

این استاندارد مناسب توکن‌های تعویض پذیر است. اینترفیسی تعریف می‌کند که نیازهای قراردادهایی با توکن‌های تعویض پذیر را برطرف کند و نحوه تعامل برقرار کردن با آن‌ها را یکسان گرداند. در این استاندارد فقط می‌توان یک نوع توکن تعویض پذیر 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<sup>۵</sup> یا اپن‌زپلین<sup>۶</sup> مشاهده کرد.

<sup>۴</sup>OpenZeppelin

<sup>۵</sup><https://ethereum.org/en/developers/docs/standards/tokens/erc-20>

<sup>۶</sup><https://docs.openzeppelin.com/contracts/4.x/api/token/erc20>

## ۲.۴.۲ استاندارد ERC721

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

همچنین متدی وجود دارد که می‌تواند آیدی 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 بدهند که توکن‌ها را به شخص دیگری ارسال کند.

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

اطلاعات دقیق‌تر در مورد این استاندارد را نیز می‌توان در وبسایت اتریوم<sup>۷</sup> یا این‌زپلین<sup>۸</sup> مشاهده کرد.

## ۳.۴.۲ استاندارد ERC1155

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

- داشتن توکن‌های NFT 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.

<sup>۷</sup><https://ethereum.org/en/developers/docs/standards/tokens/erc-721>

<sup>۸</sup><https://docs.openzeppelin.com/contracts/4.x/api/token/erc721>



یک مثال از کاربردی که به این قابلیت‌ها نیاز دارد می‌تواند یک بازی مثل مونوپولی باشد که در آن هر کاربر مقداری پول دارد که در واقع یک توکن تعویض پذیر هست، به عنوان دارایی چند خانه دارد که به عنوان توکن‌های تعویض ناپذیری 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 تعویض ناپذیر باشد تعداد آن یک قرارداد 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<sup>۹</sup> یا اپن‌زپلین<sup>۱۰</sup> مشاهده کرد.

---

<sup>۹</sup><https://ethereum.org/en/developers/docs/standards/tokens/erc-1155>

<sup>۱۰</sup><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 Solidity<sup>۱</sup> استفاده شده است. این پلاگین با یافتن اشتباه‌ها پیش از کامپایل و راهنمایی در نوشتن کد قرارداد کمک شایانی به افزایش سرعت توسعه می‌کند.

#### • ورژن کنترل

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

---

<sup>۱</sup><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 جاوااسکریپت هستند، به ابزارهای توسعه اپلیکیشن‌های جاوااسکریپت برای توسعه سالی‌دیتی نیاز است. ابزارهایی مانند Node برای کامپایل کردن برنامه‌های جاوااسکریپت و npm که مدیریت پکیج‌های جاوااسکریپتی 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 غیر متمرکز 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<sup>۲</sup> 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 غیر متمرکز 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 غیر متمرکز متن‌باز هستند، سرعت رشد و تکامل بالایی دارند و به کمک توسعه‌دهندگان<sup>۳</sup> این

<sup>۲</sup>Private key

<sup>۳</sup>Developers

حوزه، هر روز نسبت به روز گذشته پیشرفت می‌کنند.

برای توسعه این پروژه از چارچوب Truffle<sup>۴</sup>، کتابخانه‌ی OpenZeppelin<sup>۵</sup>، کتابخانه‌ی Web3JS<sup>۶</sup> استفاده شده است. در این قسمت به توضیح هر یک از این موارد پرداخته می‌شود.

### ۱.۳.۳ چارچوب Truffle

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، 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` و یا `truffle console` می‌توان وارد خط فرمان ترافل شد.

دستورات لازم برای اجرای تست‌ها، کامپایل کردن قراردادهای 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 Ganache و Drizzle است.

<sup>۴</sup><https://trufflesuite.com>

<sup>۵</sup><https://openzeppelin.com/contracts>

<sup>۶</sup><https://github.com/ChainSafe/web3.js>

```

+ back git:(main) * truffle develop
Truffle Develop started at http://127.0.0.1:9545/

Accounts:
(0) 0x00eade7f45de5837119f5dd65fe63e58dcf8f7138
(1) 0x09ef5651770dcb33d926a1b75e10b2944924736
(2) 0x3e6cfcf6153d6dda9e2654afc8cfda499818c3e9
(3) 0xcefd7ce63b03aaccacd420828abe96acbe968d7
(4) 0xa84ab5be39670309d305e7aad3bcc347e75e9537
(5) 0x318913b83fe0c2d63ab29bb84e39b6e39ccd93ef
(6) 0xde2602e64a047482e4606af26c89abb97afe6bdf
(7) 0x4d85fb20085db61ab33e8159bfe88f3e1a7a1279
(8) 0xa031538284a6910cf832e25b2ec8105f52a87b13
(9) 0x3973a0026d8e807d0b9b5fb91c35a16ea990063a

Private Keys:
(0) 9453db0e1f3c1034f80a01b335b141e0d519f21aced3dbb7d2da54f37d773a6d
(1) bf85b2b427c669793da84d9c98e78d0fabd8676938d5c87cc01de50d42e235b7
(2) eef37f33a88597e7434be9def2e22594047c678a451a7f9581e8378a839e19c0
(3) fdd92e2593bb15465bb7d6b61b3894c9fe50acfaec60327ec7c1c2f378664dc8
(4) 0c50a6d545d747b33db3744321eea3e212400c4cd9497a0211fe4bd729a90c9f
(5) 28f97ce0d95990b234c50438ee46fd27461cf2ecf429131c9aa4176c1c05a0ab
(6) e6eb577c2aa69993aeab80ad275c346c6cf9b8506e8ab29cc69234744593a622
(7) 039c1150cf5eb82756320f0c904035cf0017da2a87c4c1d2dd057a39b3f0c452
(8) a78d582b2fad08f58f59680ade3de3e3e8a9e609fe101c8df850e5444c1b59c0
(9) 77e77e250d6551b2d7af2b2088e350e84577ca39debd66c41136ab7a90e69f76

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

### ۲.۳.۳ کتابخانه OpenZeppelin

یکی از معروف‌ترین کتابخانه‌های قراردادهای هوشمند و استانداردهایشان است. قراردادهای و استانداردهای موجود در این کتابخانه کاملاً تست شده، داکيومنت شده، ایمن و پایه بسیاری از قراردادهای هوشمند بر بستر بلاکچین هستند. استانداردهای ذکر شده در این متن مانند، ERC20، ERC721، 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 این زپلین 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 قطعه کد زیر مشاهده می‌شود که کاپو چگونه از قرارداد ERC721 موجود در این 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 OpenZeppelin

### 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<sup>۷</sup> 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 غیرمتمرکز 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 امضا شدن 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 امضا و روی شبکه ارسال کند. طبیعتاً کیف پول کاربر برای انجام هر یک از این مراحل از کاربر درخواست تاییدیه می‌کند.

### ۴.۳ شبکه محلی برای توسعه

برای توسعه یک اسمارت کانترکت نیاز است که پس از هر تغییر کامپایل و روی یک شبکه بلاکچین بارگذاری شود، به نحوی که واسط کاربری اپلیکیشن و همچنین کیف پول متامسک بتوانند به آن متصل شوند. از شبکه

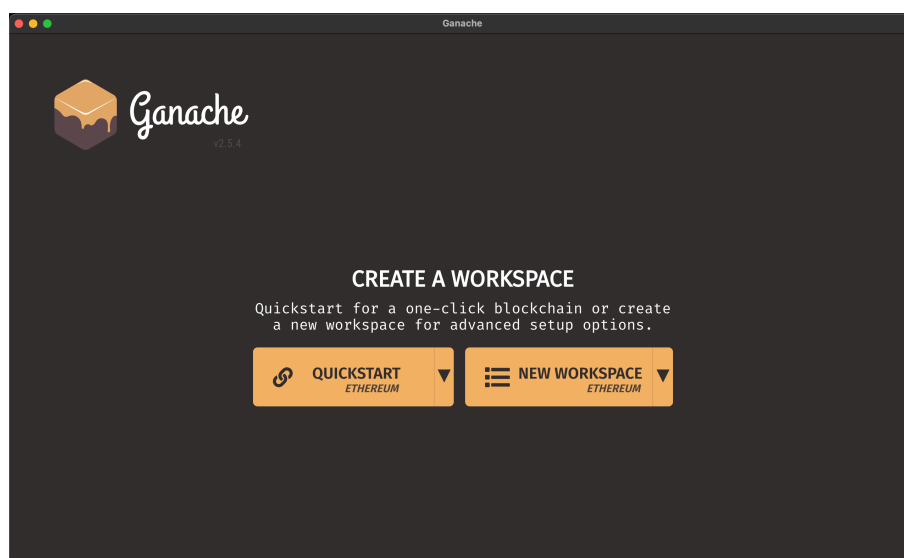
<sup>7</sup>State

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 زمان‌بر 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 بلافاصله 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. واسط کاربری 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 Ganache هست. این ابزار با توجه به این که متعلق به اکوسیستم Truffle هست به آسانی به آن متصل می‌شود و با اضافه کردن آدرس آن به شبکه‌های متامسک، این کیف پول هم به شبکه محلی متصل می‌شود. جزئیات ساخت شبکه محلی و اتصال ترافل و متامسک به آن به ترتیب زیر است.

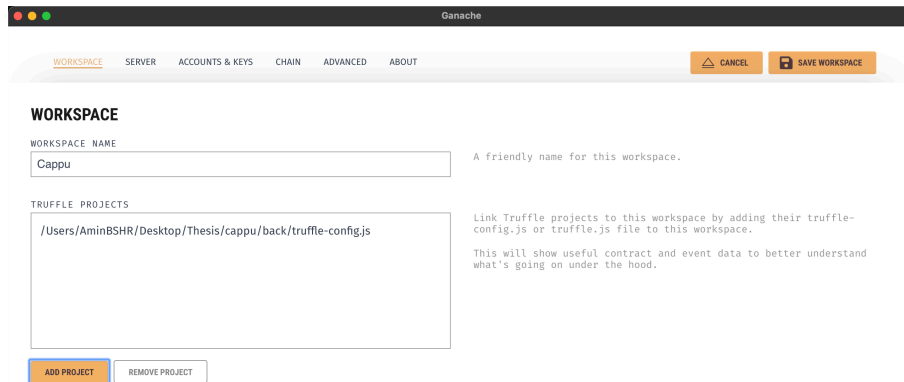
پس از نصب برنامه Ganache باید یک محیط توسعه Ethereum ساخته شود. برای انجام این کار گزینه New workspace (Ethereum) انتخاب می‌شود.



شکل ۴.۳: صفحه اول Ganache

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.



شکل ۵.۳: ساخت شبکه جدید در گاناچه

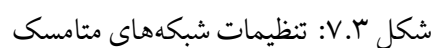
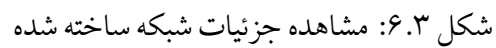
پس از انجام این مراحل محیط توسعه ساخته شده است و می‌توان جزئیات شبکه محلی را مشاهده کرد. از آنجایی که در مرحله قبل برای ساخت این محیط توسعه فایل `truffle-config.js` پروژه انتخاب شد، حال اگر دستورات `truffle console` یا هر دستور دیگری مانند `migrate` بدون انتخاب شبکه بلاکچین خاصی اجرا شود به صورت پیش‌فرض 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<sup>۸</sup> و سپس شبکه‌ها<sup>۹</sup> یک شبکه جدید با جزئیات زیر ساخته می‌شود، همانطور که در تصویر بالا ۶.۳ مشخص است اطلاعات شبکه محلی در صفحه اصلی Ganache قابل مشاهده هستند.

پس از ذخیره شبکه جدید کافیهست که برای توسعه شبکه Ganache انتخاب شود. همچنین باید یکی از آدرس‌هایی که در صفحه اصلی Ganache 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 Ganache انتخاب می‌شود و به کمک کلید اختصاصی نمایش داده شده کیف پول در متامسک وارد می‌شود.

<sup>۸</sup>Settings

<sup>۹</sup>Networks





## فصل ۴

# پیاده‌سازی

### ۱.۴ نوشتن کد قرارداد

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

#### ۱.۱.۴ نیازمندی‌های قراردادهوشمند

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 قراردادهوشمند 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.
- قراردادهوشمند کاپو از استاندارد ERC۷۲۱ پیاده‌سازی شده در کتابخانه اپن‌زپلین<sup>۱</sup> ارث‌بری می‌کند که یکی از معروف ترین 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 fee gas 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.

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

اگر متدی 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<sup>۲</sup> خودش دارد. اگر متدی 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 متد نیازی به پرداخت کارمزد تراکنش توسط کاربر ندارند، اما اگر در تعریف متدی 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 قراردادهای 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 آیدی هر توکن از hash 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 آیدی آن‌ها باید یکسان باشد و این امکان پذیر نیست زیرا آیدی توکن‌ها یکتاست. همچنین آیدی توکن‌ها دیگر ترتیبی نخواهند بود و ترتیب ساخت توکن‌ها مشخص نخواهد بود.

در یک قرارداد ERC۷۲۱ استاندارد فقط آیدی توکن‌ها ذخیره می‌شود. در کاپو علاوه بر آیدی توکن‌ها یک map از آیدی توکن‌ها به داده‌ی آن‌ها با نام tokenDatas نیز نگهداری می‌شود. همچنین در کاپو map دیگری نیز از آدرس به لیست توکن‌های آن آدرس با نام ownerTokens نگهداری می‌شود. متغیر اول کمک می‌کند که با داشتن آیدی 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 numberOfTokenHolders و numberOfMintedTokens نیز در کاپو نگه‌داشته می‌شوند که برای نمایش آمار استفاده از قرارداد در صفحه اصلی اپلیکیشن مورد استفاده قرار می‌گیرند.

<sup>2</sup>Scope

متد mint به نحوی نوشته شده است که برای عموم قابل استفاده باشد. پس از محاسبه hash داده‌ی توکن از آن به عنوان آیدی توکن استفاده می‌کند، توکن را می‌سازد و متغیرهای tokenDatas و numberOfMintedTokens را بروزرسانی می‌کند.

```

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 ERC۷۲۱ به نحوی بازنویسی<sup>۳</sup> شده است که پس از هر انتقال توکن با بررسی آدرس‌های مبدا و مقصد، متغیرهای numberOfTokenHolders و numberOfMinted- Tokens و ownerTokens را بروزرسانی کند.

```

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

<sup>3</sup>Overwrite

متد جدیدی با نام `getUserTokens` نیز نوشته شده است که در استاندارد ERC۷۲۱ به صورت پیش فرض وجود ندارد. این متد با گرفتن یک آدرس و استفاده از `ownerTokens` و `tokenDatas` دو خروجی برمی گرداند، لیستی از آیدی توکن‌های آدرس و لیستی از داده‌های توکن‌های آدرس.

```

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 پیش فرض امکان حذف یک داده از یک آرایه با داشتن مقدار آن را ندارد، عدم وجود این قابلیت هزینه‌بر بودن آن است، در سالیدیتی توسعه دهندگان به استفاده از `map` و دوری از `array` ها تشویق می‌شوند. اما برای نمایش نحوه ارث‌بری از دو یا چند قرارداد پدر، برای کاپو یک قرارداد به نام `Helper` نوشته شد که این قابلیت را فراهم می‌کند. کاپو علاوه بر ERC721 از قرارداد `Helper` نیز ارث‌بری می‌کند.

## ۲.۴ نوشتن و اجرای تست‌ها

پیش‌تر اشاره شد که از مزیت‌های ارث‌بری از کتابخانه‌های متن‌باز معروف این است که احتمال وجود خطا 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 فرزند تمرکز 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 متد `mint` یک توکن ساخته می‌شود و سپس

```

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

با فراخوانی متد `balanceOf` دارایی 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 زیر 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 ۵.۴: 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 زیر 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--mpJl9p9Ttlws
> 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

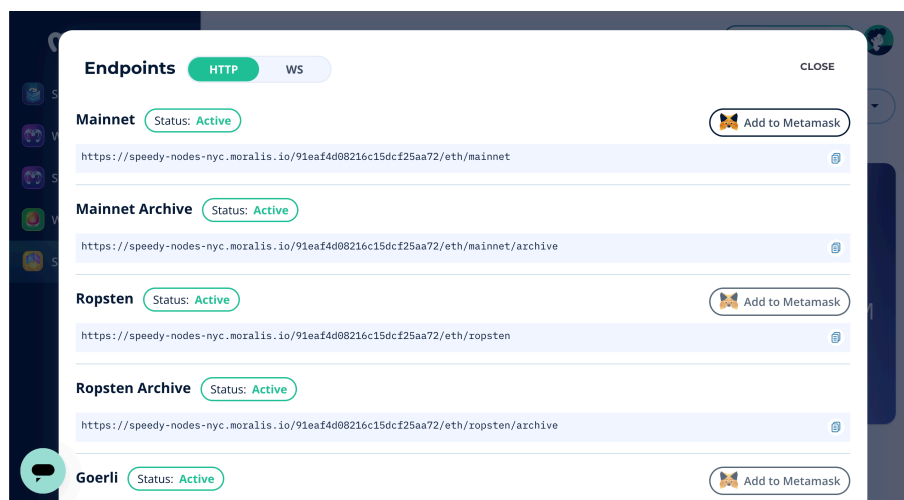
آدرس نودهای یک شبکه بلاکچین همه به صورت عمومی در دسترس هستند زیرا نودها باید بتوانند یکدیگر را ببینند. راه‌های زیادی برای به دست آوردن آدرس یک نود وجود دارد. یکی از آسان‌ترین راه‌های به دست آوردن آدرس یکی از نودهای شبکه مراجعه به وبسایت ماینر است. برای این پروژه از وبسایت Moralis<sup>۴</sup> برای پیدا کردن آدرس نود 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، فایل با نام truffle-config.js ساخته می‌شود. تنظیمات مربوط به ترافل در این فایل نوشته شده‌است. برای این که ترافل شبکه مورد نظر را

<sup>۴</sup> <https://moralis.io>





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

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 زیر 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: 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

### ۳.۳.۴ آماده شدن mnemonics

برای انجام این پروژه به کمک دستور npm mnemonics یک آدرس تستی ساخته می‌شود. این دستور، mnemonics متناسب با این آدرس را به عنوان خروجی می‌دهد. دقت کنید که برای بارگذاری روی شبکه اصلی<sup>۵</sup> حتماً باید از mnemonics مربوط به یک کیف پول واقعی استفاده شود و اطلاعات آن در اختیار کسی قرار نگیرد.

<sup>۵</sup>Mainnets

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

شکل ۹.۴: ایجاد mnemonics تستی

#### 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 mnemonics یا کلید خصوصی آن دسترسی داشته باشد. به این منظور فایلی با نام secrets.json در دایرکتوری اصلی برنامه ساخته می‌شود و mnemonics کیف پول به شکل زیر در آن قرار داده می‌شود.

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

شکل ۱۰.۴: قراردادن mnemonics در فایل secrets.json

سپس در تنظیمات ترافل باید ذکر شود که می‌تواند آدرس کیف پول 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 کیف پول hdwallet

ترافل برای استفاده از mnemonics کیف پول ما نیاز به نصب پکیج hdwallet-provider دارد، این پکیج کاربری‌های یک کیف پول 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 کیف پول در تنظیمات ترافل در فایل `truffle-config.js` ذکر می‌شود که از این کیف پول استفاده شود.

```

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

```

شکل ۱۲.۴: استفاده از کیف پول hdwallet در تنظیمات ترافل

### ۶.۳.۴ انتخاب شبکه اضافه شده

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

```

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

```

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

### ۷.۳.۴ بررسی آدرس کیف پول 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 آدرس های کیف پول از دستور زیر در خط فرمان ترافل استفاده می‌شود.

```

truffle(ropsten)> await web3.eth.getAccounts()
[
  '0xF51f5f41BfA8ADa57a43862cBc18dA4750AecB4c',
  '0x909ebC92395FC4335c35894C7DDc8bFFFDCEf06',
  '0x48156708DF687C7a8F97C951b5E734E132e891D1',
  '0xF1C6c91D80032528e2C01F73DAd588D11DA0f17d',
  '0xA6f899d10B4E1c1195AFD1C6f29E4e539C828450',
  '0xB63191Dd13637c024C7F1F339F254F0F13d4bB34',
  '0x1699Ba468F7E5af64f510B323537bbcd107373F9',
  '0x6eDd855A6D2d3De5D96749e1bD3E9580c33468E7',
  '0x8A97C0bfc3086DFcd9E1B25D69A1A238A1290BE6',
  '0xc4838dF4d46862d1226BDC409EbE8395cA6fE703'
]

```

شکل ۱۴.۴: دریافت آدرس های کیف پول در خط فرمان ترافل

برای دریافت مانده حساب آدرس، دستور زیر در خط فرمان ترافل اجرا می‌شود.

```
truffle(ropsten)> await web3.eth.getBalance("0xf51f5f418fA8ADa57a43862c8c18dA4750Aec84c")
'790887817599784390'
truffle(ropsten)>
```

شکل ۱۵.۴: دریافت موجودی کیف پول 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 صحت بارگذاری قرارداد هوشمند

س از اتمام بارگذاری قرارداد هوشمند برای اطمینان از به درستی انجام شدن فرآیند بارگذاری قرارداد، می‌توان از جستجوگرهای شبکه<sup>۶</sup> بلاکچین استفاده کرد. برای مثال قرارداد هوشمند کاپو بر روی شبکه راپستن بارگذاری شده است، که با رفتن به وبسایت اتراسکن<sup>۷</sup> و قراردادن آن روی شبکه راپستن می‌توان قرارداد بارگذاری شده و تراکنش‌های آن را مشاهده کرد.

## 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 که کمک می‌کند در زمان کوتاه بتوان ظاهری زیبا و یکدست در اپلیکیشن ایجاد کرد و کتابخانه Web3JS که واسط کاربری را به کیف پول کاربر و شبکه بلاکچین متصل می‌کند، همه‌ی قابلیت‌های مورد نیاز برای توسعه یک واسط کاربری زیبا و کارآمد را در اختیار توسعه دهنده قرار می‌دهد.

<sup>۶</sup>Block Explorers

<sup>۷</sup> <https://etherscan.io>

The screenshot shows the Etherscan interface for a contract on the Ropsten Testnet. The contract address is 0xf1359760a1b37a9f8d85341E72cEF0644DB660A1. The contract overview shows a balance of 0 Ether. The transactions table lists the latest 13 transactions, including a 'Safe Transfer From' and two 'Mint' transactions.

Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0xd207a3f62e89fcd054c...	Safe Transfer From	12147200	7 days 19 hrs ago	0xf51f541bfa8ada57a43...	0xf1359760a1b37a9f8d8...	0 Ether	0.000353810001
0xece8b0c2be5bd13534...	Mint	12147179	7 days 19 hrs ago	0xf51f541bfa8ada57a43...	0xf1359760a1b37a9f8d8...	0 Ether	0.021437500002
0xd754232c700bd4cb75...	Mint	12147178	7 days 19 hrs ago	0xf51f541bfa8ada57a43...	0xf1359760a1b37a9f8d8...	0 Ether	0.000273017508

شکل ۱۶.۴: مشاهده قرارداد کاپو در Etherscan روی شبکه راپستن

در پوشه اصلی واسط کاربری فایللی 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 دقت 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 آداپتور میان Web3JS و کد کاپو عمل می‌کند. تمامی قابلیت‌های مورد نیاز مانند اتصال به کیف پول و ورود<sup>۸</sup> کاربر، خروج<sup>۹</sup> کاربر، گرفتن آدرس و شبکه‌ی کیف پول و ... در این فایل انجام می‌شود.

واسط 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 واسط 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، جابه‌جایی 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.

<sup>۸</sup>Login

<sup>۹</sup>Logout

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

این قابلیت گیت‌هاب در واقع به این صورت عمل می‌کند که یک برنچ به نام gh-pages در repository 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 متناسب با آدرس repository بارگذاری 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 قراردادهوشمند

برای سرعت بخشیدن به توسعه قراردادهوشمند، 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 قراردادهوشمند 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 ایميج پایه داکر نوشته شد که داکر فایل آن را می‌توان در گیت‌هاب<sup>۱۰</sup> مشاهده کرد، همچنین این ایميج داکر در داکرهاب<sup>۱۱</sup> نیز پوش شد.

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

#### ۲.۵.۴ اجرای خودکار تست‌های قرارداد

با داشتن داکر فایلی 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 مرج ریپوژست به برنچ `master` و با پوش شدن یک کامیت در برنچ `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 بخشیدن 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 پوش شدن تغییرات جدید روی برنچ 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 برنچ دیگری به نام `gh-pages` پوش کند. پس این استیج پایپلاین نیاز به دسترسی پوش کردن روی ریپازیتوری دارد.

<sup>10</sup> <https://github.com/bshramin/truffle-docker>

<sup>11</sup> <https://hub.docker.com/r/aminbshr/truffle>

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

## ۲.۱.۵ پلتفرم ایجاد شده

قرارداد هوشمند نوشته شده در این پروژه، کاپو، با عملکرد کامل بر بروی شبکه تستی راپستن بارگذاری شد و امکانات لازم برای دسترسی عموم مردم به روشی آسان و ارزان به توکن‌های تعویض ناپذیر 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.

## ۳.۱.۵ 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 ایمج ترافل نوشته شد، کد آن به صورت متن‌باز بر روی گیت‌هاب بارگزاری و ایمج 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.

## 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 ببخشد.

## ۱.۲.۵ استفاده از استانداردها

خوشبختانه 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 ERC1155 به جای ERC721

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 Drizzle

در ابتدای انجام پروژه سعی شد که برای برقراری ارتباط رابط کاربری با قرارداد هوشمند از دریزل<sup>۱</sup> 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<sup>۲</sup> 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 حتما 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 داکر 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 راهنما و مستندات کافی

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

<sup>1</sup>Drizzle

<sup>2</sup>Community

## مراجع





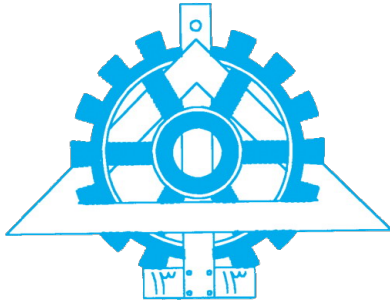


## **Abstract**

This thesis studies on writing projects, theses and dissertations using tehran-thesis class. It ...

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





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 Graduate Studies Office  
In partial fulfillment of the requirements for  
The degree of Master of Science  
in Computer Engineering - Software Engineering

By:

**Amin Bashiri**

Supervisor:

**Dr. Ehsan Khamespanah**

June 2022