



دانشکده مهندسی

پایان نامه کارشناسی ارشد
گروه مهندسی کامپیوتر

تحلیل، طراحی و پیاده سازی وب سایت مربوط به زنجیره تامین و شبکه
جهانی حمل و نقل - بخش بلاکچین

نگارنده:

سارا بلوری بزاز

استاد راهنما:

دکتر عباس رسول زادگان

زمستان ۱۴۰۱



اصالت نامه

فرم ارزشیابی

تقدیم به

مقدس‌ترین واژه‌ها در لغت‌نامه دلم

مادر مهربانم که زندگیم را مدیون مهر و عطوفت او هستم

پدرم، مهربانی مشفق، بردبار و حامی

تقدیر و تشکر

حمد و سپاس سزاوار خداوندی است که مرا نعمت هستی بخشید و در مسیر آموختن علم قرار داد. در این

مسیر با اساتیدی فرهیخته، صبور و با اخلاق آشنایم ساخت. هر چند در مقام قدردانی از زحمات ایشان زبان

قاصر و دست ناتوان است، اما بر خود لازم می‌دانم از زحمات و راهنمایی‌های استاد گرانقدر جناب آقای دکتر

عباس رسول‌زادگان قدردانی و تشکر نمایم چرا که بدون راهنمایی‌ها و دلسوزی‌های ایشان گردآوری این

پایان‌نامه امکان‌پذیر نبود. همچنین از راهنمایی‌ها و کمک‌های همه اعضای محترم آزمایشگاه کیفیت نرم‌افزار

نیز صمیمانه کمال تشکر و قدردانی را دارم.



بسمه تعالی

مشخصات رساله / پایان نامه تحصیلی دانشجویان

دانشگاه فردوسی مشهد

عنوان رساله / پایان نامه: تحلیل، طراحی و پیاده سازی وب سایت مربوط به زنجیره تامین و شبکه جهانی حمل و

نقل - بخش بلاکچین

نام نویسنده: سارا بلوری بزاز

نام استاد(ان) راهنما: جناب آقای دکتر عباس رسول زادگان

نام استاد(ان) مشاور: --

دانشکده : مهندسی	گروه: کامپیوتر	رشته تحصیلی: نرم افزار
تاریخ تصویب:	تاریخ دفاع:	
مقطع تحصیلی: کارشناسی ارشد	تعداد صفحات:	

چکیده رساله / پایان نامه:

کلید واژه:	امضای استاد راهنما: دکتر عباس رسول زادگان
بلاکچین، ذخیره سازی امن داده، سیستم توزیع شده، قرارداد هوشمند، چارچوب Hyperledger Fabric	امضا

فهرست مطالب

۱- مقدمه	۱۲
۲- پیشینه	۱۳
۳- راهکار پیشنهادی	۱۷
۳-۱- پیاده‌سازی پروژه - سمت کاربر	۱۷
۳-۱-۱- پیاده‌سازی زیرخدمات‌های FCL و Chartering	۱۹
۳-۱-۲- پیاده‌سازی زیرخدمت Air	۲۹
۳-۲- پیاده‌سازی پروژه - سمت سرور	۳۴
۳-۲-۱- پیاده‌سازی زیرخدمات‌های FCL و Chartering	۳۴
۳-۲-۲- پیاده‌سازی زیرخدمت Air	۳۵
۴- ارزیابی	۴۲
۵- نتیجه‌گیری و کارهای آتی	۴۲
۶- راهنمای فنی	۴۳
۶-۱- راهنمای فنی - فرانت وبسایت Cayload	۴۳
۶-۲- راهنمای فنی - سمت سرور	۴۴
۶-۲-۱- زیرخدمت FCL و Chartering - سمت سرور	۴۴
۶-۲-۲- زیرخدمت Air - سمت سرور	۴۵
۶-۳- راهنمای فنی - سمت مشتری	۴۹
مراجع	۶۰
پیوست	۶۰

فهرست شکل‌ها

شکل ۱-۳: نمایی از نرم‌افزار بلاکچین	۱۹
شکل ۲-۳: نمونه‌ای از مفاد قرارداد FCL در وبسایت	۲۰
شکل ۳-۳: نمودار مورد کاربرد ریزخدمت FCL	۲۱
شکل ۴-۳: صفحه ورود به نرم افزار	۲۲
شکل ۵-۳: بارگذاری/ذخیره‌سازی کلید خصوصی	۲۶
شکل ۶-۳: صفحه نمایش لیست قراردادها	۲۷
شکل ۷-۳: نمایی از اطلاعات قرارداد در قالب PDF	۲۷
شکل ۸-۳: نمودار کلاس زیرخدمت FCL	۲۸
شکل ۹-۳: نمایی از نرم افزار - زیرخدمت Air	۳۰
شکل ۱۰-۳: نمودار مورد کاربر زیرخدمت Air	۳۱
شکل ۱۱-۳: نمودار کلاس زیرخدمت Air	۳۳
شکل ۱۲-۳: شماتیک روابط بین اجزای سازنده سمت سرور	۳۶
شکل ۱-۶: سازنده کلاس Cayload در قرارداد هوشمند	۴۷
شکل ۲-۶: تابع configNetwork جهت برقرار ارتباط با شبکه بلاکچین	۴۸
شکل ۳-۶: پوشه‌بندی سمت مشتری	۴۹
شکل ۴-۶: سازنده کلاس Ui_landing_page()	۵۱
شکل ۵-۶: تابع اجرایی بعد از اتمام عملیات ورود کاربر	۵۱
شکل ۶-۶: تابع implement مربوط به زیرخدمت FCL و Chartering	۵۳
شکل ۷-۶: تابع implement مربوط به زیرخدمت Air	۵۴

فهرست جدول‌ها

جدول ۱-۳: خدمات و زیرخدمات در پروژه حمل و نقل	۱۸
جدول ۲-۳: نگاشت توابع به سند	۲۸
جدول ۳-۳: نگاشت توابع به سند	۳۳
جدول ۴-۳: نگاشت توابع قرارداد هوشمند نصب شده بر روی بلاکچین	۳۶
جدول ۵-۳: نگاشت توابع مربوط به برنامه واسط	۳۷
جدول ۶-۳: نگاشت توابع در راه‌اندازی شبکه بلاکچین	۳۸
جدول ۱-۶: آدرس فایل فرانت زیرخدمات‌ها	۴۴
جدول ۲-۶: آدرس فایل سرور زیرخدمات‌ها	۴۴
جدول ۳-۶: نگاشت توابع سمت سرور	۴۵
جدول ۴-۶: نگاشت توابع قرارداد هوشمند	۴۷
جدول ۵-۶: نگاشت توابع سمت سرور زیرخدمت Air	۴۹
جدول ۶-۶: نگاشت توابع اجرایی بعد از ورود موفقیت آمیز کاربر	۵۱
جدول ۷-۶: نگاشت توابع مربوط به دکمه‌های نرم افزار تحت دستکتاپ	۵۲
جدول ۸-۶: نگاشت توابع فرایند امضا کردن قرارداد در زیرخدمت FCL و Chartering	۵۳

۱- مقدمه

امروزه جهان شاهد پیشرفت سریع علم کامپیوتر، کشفیات و اختراعات روز افزون است. یکی از تکنولوژی‌های جدید اختراع شده بلاکچین است که اساس آن از سال‌های ۱۹۸۰ میلادی بنیان گذاری شده و انقلابی در صنعت مالی و غیرمالی و هم در حوزه‌های مختلف علم کامپیوتر ایجاد کرده است.

تکنولوژی بلاکچین به سیستم‌های توزیع شده و غیرمتمرکز گفته می‌شود. این تکنولوژی کاربردهای بسیاری در زمینه‌های مالی، بهداشت و سلامت، اثبات مالکیت، اینترنت اشیاء، نگهداری و ذخیره‌سازی اطلاعات است و دارای ویژگی‌هایی از جمله غیرمتمرکز بودن، تغییر ناپذیری، شفافیت و امنیت دارد. از آنجایی که بلاکچین یک سیستم توزیع شده است، لازم است شرکت کنندگان در این سیستم بر سر موقعیت سیستم به یک توافق یکسان برسند؛ چرا که هیچ عضو ثالثی جهت تعیین موقعیت سیستم وجود ندارد. از این رو در این تکنولوژی از الگوریتم‌های اجماع جهت به توافق رسیدن اعضا، استفاده شده و کارایی این سیستم‌ها، با توجه به الگوریتم انتخاب شده مشخص می‌شود.

از شروع اختراع بلاکچین، این تکنولوژی پیشرفت‌هایی داشته است. از این زیرساخت در موضوعات مختلف از جمله قراردادهای هوشمند استفاده شده است. در راستای این کاربرد، مفاهیم جدیدی از جمله رمزهای غیرقابل معاوضه^۱ و برنامه‌های غیرمتمرکز^۲ که هر یک در صنایع مختلفی گسترش پیدا کرده‌اند، به وجود آمده است.

در این زمینه‌ها، زبان‌های برنامه نویسی و ابزارهای متعددی به وجود آمده است که توسعه‌دهندگان جهت تسهیل کار خود از آن استفاده می‌کنند. همچنین در حال حاضر پروژه‌ها و پلتفرم‌های مختلفی در زمینه بلاکچین در حال فعالیت و توسعه هستند که بسیاری از آنها در زمینه‌های مالی فعالیت دارند

^۱ Non-Fungible Token (NFT)

^۲ Decentralize Applicatoion (DApp)

چارچوب Hyperledger Fabric:

در تعریف کلی، بلاکچین یک دفترکل از تراکنش‌های غیرقابل تغییر است که در یک شبکه‌ی توزیع شده از گره‌ها نگهداری می‌شود. هریک از این گره‌های یک نسخه از این دفترکل را نگهداری می‌کنند و تراکنش‌ها به کمک پروتکل اجماع تایید می‌شوند و مجموعه‌ای از آنها در یک بلاک قرار می‌گیرند. این بلاک شامل یک هش می‌باشد که به بلاک قبلی خود اشاره می‌کند.

اولین و شناخته شده‌ترین کاربرد بلاکچین، ارز دیجیتال بیتکوین است که می‌توان گفت سایر ارزها راه آن را پیش گرفته‌اند. از دیگر ارزهای دیجیتال می‌توان به اتریوم اشاره کرد که با اضافه کردن قابلیت قرارداد هوشمند، جهت ایجاد شبکه‌ای برای اجرای برنامه‌ها بصورت توزیع شده، راه متفاوتی را نسبت به بیتکوین پیش گرفت. بیتکوین و اتریوم هر دو در دسته **بلاکچین‌های عمومی بدون مجوز** قرار دارند. اساساً در این بلاکچین‌ها همه می‌توانند بصورت ناشناس از شبکه استفاده و در آن تعامل داشته باشند.

با شناخته تر شدن بیتکوین و اتریوم و سایر مشتقات آنها، استفاده از زیرساخت بلاکچین، دفترکل توزیع شده و نرم افزارهای توزیع شده در سازمان‌های مختلف مورد توجه قرار گرفت. اما بعضی از این شرکت‌ها برای برآورده کردن نیازهای خود نیاز به ویژگی‌های خاصی داشتند که بلاکچین‌های عمومی بدون مجوز تامین نمی‌کرد. علاوه بر این، در بعضی از شرکت‌ها شناسایی کاربران بشدت مورد نیاز است که به کمک این نوع بلاکچی‌ها قابل اجرا نیست.

شرکت‌ها برای استفاده نیازمندی‌هایی دارند که شامل:

- قابل شناسایی بودن شرکت کنندگان
- نیاز به شبکه مجوزدار
- عملکرد بالا تراکنش
- تاخیر کم در تایید تراکنش

- حریم خصوصی و محرمانه بودن معاملات و داده‌های مربوط به شرکت

از آنجایی که بسیاری از پلتفرم‌های بلاکچین سعی در سازگار کردن خود برای کاربردهای شرکت‌ها دارند، چارچوب Hyperledger Fabric از ابتدا در این راستا قدم برداشته است؛ که در ادامه به ویژگی‌های آن و تفاوت آن با سایر پلتفرم‌ها پرداخته شده است.

چارچوب Hyperledger Fabric یک پلتفرم برای تکنولوژی دفترکل توزیع شده مجوز دار بصورت متن باز است که برای استفاده در زمینه‌های سازمانی طراحی شده است؛ که قابلیت‌های کلیدی متفاوتی را نسبت به سایر پلتفرم‌های بلاکچین ارائه می‌کند

یکی از تفاوت‌های کلیدی Hyperledger Fabric این است که توسط بنیاد لینوکس مقرر شده است که خود تاریخچه بسیار موفقی در بین پروژه‌های متن باز دارد. این چارچوب توسط کمیته راهبری فنی متنوع و چندین سازمان مدیریت می‌شود. گروه برنامه نویس آن از بیش از ۲۰۰ نفر از بیش از ۳۵ سازمان هستند.

Fabric دارای معماری قابل پیکربندی و ماژولار است که قابلیت نوآوری، تطبیق پذیری و بهینه سازی برای طیف گسترده‌ای از کاربردهای موردنیاز سازمان‌ها مانند بانکداری، مالی، بیمه، منابع انسانی، زنجیره تامین، مراقبت‌های بهداشتی و آهنگ‌های دیجیتال ارائه می‌کند.

Fabric اولین پلتفرم دفترکل توزیع شده است که قرارداد هوشمند را با زبان‌های برنامه نویسی عمومی مانند جاوا، Go و NodeJS پشتیبانی می‌کند. در نتیجه اکثر سازمان‌ها برای توسعه قرارداد هوشمند دانش لازم را دارند و نیاز به یادگیری زبان جدید ندارند.

Fabric یک پلتفرم مجوز دار است. این بدین معناست که برخلاف شبکه‌های عمومی غیرمجوز دار که شرکت‌کنندگان ناشناس و غیرقابل اعتماد، شرکت‌کنندگان در این پلتفرم شناخته و قابل تشخیص برای سایرین می‌باشند.

یکی دیگر از ویژگی‌های این پلتفرم، قابلیت اتصال پروتکل‌های اجماع می‌باشد. به کمک این قابلیت، می‌توان با انتخاب مناسب پروتکل اجماع برای کاربرد موردنظر، مدل قابل اعتمادتر و کارا تر ارائه کرد.

Fabric هیچ ارز دیجیتال بومی ندارد. در نتیجه لازم نیست جهت اجرا قرارداد هوشمند هزینه برای استخراج ارز داشته باشیم.

با توجه به ویژگی‌های ذکر شده، می‌توان گفت این چارچوب نسبت به سایر چارچوب‌ها از عملکرد بهتری در پردازش تراکنش‌ها و میزان تاخیر تایید تراکنش‌ها دارد. همچنین در این پلتفرم حریم خصوصی و محرمانگی تراکنش‌ها و قرارداد هوشمند را مهیا می‌کند.

عملکرد این پلتفرم به متغیرهای مختلفی از جمله سایز تراکنش، سایز بلاک، سایز شبکه و محدودیت‌های سخت افزاری بستگی دارد. تحقیقات بسیاری بر روی قابلیت عملکردی این پلتفرم انجام شده است. آخرین مقیاس‌پذیری ذکر شده معادل پردازش ۲۰ هزار تراکنش در ثانیه می‌باشد.

انواع بلاکچین:

بلاکچین عمومی: بلاکچین‌هایی که شبکه‌ی آن باز است و همه می‌توانند از آن استفاده کنند.

بلاکچین خصوصی: بلاکچین‌هایی که شبکه آن بسته است و همه نمی‌توانند از آن استفاده کنند.

بلاکچین ترکیبی

Permissioned: بلاکچین‌هایی که برای استفاده از شبکه و شرکت در فرایند اجماع نیاز به اجازه دارد. در این شبکه، بلاکچین در بین مجموعه‌ای شناخته شده یا بررسی شده بر اساس مدل قانونی که درجه‌ای از اطمینان را ایجاد میکند، اجرا می‌شود. بلاکچین مجوزدار تعاملات ایمن بین گروهی از موجودیت‌ها که هدف مشترکی دارند ایجاد میکند. به همین دلیل در این نوع بلاکچین از دو نوع پروتکل اجماع CFT و BFT استفاده می‌شود که هزینه استخراج ندارد.

علاوه بر این، احتمال تولید کد مخرب در قرارداد هوشمند کاهش پیدا میکند؛ زیرا تمام افراد در این شبکه یکدیگر را می‌شناسند و هرکسی خطایی در این شبکه مرتکب شود قابل شناسایی خواهد بود. بنابراین فرد مخرب را می‌توان طبق قوانین قرار داده شده تنبیه کرد.

Permissionless: بلاکچین‌هایی که برای استفاده از شبکه و شرکت در فرایند اجماع نیاز به اجازه ندارد. در این شبکه‌ها، هرکسی میتواند در شبکه شرکت کند و بصورت ناشناس باشند. در این حالت هیچ اعتمادی بین شرکت کنندگان وجود ندارد و تنها حالت بلاکچین تغییر ناپذیر است. به همین دلیل در این نوع بلاکچین

مفهوم استخراج کردن ارز بومی یا کارمزد تراکنش وجود دارد تا به کمک آن انگیزه مالی در نوعی اجماع
متحمل خطای بیزانسی بر پایه اثبات کار فراهم کند.

۳- راهکار پیشنهادی

هدف روش پیشنهادی، ذخیره‌سازی امن داده‌ها در شبکه بلاکچین است. بدین منظور سعی شده است برای به دست آوردن روش مناسب جهت ذخیره‌سازی، دو روش مختلف پیاده‌سازی شود. در ابتدا، ذخیره‌سازی داده‌های یک سیستم حمل و نقل جهانی تحت وب در شبکه بلاکچین صورت گرفته است. در انتها، کار انجام شده از حالت خاص منظوره به حالت عام منظوره تبدیل شده است. پیاده‌سازی روش‌های پیشنهادی از دو بخش کاربر و سرور تشکیل شده که در ادامه، به جزئیات پیاده‌سازی هر یک از روش‌ها پرداخته شده است.

۳-۱- پیاده‌سازی پروژه - سمت کاربر

در پروژه حمل و نقل، امکان عقد قرارداد در خدمات مختلف برای کاربران مهیا شده است. این خدمات شامل موارد حمل و نقل دریایی، ریلی، زمینی، هوایی و چند وجهی است که هر یک شامل زیرخدمات مختلفی می‌شوند که در جدول ۳-۱: خدمات و زیرخدمات در پروژه حمل و نقل زیر آورده شده است:

زیرخدماتها									خدماتها
Ship Chandler	Shipyard	Sale Container	Broker	LCL	FCL	Line	Bulk	Chartering	دریایی
Rail Industry			Owner Wagons		Expeditor				ریلی

Drivers	Less Truck Loading	Full Truck Loading	زمینی
Courier	Air Cargo		هوایی
Logistic	Freight Forwarding		چند وجهی

جدول ۱-۳: خدمات و زیر خدمات در پروژه حمل و نقل

در زیر خدمات FCL و Chartering مربوط به خدمت دریایی و Air مربوط به خدمت هوایی، تکنولوژی بلاکچین جهت ذخیره سازی قراردادهای پیاده سازی شده است. در این زیر خدماتها، کاربران بعد از نهایی کردن قراردادهای خود، می توانند قراردادهای مذکور را در نرم افزار تحت بلاکچین مشاهده کنند. سپس در صورت تمایل، هر کاربر می تواند قرارداد خود را با کمک کلید خصوصی خود امضا کند و آن را وارد بلاکچین کند. با اضافه شده قرارداد به بلاکچین، قرارداد ذخیره و غیر قابل ویرایش خواهد شد. نهایی شدن قرارداد بدین معناست که طرفین قرارداد بر سر تعدادی از مفاد خاص قرارداد به توافق برسند و در سایت، قرارداد اولیه خود را نهایی کنند.

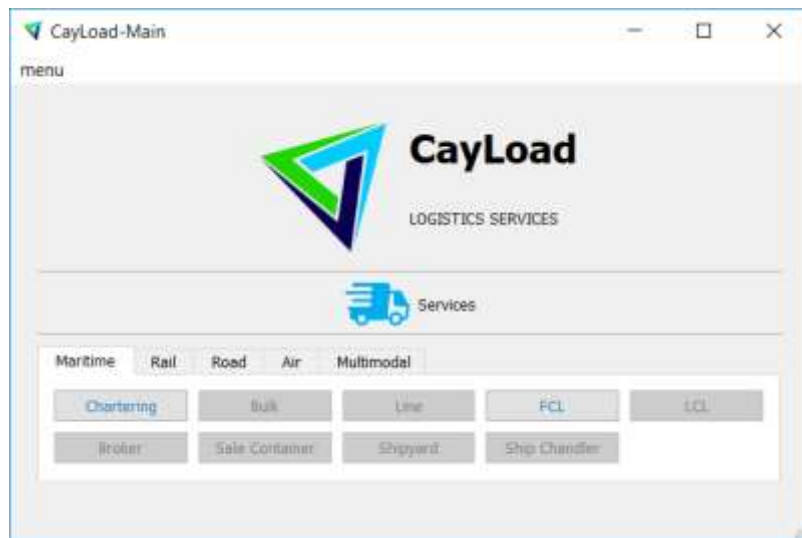
در امضا کردن قراردادهای لازم است که برای طرفین قرارداد کلید عمومی تولید شده باشد (به عبارت دیگر حداقل یکبار در نرم افزار بلاکچین وارد شده باشند) تا اجازه امضا کردن قرارداد به طرفین داده شود؛ در غیر این صورت امکان امضا کردن قرارداد در نرم افزار از طرفین گرفته می شود.

در این طرح، از آنجایی که پروژه یک پروژه خصوصی است، بلاکچین آن از نوع خصوصی می باشد و اعضای موجود در بلاکچین توسط سرور مرکزی سایت مورد احراز هویت قرار می گیرند. به عبارت دیگر، تنها کاربرانی که در وبسایت مربوطه ثبت نام کرده باشند امکان استفاده از نرم افزار بلاکچین را دارد؛ چرا که در ابتدای استفاده از نرم افزار لازم است کاربر احراز هویت کند.

از آنجایی که پیاده سازی زیر خدماتهای FCL و Chartering با Air متفاوت است، جزئیات پیاده سازی هر یک به صورت جداگانه شرح داده شده است.

۳-۱-۱- پیاده‌سازی زیرخدمات‌های FCL و Chartering

از آنجایی که پیاده‌سازی این دو زیرخدمت مشابه یکدیگر است، تنها به جزئیات نحوه‌ی پیاده‌سازی زیرخدمت FCL پرداخته خواهد شد. در این برنامه، پیاده‌سازی بلاکچین با زبان پایتون زده شده است. همچنین برای بخش گرافیک نرم‌افزار از کتابخانه QTPython استفاده شده است. جهت پیاده‌سازی بلاکچین از کتابخانه‌های مختلفی استفاده شده که در ادامه به آن پرداخته می‌شود.



شکل ۳-۱: نمایی از نرم‌افزار بلاکچین

همانطور که گفته شد، ابتدا باید طرفین قرارداد، قرارداد مورد نظر خود را تایید نهایی کنند. برای انجام این کار، هر کاربر می‌تواند قراردادهای خود را در وب سایت مربوط به پروژه‌ی حمل و نقل مشاهده کند و در صورت تمایل آن را تایید نهایی کند. به عنوان مثال، در تصویر زیر نمونه قراردادی از زیرخدمت FCL آورده شده است که در انتهای آن کاربر می‌تواند با زدن دکمه sign the contract قرارداد مربوطه را تایید نهایی کند. در نتیجه زمانی که تمامی افراد داخل قرارداد، مفاد قرارداد را تایید نهایی کنند، قرارداد مربوطه در برنامه تحت **دسکتاپ** نمایش داده می‌شود.

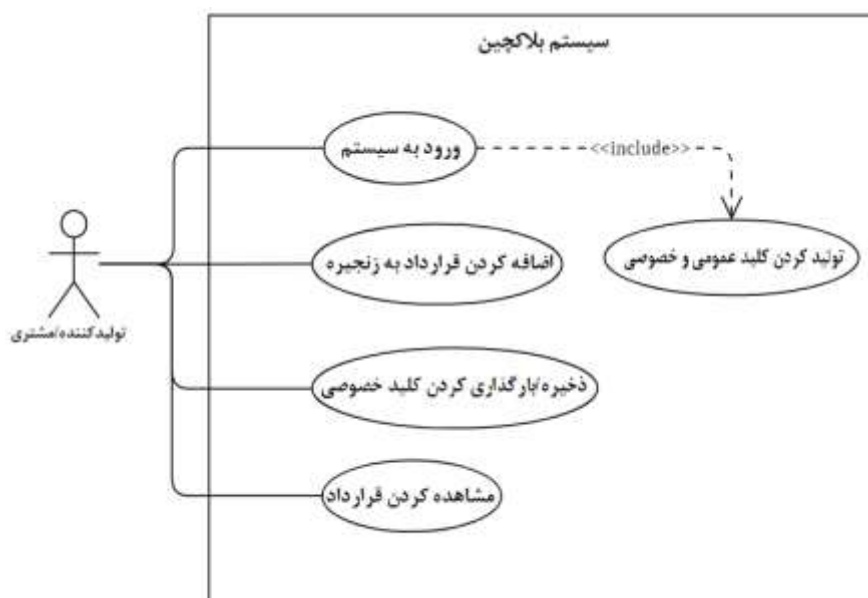
شکل ۳-۲: نمونه‌ای از مفاد قرارداد FCL در وبسایت

با توجه به وجود احتمال افزایش کاربران، مشکلاتی در ذخیره‌سازی و انتقال بلاکچین بین کاربران و سرور به وجود می‌آید. دلیل بروز این مشکل این است که افزایش کاربران باعث افزایش حجم قراردادهای عقد شده بین کاربران می‌شود؛ در نتیجه احتمال افزایش حجم زنجیره بلاکچین وجود دارد. در جهت حل این مسئله تصمیم بر این گرفته شد که برای هر قرارداد یک شبکه بلاکچین جداگانه تشکیل شود و در نهایت شبکه مربوطه برای هر قرارداد در سرور ذخیره می‌شود. در نتیجه هر شبکه بلاکچین به تعداد طرفین قرارداد بعلاوه یک بلوک جنسیس، بلوک خواهد داشت. به عنوان مثال، برای قراردادی که طرفین قرارداد آن دو کاربر هستند، تعداد بلوک‌های شبکه بلاکچین مربوط به آن قرارداد سه بلوک خواهد بود. در نتیجه حجم کمی نیاز خواهد بود تا در سرور ذخیره شود. این مسئله می‌تواند امنیت شبکه بلاکچین را کاهش دهد؛ زیرا در این صورت یک کاربر مخرب می‌تواند با تغییر دادن کل بلوک‌ها (که تعداد آن بسیار کم است) متناسب با خواسته خود، داده‌های ذخیره شده را تغییر دهد.

در ادامه به کمک نمودار مورد کاربرد و نمودار کلاس به جزئیات هر بخش پرداخته شده است. همانطور که گفته شده، از آنجایی که پیاده سازی دو زیرخدمت ذکر شده مشابه یکدیگر است و تنها در API ها با هم متفاوت هستند، به توضیحات تنها یکی از این زیرخدمت ها، FCL، پرداخته شده است.

• نمودار مورد کاربرد زیرخدمت FCL

در پروژه حمل و نقل، بخش بلاکچین یک مورد کاربرد به حساب می آید؛ اما برای فهم بهتر جزئیات این بخش، نمودار مورد کاربرد آن بصورت جزئی تر رسم شده که به شکل زیر می باشد. در این نمودار بخش های مهم سیستم در قالب نمودار مورد کاربرد بیان شده است.



شکل ۳-۳: نمودار مورد کاربرد ریزخدمت FCL

۱. مورد کاربرد ورود به سیستم

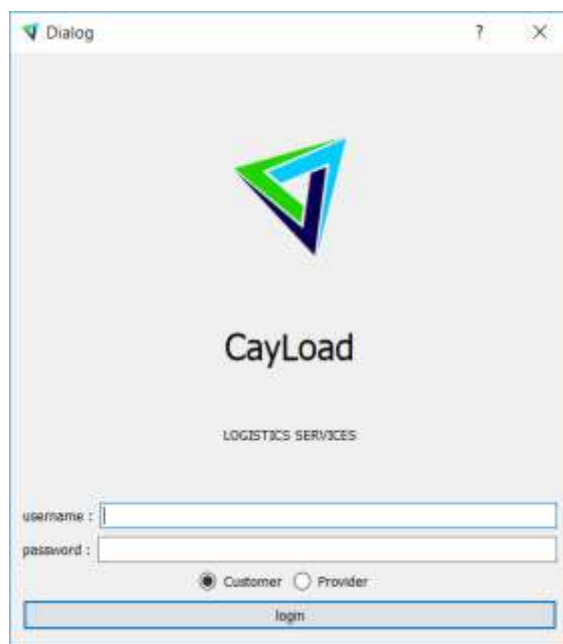
در این مورد کاربرد، برای ورود کاربر به نرم افزار از همان مکانیزم و API های پیاده سازی شده مربوط به سایت استفاده شده است. در فرایند ورود به سیستم، عملیات تولید کلید نیز صورت می گیرد. در طی این فرایند سه حالت برای تولید کلیدها به وجود می آید که به شرح زیر می باشد:

حالت اول: کاربر برای اولین بار نرم افزار را نصب کرده باشد. در اینصورت فیلد کلید عمومی کاربر در سمت سرور خالی است و تنها نام کاربری و رمز عبور فرد وجود دارد. در اینصورت بعد از ورود موفقیت آمیز کاربر،

سیستم کلید عمومی و کلید خصوصی تولید می‌کند و آنها را در سیستم کاربر ذخیره می‌کند؛ همچنین کلید عمومی او به سرور ارسال می‌شود و در آنجا نیز ذخیره می‌شود.

حالت دوم: کاربر نرم افزار را از قبل نصب کرده و مجدد از نرم افزار استفاده می‌کند. در اینصورت کاربر بعد از ورود موفقیت آمیز، از آنجایی که کلید عمومی و خصوصی از قبل تولید شده و در سیستم ذخیره شده است، کلید عمومی و خصوصی جدیدی تولید نمی‌شود و از کلیدهای قبلی استفاده می‌شود.

حالت سوم: کاربر مجبور به نصب مجدد نرم افزار شده است. در اینصورت کاربر بعد از ورود موفقیت آمیز، از آنجایی که کلید عمومی تولید شده او در سرور وجود دارد ولی در سیستم وجود ندارد، مشخص می‌شود کلیدهای عمومی و خصوصی برای او تولید شده است. در نتیجه لازم است کاربر کلید خصوصی خود را بازگذاری کند تا به کمک آن کلید عمومی بازیابی شود و آن را با کلید عمومی موجود در سرور مقایسه شود تا صحت آن تایید شود.



شکل ۳-۴: صفحه ورود به نرم افزار

۲. مورد کاربرد تولید کلید عمومی و خصوصی

در این مورد کاربرد به تولید کلید جهت رمزنگاری نامتقارن پرداخته می‌شود. رمزنگاری نامتقارن یک سیستم رمزنگاری است که از دو کلید جهت رمزگذاری و رمزگشایی استفاده می‌کند. جهت تولید کلید عمومی و خصوصی و قابلیت امضا کردن از الگوریتم رمزنگاری RSA موجود در کتابخانه‌ی آماده‌ی cryptography استفاده شده است. برای تولید کلید خصوصی از تابع `rsa.generate_private_key()` استفاده شده است. یکی از مهمترین پارامترهای این تابع سایز کلید است که معادل ۲۰۴۸ بیت قرار داده شده است. سایز کلید میزان امنیت کلید را مشخص می‌کند که هرچه این سایز بیشتر باشد از امنیت بالاتری برخوردار است. در حال حاضر از آنجایی که روز به روز سیستم‌های قوی‌تری به بازار عرضه می‌شوند در نتیجه حداقل سایز مورد نیاز جهت جعل نشدن کلید در الگوریتم RSA معادل ۲۰۴۸ بیت اعلام شده است. با کلید خصوصی تولید شده، به کمک تابع `public_key()` کلید عمومی متناظر تولید می‌شود.

بعد از ساخت کلیدها، کلیدهای تولید شده در سیستم کاربر ذخیره می‌شود و کلید عمومی به سرور ارسال می‌شود و در سرور نیز ذخیره می‌شود. از آنجایی که کلیدهای گفته شده از اهمیت زیادی برخوردار هستند، قبل از ذخیره سازی کلیدها، آنها به کمک کلید متقارن رمزنگاری می‌شوند. برای رمزنگاری متقارن از کتابخانه PBKDF2HMAC و Fernet استفاده شده است.

۳. مورد کاربرد اضافه کردن قرارداد به زنجیره

یکی از بخش‌های مهم بلاکچین در این پروژه، اضافه کردن قرارداد به زنجیره بلوک است. این مورد کاربرد از بخش‌های مختلفی جهت تکمیل فرایند خود استفاده می‌کند که هر یک به شرح زیر است.

• ایجاد کردن بلوک

هر بلوک شامل اطلاعات مختلفی است. این اطلاعات به شرح زیر می‌باشد:

- index: شماره بلوک در زنجیره بلوک
- timestamp: زمان ایجاد بلوک
- data: اطلاعات مربوط به قرارداد

- signature: شامل نام، امضا و کلید عمومی کاربر

- previuse_hash: هش اطلاعات بلوک قبلی

- hash: هش اطلاعات بلوک فعلی

- proof_of_work: عدد نانس

برای ایجاد بلوک جدید ابتدا لازم است تمام اطلاعات مربوط به قرارداد از سرور دریافت و به رشته تبدیل شود؛ همچنین لازم است زمان ایجاد بلوک به این رشته اضافه شود. در نتیجه زمان ایجاد بلوک در قالب رشته به اطلاعات قرارداد اضافه می‌شود.

در تکنولوژی بلاکچین، الگوریتم‌های مختلفی برای رسیدن به اجماع استفاده می‌شود. الگوریتم مورد استفاده در این پروژه، الگوریتم اثبات کار^۳ است. این الگوریتم برای رسیدن به اجماع جهت جلوگیری از حملات مربوط به شبکه رایانه‌ای استفاده می‌شود. در این مکانیزم، فرستنده با صرف هزینه پردازش محاسبات ریاضیاتی به یک مقدار عددی می‌رسد و گیرنده تنها با کمک آن عدد صحت کار را اثبات می‌کند.

مکانیزم مربوطه با صرف پردازش ریاضیاتی مسئله‌ای را حل می‌کند. این مسئله با کمک تابع هش SHA256 الگوی مشخص شده را تولید می‌کند. در پروژه الگو داده شده 00 در ابتدا مقدار هش تولید شده است. تا زمانی که هش مورد نظر با این الگو تولید نشده باشد، عملیات تولید هش ادامه خواهد داشت. مقدار ورودی تابع SHA256 رشته داده‌ی قرارداد به همراه زمان و عدد نانس^۴ می‌باشد. در هر بار تولید مجدد هش، مقدار عددی نانس اضافه می‌شود. زمانی که هش با الگوی مورد نظر تولید شد آنگاه مقدار عددی نانس در اطلاعات بلوک با عنوان proof_of_work ذخیره می‌شود. در ابتدا مقدار عددی نانس برابر صفر است.

- امضا کردن قرارداد

^۳ Proof of work

^۴ Nonce

در ادامه بعد از کامل شدن اطلاعات بلوک و بدست آمدن هش مورد نظر، عملیات امضای کاربر صورت می‌گیرد. کاربر با کلید خصوصی خود هش بدست آمده را امضا می‌کند تا عملیات انجام شده به نام او ثبت شود. با انجام عملیات امضا، امکان انکار کردن از کاربر گرفته می‌شود؛ چرا که تنها کسی که کلید خصوصی او را دارد خودش است؛ بنابراین امضا شدن با کلید خصوصی کاربر به منزله‌ی امضا کردن توسط خود کاربر می‌باشد.

- **جایگزین کردن زنجیره بلوک جدید در سرور**

در ابتدای ایجاد بلوک جدید، آخرین نسخه زنجیره بلوک از سرور دریافت می‌شود. بعد از ایجاد بلوک جدید و امضا شدن آن با کلید خصوصی کاربر، بلوک جدید به زنجیره بلوک دریافت شده اضافه می‌شود. در ادامه لازم است زنجیره بلوک ایجاد شده جایگزین زنجیره بلوک موجود در سرور شود تا زنجیره بروز شده برای دیگر طرفین قرارداد قابل دسترس باشد. به همین دلیل لازم است قبل ارسال زنجیره بلوک جدید به سرور، شروطی مورد بررسی قرار گیرد؛ چرا که ممکن است در حین ایجاد بلوک جدید، کاربر(ان) دیگر زنجیره بلوک را بروزرسانی کرده باشد و کاربر مذکور از آن مطلع نباشد.

به این منظور در دو مرحله زنجیره بلوک از سرور دریافت می‌شود:

مرحله اول: دریافت آخرین نسخه زنجیره بلوک قبل از ایجاد بلوک جدید (زنجیره بلوک قدیمی)

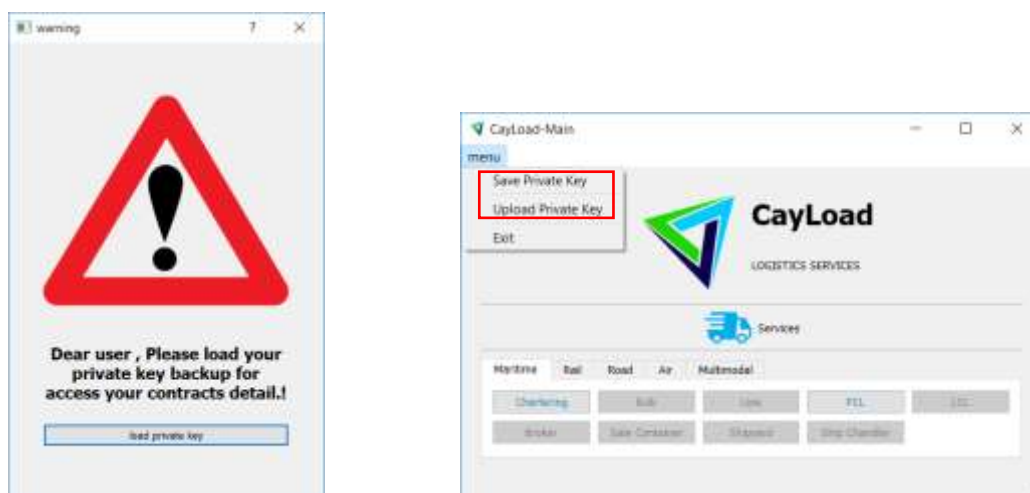
مرحله دوم: دریافت آخرین نسخه زنجیره بلوک بعد از ایجاد بلوک جدید (زنجیره بلوک جدید)

در این مرحله دو شرط مورد بررسی قرار می‌گیرد. شرط اول بررسی طول زنجیره‌های قدیمی و جدید و شرط دوم بررسی زمان ایجاد آخرین بلوک در زنجیره‌ی قدیمی و جدید است. اگر طول زنجیره قدیمی از طول زنجیره جدید کمتر باشد یا زمان ایجاد آخرین بلوک ایجاد شده در زنجیره بلوک قدیمی جلوتر از زمان ایجاد آخرین بلوک ایجاد شده در زنجیره بلوک جدید باشد، در این صورت زنجیره بلوک طی ایجاد بلوک جدید بروزرسانی شده است. در نتیجه لازم است مجدد بلوک جدید ایجاد شود؛ چرا که بلوک جدید از نظر زمانی باید از آخرین بلوک موجود در زنجیره جلوتر باشد. اگر شرایط فوق برقرار نباشد بدین

معناست که زنجیره‌ی ایجاد شده توسط کاربر جدیدترین زنجیره بلوک است و باید جایگزین زنجیره موجود در سرور شود.

۴. مورد کاربرد ذخیره/بارگذاری کلید خصوصی:

از آنجایی که کلید خصوصی از اهمیت بالایی برخوردار است، امکان ذخیره سازی کلید خصوصی در مسیر دلخواه کاربر پیاده‌سازی شده است. باید در نظر داشته باشیم که بازیابی کلید خصوصی به هیچ روشی امکان پذیر نیست و در صورت پاک شدن، امکان دسترسی به قراردادها از کاربر گرفته می‌شود. در نتیجه در این مورد کاربرد، کاربر می‌تواند کلید خود را ذخیره کند. همچنین همانطور که در بخش ورود به سیستم گفته شد، لازم است در مواقعی کاربر کلید خصوصی خود را بارگذاری کند تا کلید عمومی آن تولید شود. در این مورد کاربرد این امکان به کاربر داده می‌شود.




شکل ۳-۵: بارگذاری/ذخیره‌سازی کلید خصوصی


۵. مورد کاربرد مشاهده قراردادها:

مشاهده قراردادهای نهایی شده توسط طرفین قرارداد برای هر یک از کاربران در نرم افزار مهیا شده است. در صفحه اصلی این نرم افزار تمام خدمات و زیرخدمات قابل مشاهده هستند. همانطور که در شکل ۳-۱ مشاهده کردید، در حال حاضر تنها بخش‌هایی که دارای بلاکچین هستند فعال شده است. بنابراین لیست قراردادهای نهایی شده در هر خدمت و زیرخدمت در بخش خود قرار گرفته است. به عنوان مثال، شکل ۳-۶ قرارداد نهایی

شده در بخش FCL برای کاربر قابل مشاهده است. همچنین کاربران می‌توانند قراردادهای خود را در هر زیرخدمت مشاهده کنند. همچنین این قابلیت به کاربر داده شده تا جزئیات قرارداد خود را در قالب یک فایل PDF مشاهده کند.

My Contracts				
No.	Contract ID	Date	Operates	Status
1	36	2021 / 4 / 6	Contract Detail	

شکل ۳-۶: صفحه نمایش لیست قراردادها



A. Client

Trans Asia

B. Supplier

Cayload

Subject of Contract for Making :

1x40 HC CDC

1x20 GP CDC

Carrier Line :

Contract Conditions :

Cargo Category

Commodity

HS Code

Cross Weight

Net Weight

Port of Landing

Port of Discharge

POL Term

POD Term

Loading Date

Transit Time

Liner

Include

Exclude

CLS

ETD

Free Time POL (Days)

Free Time POD (Days)

Shipper :

Cnee :

Third Party :

Contract Amount

Total Amount

Third Party :

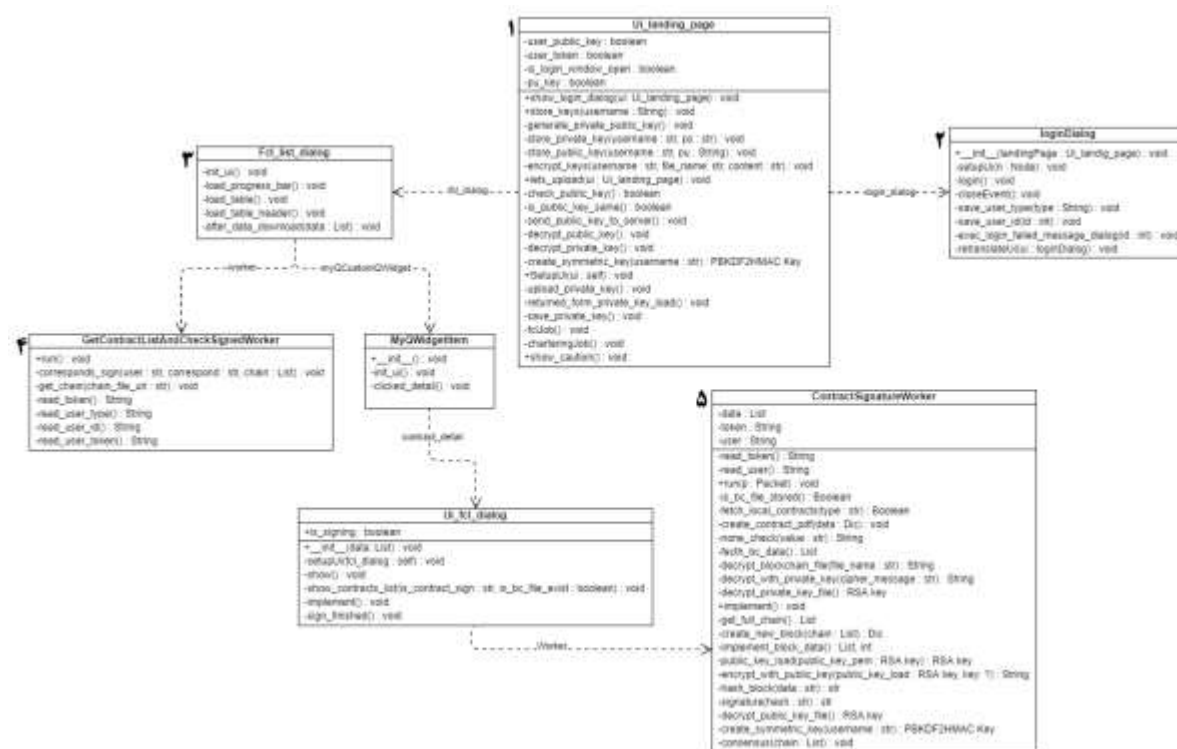
Collect / Prepaid

شکل ۳-۷: نمایشی از اطلاعات قرارداد در قالب PDF

• نمودار کلاس زیر خدمت FCL

جهت پیاده سازی این نرم افزار، نمودار کلاس آن رسم شده است که در تصویر زیر قابل مشاهده می باشد.

کلاس شروع کننده کلاس `Ui_landing_page` می باشد که در فایل `main.py` پیاده سازی شده است. در نمودار زیر، تنها کلاس های مربوط به خدمت `FCL` رسم شده است. سایر خدمات کلاس های مشابه و مشترکی دارند و تنها باید `API` هر یک مطابق خدمت خود جایگزین شود.



شکل ۳-۸: نمودار کلاس زیر خدمت FCL

جدول ۳-۲: نداشت توابع به سند

نام مورد کاربرد	قابلیت	نام کلاس	نام متد
ورود به سیستم	نمایش اولیه صفحه ورود	Ui_landing_page	show_login_dialog()
	فرایند احراز هویت	Ui_logging_dialog	login()
	ذخیره اطلاعات در سیستم میزبان	Ui_logging_dialog	save_user_type() save_user_id()

lets_upload() check_public_key() is_public_key_same()	۱	Ui_landing_page	بررسی وضعیت کلیدها	
generate_private_public_key() store_private_key() store_public_key()	۱	Ui_landing_page	تولید کلیدها	تولید کردن کلید عمومی/خصوصی
encrypt_keys() create_symmetric_key()	۱	Ui_landing_page	رمزنگاری کلیدها	
create_new_block()	۵	contractSignatureWorker	ایجاد کردن بلوک	اضافه کردن قرارداد به زنجیره
signature()	۵	contractSignatureWorker	امضا کردن قرارداد	
consensus()	۵	contractSignatureWorker	جایگزین کردن زنجیره بلوک جدید در سرور	
save_private_key()	۱	Ui_landing_page	ذخیره کردن کلید	ذخیره/بارگذاری کلید خصوصی
upload_private_key()	۱	Ui_landing_page	بارگذاری کردن کلید	
init_ui() load_progress_bar() load_table() load_table_header()	۳	Fcl_list_dialog	ایجاد جدول قراردادهای	مشاهده قرارداد
run() corresponds_sign()	۴	GetContractListAndCheckSignedWorker	نمایش لیست قراردادهای	
run() create_contract_pdf()	۵	contractSignatureWorker	نمایش به صورت PDF	

۳-۱-۲- پیاده سازی زیر خدمت Air

در زیر خدمت Air، برای استفاده از بلاکچین از چارچوب آماده‌ی Fabric Hyperledger استفاده شده است که جزئیات عملکرد آن در بخش پیشینه آورده شده است. در این زیر خدمت، چارچوب ذکر شده جهت ذخیره سازی امن داده‌ها استفاده می‌شود. با توجه به اینکه این چارچوب مشکل مقیاس پذیری کمتری نسبت به روش

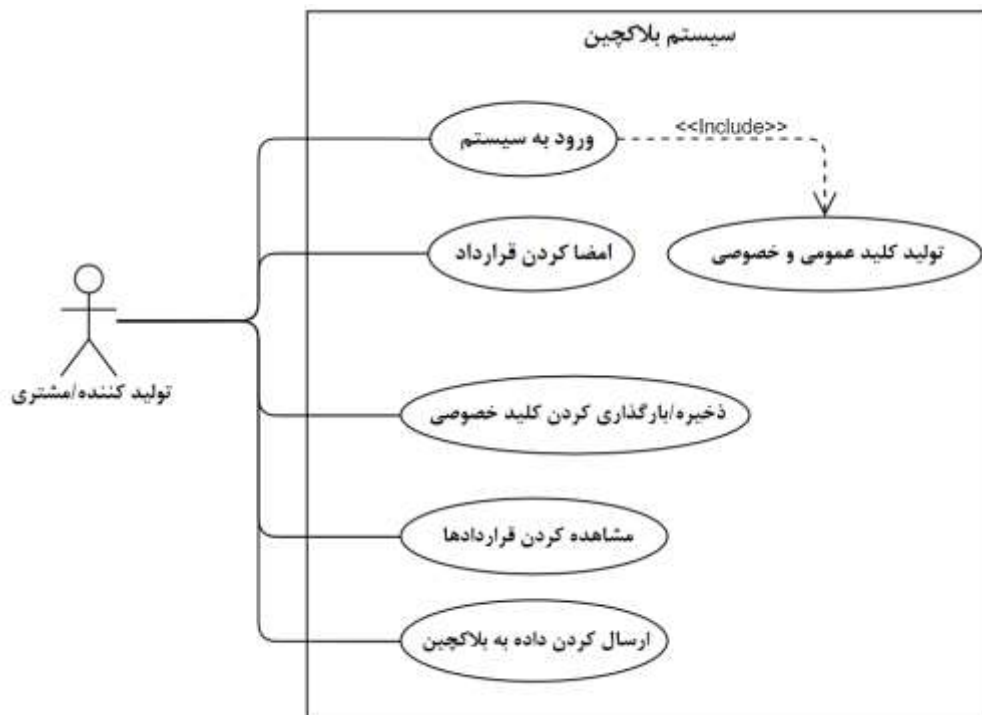
زده شده در دو زیرخدمت دیگر دارد، در نتیجه تمامی قراردادها در یک زنجیره ذخیره می‌شود. در این زیرخدمت، مانند دو زیرخدمت دیگر، از یک نرم افزار یکسان استفاده شده است با این تفاوت که در پیاده‌سازی سرور با یکدیگر متفاوت هستند.



شکل ۳-۹: نمایی از نرم افزار - زیرخدمت Air

در ادامه به کمک نمودار مورد کاربرد و نمودار کلاس به جزئیات هر بخش پرداخته شده است.

- نمودار مورد کاربرد زیرخدمت Air



شکل ۳-۱۰: نمودار مورد کاربر زیر خدمت Air

در نمودار مورد کاربر نشان داده شده، تعدادی از مورد کاربردها با ماژول FCL و Chartering یکسان است که شامل موارد کاربرد ورود به سیستم، امضا کردن قرارداد، ذخیره/بارگذاری کردن کلید خصوصی، مشاهده قراردادها و تولید کلید عمومی و خصوصی است. بنابراین از توضیح مجدد آن در این سند خودداری شده است. بنابراین تنها در مورد کاربرد "ارسال کردن داده به بلاکچین" با یکدیگر متفاوت هستند که در ادامه با جزئیات بیشتری به آن پرداخته شده است.

از آنجایی که تنها تعدادی از داده‌ها مهم ضرورت ذخیره سازی در بلاکچین را دارند، در نتیجه باید داده‌های مورد نظر را از بین داده‌های دریافتی از سایت پیش پردازش شود. در نتیجه داده‌ها قبل از ارسال در داخل یک دیکشنری و در فرمت JSON نگهداری می‌شود.

داده‌هایی که در قالب دیکشنری نگهداری می‌شود مطابق زیر است:

- key: آیدی هر قرارداد در بلاکچین (یکتا)
- data: داده قرارداد بدون امضا کاربر
- data_signed: داده‌س امضا شده قرارداد توسط کاربر

- `username`: نام کاربری

- `public_key`: کلید عمومی کاربر (کلید متناظر با کلید خصوصی که با آن قرارداد امضا شده است)

داده‌ها بعد از پردازش و آماده شدن، جهت ذخیره سازی در شبکه بلاکچین، به سمت سرور ارسال می‌شوند تا با بررسی آنها داده‌های مربوطه در شبکه ذخیره شوند. مکانیزم اجرایی چارچوب جهت ذخیره‌سازی داده‌ها در بخش پیاده سازی سرور توضیح داده شده است.

- **نمودار کلاس زیرخدمت Air**

جهت پیاده‌سازی این زیرخدمت در نرم افزار، نمودار کلاس آن رسم شده است که در تصویر زیر قابل مشاهده می‌باشد. کلاس شروع کننده کلاس `Ui_landing_page` می‌باشد که در فایل `main.py` پیاده‌سازی شده است. در فایل `main.py` سه زیرخدمت ذکر شده فعال شده است. همانطور که در نمودار مورد کاربرد مشاهده کردید، زیرخدمت `Air` تنها در یک مورد کاربرد متفاوت است. در نتیجه نمودار کلاس آن با دو زیرخدمت دیگر یکسان است و تنها تعدادی از توابع آن در کلاس `contractSignatureWorker` تغییر کرده است یا تابعی اضافه شده است. به عبارت دیگر در این کلاس، نحوه پردازش داده و ارسال آن تغییر یافته است. سایر خدمات ارائه شده در این نرم افزار از سمت کاربر با سایر ریزخدماتها یکسان می‌باشد.

۳-۲- پیاده‌سازی پروژه - سمت سرور

برای تکمیل فرایند بلاکچین لازم است فرایندهایی سمت سرور اجرا و پردازش شود. در سه زیرخدمت که پیاده سازی بلاکچین انجام شده است، پیاده سازی دو زیرخدمت FCL و Chartering با زیرخدمت Air متفاوت است. در ادامه به جزئیات هر یک پرداخته می‌شود.

۳-۲-۱- پیاده‌سازی زیرخدمت‌های FCL و Chartering

در این دو زیرخدمت پردازش‌های سمت سرور شامل ایجاد فایل اولیه بلاکچین، تایید بلوک جدید و تایید زنجیره بلوک است. همانطور که گفته شد، یکی از قابلیت‌های موجود این نرم افزار مشاهده قراردادها توسط کاربر می‌باشد. شرط نمایش قرارداد در نرم افزار ایجاد فایل اولیه بلاکچین است. به عبارت دیگر، زمانی که برای قرارداد فایل اولیه بلاکچین تولید شد، آنگاه توسط کاربر قابل مشاهده خواهد شد؛ در غیر اینصورت کاربر قادر به مشاهده و امضای قراردادی که در سایت ایجاد کرده است در نرم افزار نخواهد بود. به این منظور در مرحله اول، زمانی که قراردادی توسط طرفین قرارداد به تایید نهایی می‌رسد، در سمت سرور فایل اولیه بلاکچین آن قرارداد که شامل بلوک اولیه که به آن بلوک جنسیس گفته می‌شود را تولید خواهد کرد و در مسیر سرویس جاری ذخیره خواهد شد. تایید نهایی کاربر به منزله‌ی عدم تغییر مفاد قرارداد خواهد بود؛ به عبارت دیگر، زمانی که طرفین قرارداد، قرارداد مذکور را تایید نهایی می‌کنند، قادر به تغییر محتوای آن نخواهد بود. چرا که بعد از این تایید کاربر قادر به مشاهده قرارداد در نرم افزار و امضای آن خواهد بود.

در ادامه بعد از اینکه کاربر قراردادی را امضا کردن، که امضا کردن قرارداد به معنای ایجاد بلوک جدید و اضافه کردن قرارداد جدید به زنجیره است، لازم است زنجیره بلوک جدید ایجاد شده درستی و صحت آن مورد بررسی قرار گیرد. در مرحله اول باید بلوک جدید اضافه شده مورد بررسی قرار گیرد. در بررسی بلوک جدید دو شرط در آن مورد بررسی قرار می‌گیرد:

۱- بررسی صحت هش تولید شده بلوک. سرور مجدد با توجه به اطلاعات بلوک، هش آن را محاسبه

می‌کند و مقدار آن را با هش تولید شده توسط کاربر مقایسه می‌کند.

۲- بررسی امضای کاربر در بلوک. سرور مجدد با توجه به اطلاعات بلوک، امضای کاربر را مورد صحت

سنجی قرار می‌دهد. برای صحت سنجی امضای کاربر از کلید عمومی او استفاده خواهد شد. با

کمک کلید عمومی کاربر، اطلاعات امضا شده صحت سنجی می‌شود.

در صورتی که هر دو شرط بالا برقرار باشد آنگاه درستی بلوک تولید شده تایید می‌شود. در مرحله دوم باید

زنجیره بلوک صحت سنجی شود. به عبارت دیگر؛ باید ارتباط بین بلوک‌های تولید شده در بلاکچین مورد

بررسی قرار گیرد. به این منظور، هش بلوک قبلی که در بلوک فعلی ذخیره شده است با هش خود بلوک قبلی

مورد مقایسه قرار می‌گیرد. در صورت یکسان نبودن این دو مقدار می‌توان گفت بلوک قبلی دستکاری شده و

در نتیجه زنجیره بلوک موجود درست نمی‌باشد. در غیر اینصورت، صحت زنجیره بلوک تایید خواهد شد.

برای اجرای فرایندهای گفته تنها سه متد در سمت سرور در شرایط لازم اجرا خواهند شد. که عبارتند

create_genesis_block(), verify() و verify_sign() می‌باشد.

۳-۲-۲- پیاده‌سازی زیرخدمت Air

در این زیرخدمت برای ذخیره‌سازی داده‌ها از چارچوب Hyperledger Fabric استفاده شده است. این بخش

از سیستم، به دلیل کمبود سرور تهیه شده توسط کارفرما، چارچوب مذکور در یک سرور اجرا شده است. لازم

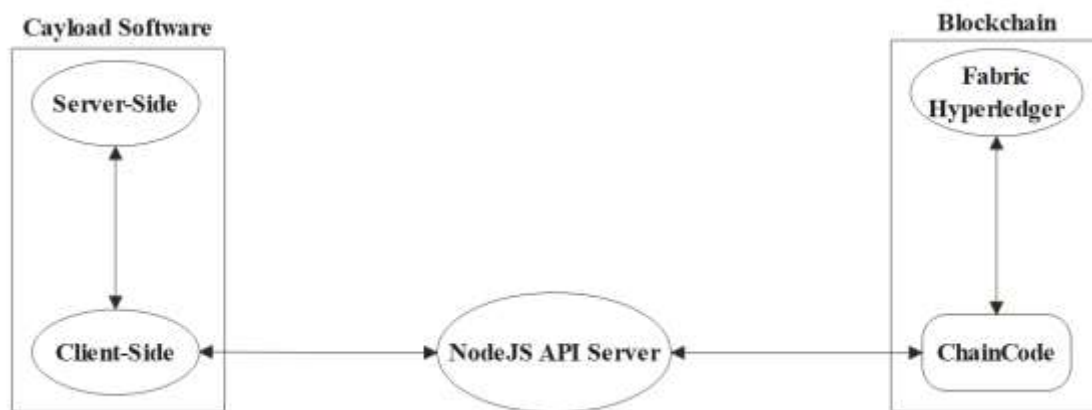
به ذکر است با توجه به کارکرد بلاکچین مربوط به چارچوب ذکر شده، می‌توان برنامه مورد نظر را در چندین

سرور اجرا کرد که این کار باعث افزایش امنیت زنجیره‌ی بلاکچین می‌شود.

پیاده‌سازی سمت سرور از دو بخش تشکیل شده است. بخش اول مربوط به چارچوب است و بخش دوم مربوط

به پیاده‌سازی سرور جهت برقراری ارتباط با فریم. جزئیات هر یک از بخش‌ها را در ادامه خواهیم داشت. بطور

کلی معماری نرم افزار به شرح زیر است:



شکل ۳-۱۲: شماتیک روابط بین اجزای سازنده سمت سرور

با توجه به مکانیزم عملکرد چارچوب Fabric Hyperledger، برای دسترسی به زنجیره‌ی بلاک‌ها لازم است به کمک یک قرارداد هوشمند نحوه برقراری ارتباط با بلاکچین را مشخص کرد. به عبارت دیگر، هرگونه اقدام برای دسترسی به بلاکچین باید در قرارداد هوشمند برنامه نویسی شود. به عنوان مثال، اگر بخواهیم به یکی از بلوک‌های موجود در بلاکچین دسترسی داشته باشیم، لازم است تابعی را جهت دریافت بلوک مورد نظر در قرارداد هوشمند پیاده‌سازی کرده باشیم. در اصطلاح به این قرارداد هوشمند Chaincode گفته می‌شود.

در سیستم مورد نظر، برای جستجو کردن بر روی زنجیره، اضافه کردن داده بر روی آن یا دریافت یک قرارداد خاص، سه تابع جداگانه در قرارداد هوشمند پیاده‌سازی شده است که در Error! Reference source not found آمده است. لازم به ذکر است، با توجه به نحوه عملکرد چارچوب، از آنجایی که این قرارداد هوشمند باید بر روی بلاکچین نصب شود، با هر بار تغییر کد لازم است سیستم بلاکچین مجدد راه‌اندازی شود که این بدین معناست زنجیره جدیدی ایجاد می‌شود. در نتیجه باید در پیاده‌سازی آن دقت لازم را داشت چرا که بعد از نصب، امکان تغییر آن با حفظ اطلاعات موجود در بلاکچین وجود ندارد.

جدول ۳-۴: نداشت توابع قرارداد هوشمند نصب شده بر روی بلاکچین

API	قابلیت	
InitLedger()	مقداردهی اولیه زنجیره	قرارداد هوشمند (Chaincode)
queryContract()	جستجوی داده‌ی خاص	
queryAllContract()	جستجوی تمام داده‌ها	
addContract()	اضافه کردن داده به زنجیره	

از سمت دیگر، با توجه به شکل ۳-۱۲، برای برقراری ارتباط بین بلاکچین و برنامه تحت دسکتاپ لازم است برنامه‌ای به عنوان واسط وجود داشته باشد تا بتواند درخواست‌ها از سمت برنامه تحت دسکتاپ را به قرارداد هوشمند ارسال کند. از آنجایی که قرارداد هوشمند تنها با چند زبان محدود برنامه نویسی امکان برقراری ارتباط را دارد، در نتیجه این برنامه به زبان NodeJS که یکی از زبان‌های قابل قبول قرارداد هوشمند است، پیاده‌سازی شده است. این برنامه در قالب REST API زده شده است. برای برقراری ارتباط، سه API زده شده است که در جدول زیر آورده شده است. این برنامه برای برقراری ارتباط با قرارداد هوشمند بازم است اطلاعات شبکه‌ای را که قرارداد هوشمند در آن قرار دارد را داشته باشد و به شبکه متصل شود. به همین منظور تابعی پیاده سازی شده است که به کمک یک کتابخانه، نوشته شده توسط تیم سازنده‌ی چارچوب Fabric Hyperledger، به نام fabric-network می‌توان یک دروازه اینترنت ایجاد کرد که با شبکه مربوطه متصل شد.

جدول ۳-۵: نگاشت توابع مربوط به برنامه واسط

API/Method		قابلیت	
GET	query()	دریافت تمام قراردادها	NodeJS APIs
GET	queryContract()	دریافت قرارداد خاص	
POST	addContract()	اضافه کردن قرارداد	
Method	configNetwork ()	اتصال به شبکه بلاکچین	

در چارچوب Fabric Hyperledger، برای راه اندازی شبکه بلاکچین از داکر استفاده شده است. در این زیرخدمت، شبکه بلاکچین از سه گره مجازی تشکیل شده که نام آن‌ها Org1، Org2 و Orderer1 است. برای راه اندازی شبکه، دو فایل bash نوشته شده است که به کمک آن فرایند مربوطه، به صورت خودکار و پشت سر هم اجرا می‌شود. در ابتدا فایل startFabric.sh را اجرا می‌کنیم. در این فایل، ابتدا اگر شبکه‌ای موجود باشد آن را غیرفعال می‌کند؛ سپس شبکه جدیدی را ایجاد می‌کند. سپس کانال جدیدی را ایجاد میکند تا به کمک آن گره‌های نام برده شده به یکدیگر متصل شوند و با یکدیگر در تعامل باشند. برای ایجاد کانال جدید

لازم است که فایل اصلی به نام network.sh اجرا شود؛ که همانطور که گفته شد این کارها بصورت خودکار اجرا می‌شود و لازم به وارد کردن دستورات مورد نظر به صورت دستی نیست. در فایل network.sh توابعی نوشته شده است که به شرح زیر است. لازم به ذکر است که بعضی از این توابع به یک فایل bash دیگر ارجاع داده شده است. به عبارتی دیگر، فایل network.sh فایلی اجرایی اصلی به حساب می‌آید.

جدول ۳-۶: نداشت توابع در راه‌اندازی شبکه بلاکچین

فایل Bash	توابع	قابلیت	
-	clearContainers()	پاک کردن ظرف‌های غیرفعال داکر	network.sh
-	clearUnwantedImages()	پاک کردن ایمیج‌های اضافی داکر	
-	checkPrereqs()	بررسی داشتن پیش‌فرض‌ها	
registerEnroll.sh ccp-generate.sh	createOrgs()	ایجاد گره	
-	createConsortium()	ایجاد شبکه جدید	
-	networkUp()	فعال کردن شبکه	
createChannel.sh	createChannel()	ایجاد کانال	
deployCC.sh	deployCC()	نصب قرارداد هوشمند به کانال	
-	networkDown()	غیرفعال کردن شبکه	

لازم به ذکر است که شبکه‌ی گفته شده یک شبکه مجازی می‌باشد که امکان دسترسی به آن تنها به افرادی خاص داده شده است و عموم مردم امکان دسترسی به این شبکه را ندارند. برای برقراری ارتباط با قرارداد هوشمند باید احراز هویت صورت بگیرد. در این برنامه تنها دو کاربر مجازی امکان دسترسی به شبکه بلاکچین را دارند (یکی مدیر شبکه و دیگری کاربر عادی) که به کمک یکی از این دو کاربر امکان دسترسی به بلاکچین داده می‌شود. به عبارت دیگر، برنامه واسط برای هر درخواست باید به واسط یکی از این دو کاربر درخواست خود را به قرارداد هوشمند ارسال کند.

جهت مشاهده سایت و دانلود برنامه بلاکچین می‌توانید به دو لینک زیر مراجعه کنید.

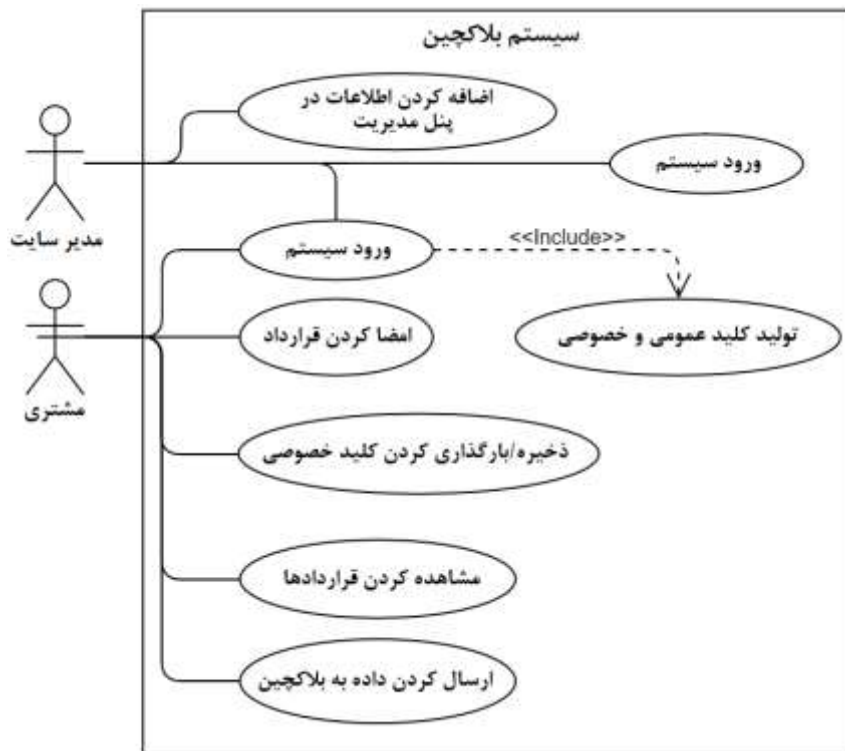
[آدرس سایت](#)

[لینک دانلود برنامه بلاکچین تحت دسکتاپ](#)

۳-۳- پیاده‌سازی نرم‌افزار جامع ذخیره‌سازی داده

باتوجه به روش‌های پیاده‌سازی شده در دو بخش قبل، از آنجایی که پروژه مورد نظر برای یک سایت خاص پیاده‌سازی شده است؛ در نتیجه قالب داده‌ها مشخص بود و بصورت مستقیم در کد برنامه وارد شده بود. از این رو این نرم‌افزار بصورت جامع قابل استفاده نخواهد بود. در جهت استفاده از این راهکار به صورت جامع، نرم‌افزار جامع ذخیره‌سازی داده طراحی شده است که دیگر سایت‌ها نیز قابلیت استفاده از آن را داشته باشند. روش پیاده‌سازی در این نرم‌افزار مطابق روش پیاده شده در زیرخدمت Air است، با این تفاوت که این امکان وجود دارد که اکثر شرکت‌ها یا سایت‌ها در این نرم‌افزار ثبت نام کنند و از قابلیت ذخیره‌سازی داده در محیط امن بلاکچین استفاده کنند. به عبارت دیگر، در این نرم‌افزار جهت ذخیره‌سازی داده‌ها از چارچوب Hyperledger Fabric استفاده شده است. بنابراین تغییراتی در این بخش وجود ندارد.

جهت راه‌اندازی درست این نرم‌افزار برای هر شرکت، لازم بود در پیاده‌سازی سمت مشتری قابلیت ثبت نام شرکت/سایت و اضافه کردن اطلاعات مورد نیاز توسط مدیر شرکت/سایت اضافه شود. نمودار مورد کاربر این نرم‌افزار در شکل ... قابل مشاهده است. مطابق تصویر زیر تنها دو مورد کاربرد به آن اضافه شده است و سایر مورد کاربردها مطابق زیرخدمت Air می‌باشد.



- **مورد کاربر ثبت نام:** در این مورد کاربر، مدیر سایت/شرکت می‌تواند در این نرم افزار ثبت نام کند.
- **مورد کاربرد ورود به سیستم:** در این مورد کاربر، مدیر سایت/شرکت می‌تواند بعد از ثبت نام در این نرم افزار وارد پنل مدیریت شود.
- **مورد کاربرد اضافه کردن اطلاعات در پنل مدیریت:** مدیر شرکت ثبت نام شده در این نرم افزار موظف است اطلاعات لازم را تکمیل کند. به این منظور پروفایل مدیر در این نرم افزار طراحی شده است تا به کمک آن اطلاعات مورد نیاز را در هر بخش تکمیل کند.
- جهت پیاده سازی این نرم افزار، نمودار کلاس آن رسم شده است که در **شکل** قابل مشاهده می‌باشد. کلاس شروع کننده کلاس `Ui_landing_page` می‌باشد که در فایل `main.py` پیاده سازی شده است.

consensus()	۵	contractSignatureWorker	جایگزین کردن زنجیره بلوک جدید در سرور	
save_private_key()	۱	Ui_landing_page	ذخیره کردن کلید	ذخیره/بارگذاری
upload_private_key()	۱	Ui_landing_page	بارگذاری کردن کلید	کلید خصوصی
init_ui() load_progress_bar() load_table() load_table_header()	۳	Ui_client_page	ایجاد جدول قراردادها	مشاهده قرارداد
run() corresponds_sign()	۴	GetContractListAndCheckSignedWorker	نمایش لیست قراردادها	
run() create_contract_pdf()	۵	contractSignatureWorker	نمایش به صورت PDF	

۴- ارزیابی

۵- نتیجه‌گیری و کارهای آتی

۶- راهنمای فنی

در این بخش جهت اجرای برنامه راهنمای فنی آن اضافه شده است که با توجه به دو نوع پیاده‌سازی انجام شده، دو راهنمای فنی خواهیم داشت. در ابتدا به بخش فرانت وبسایت که در هر سه زیرخدمت کارهای یکسانی صورت می‌گیرد، پرداخته شده است. سپس به بخش سرور می‌پردازیم. در انتها بخش کاربر را بررسی خواهیم کرد.

۶-۱- راهنمای فنی – فرانت وبسایت Cayload

در ابتدا لازم است قراردادهای منعقد شده توسط طرفین قرارداد تایید نهایی شود. به این منظور لازم است به کمک متغیری در سمت سرور از تایید نهایی شدن قرارداد مطلع شویم. به همین دلیل در وبسایت مورد نظر،

در انتهای هر قرارداد دکمه‌ای قرار داده شده است. با کلیک کردن دکمه مورد نظر، تابع `handleFinishByRole()` اجرا خواهد شد (تابع ذکر شده و دکمه‌ی مورد نظر در فایل‌های مربوط به بخش فرانت که آدرس آنها در جدول ۱-۶ آورده شده است، تعریف شده‌اند). به کمک این تابع، متغیر `finished_by_customer` یا `finished_by_provider` که در سمت سرور تعریف شده است به حالت `True` تغییر پیدا می‌کند. عملیات تابع ذکر شده در هر سه زیرخدمت یکسان است و تنها در API ارسال داده متفاوت می‌باشند.

جدول ۱-۶: آدرس فایل فرانت زیرخدمت‌ها

نام زیرخدمت	آدرس فایل
FCL	src/scenes/dashboard/shipping/fcl/info.js
Chartering	src/scenes/dashboard/shipping/chartering/quotation.js
Air	src/scenes/dashboard/air/info.js

۲-۶- راهنمای فنی – سمت سرور

۱-۲-۶- زیرخدمت FCL و Chartering – سمت سرور

با توجه به نوع پیاده‌سازی، بخش سرور دو زیرخدمت FCL و Chartering زبان پایتون و چارچوب Django زده شده است. از آنجایی که فرایند این دو زیرخدمت در بخش سرور بصورت یکسان عمل می‌کند، تنها به زیرخدمت FCL پرداخته شده است. فایل‌های سمت سرور هر زیرخدمت در جدول ۲-۶ آورده شده است.

جدول ۲-۶: آدرس فایل سرور زیرخدمت‌ها

نام زیرخدمت	آدرس فایل
FCL	core/models/services/shipping/fcl.py
Chartering	core/models/services/shipping/chartering.py

همانطور که در بخش قبل گفته شد، زمانی که هر دو متغیر `finished_by_provider` و `finished_by_customer` برابر `True` باشند، در این صورت فایل بلاکچین آن که شامل بلوک جنسیس است به کمک تابع `blockchain_on_finished_contract()` تولید خواهد شد.

از طرفی دیگر، زمانی که قراردادی در نرم افزار بلاکچین توسط کاربر امضا می شود، بلوک مربوط به قرارداد مذکور ساخته شده و به سرور ارسال می شود. بلوک ارسال شده از سه جهت مورد بررسی قرار می گیرد که شامل صحت هش تولید شده، صحت زنجیره تولید شده و صحت امضا می شود. در صورتی که هر سه این موارد صحیح باشند، بلوک مورد نظر روی زنجیره قرار می گیرد. در جهت بررسی صحت موارد گفته شده توابع طبق جدول ۳-۶ فراخوانی می شوند.

جدول ۳-۶: نگاشت توابع سمت سرور

تابع	عملیات	توضیح نحوه عملکرد	جزئیات کد
<code>valid_hash()</code>	صحت سنجی هش	در این تابع به کمک هش <code>SHA256</code> و داده های خام، هش تولید شده و با هش ثبت شده در بلوک مقایسه می شود	پیوست ۱
<code>chain_validity()</code>	صحت سنجی زنجیره	در این تابع هش هر بلوک یا مقدار هش ثبت شده در بلوک قبلی مورد مقایسه قرار می گیرد	پیوست ۲
<code>sign_validity()</code>	صحت سنجی امضا	در این تابع به کمک کلید عمومی کاربر و داده های خام، صحت سنجی امضا مورد بررسی قرار می گیرد	پیوست ۳

۶-۲-۲- زیر خدمت Air – سمت سرور

از آنجا که در زیر خدمت `Air` از چارچوب `Hyperledger Fabric` استفاده شده است، پیاده سازی آن با سایر زیر خدمت ها متفاوت شده است. در بخش سرور این زیر خدمت لازم است علاوه بر راه اندازی چارچوب `Hyperledger Fabric`، برنامه ای جهت برقراری ارتباط بین این چارچوب با برنامه تحت دسکتاپ به زبان `NodeJS` نوشته شود.

چارچوب Hyperledger Fabric این قابلیت را دارد تا عملیاتی را که لازم داریم بر روی بلاکچین انجام شود را تعریف کنیم. برای اینکار لازم است قرارداد هوشمندی را تعریف کنیم تا عملیات مورد نیاز ما را اجرا کند. در این پروژه، اضافه کردن داده جدید، جستجوی بلوک خاص و دریافت تمام بلوک‌ها در قرارداد هوشمند تعریف شده است.

با توجه به شکل ۳-۱۲، جهت انجام عملیات مختلف با چارچوب Hyperledger Fabric لازم است که در یک قرارداد هوشمند عملیات مجاز برای برقراری با بلاکچین مورد نظر با زبان NodeJS پیاده‌سازی شود. این قرارداد هوشمند بعد از نصب شدن دیگر قابلیت تغییر ندارد؛ بنابراین در پیاده‌سازی آن باید دقت لازم را داشته باشیم، زیرا در صورتی که لازم باشد کد آن تغییر کند، تمام اطلاعات موجود در بلاکچین از بین خواهد رفت. از طرفی برای برقراری ارتباط با این قرارداد هوشمند، برنامه‌ای نوشته شده تا بتواند به این قرارداد هوشمند متصل شود و عملیات مورد نیاز کاربر را انجام دهد. این برنامه با زبان NodeJS و با REST API زده شده است.

در ابتدا به عملیات‌های تعریف شده در قرارداد هوشمند می‌پردازیم. در زبان برنامه نویسی NodeJS کتابخانه‌ای به نام fabric-contract-list وجود دارد که به کمک آن میتوان به Hyperledger Fabric متصل شد. در این برنامه کلاسی به نام Cayload تعریف شده است. در سازنده‌ی آن اولین بلوک ساخته می‌شود که اطلاعات ذخیره شده در آن بصورت دستی وارد شده است.

```

async initLedger(ctx) {
  console.info('===== START : Initialize Ledger =====');
  const contracts = [
    {
      data: 'This is the first transaction, which must be original',
      signature: {
        data_signed: 'This is the first transaction, which must be signed by user',
        username: 'Server',
        public_key: 'No public key for this user'
      }
    }
  ];

  for (let i = 0; i < contracts.length; i++) {
    contracts[i].docType = 'contract';
    await ctx.stub.putState('CONTRACT' + i, Buffer.from(JSON.stringify(contracts[i])));
    console.info('Added <--> ', contracts[i]);
  }
  console.info('===== END : Initialize Ledger =====');
}

```

شکل ۶-۱: سازنده کلاس Cayload در قرارداد هوشمند

در این کلاس سه تابع تعریف شده است که طبق جدول ۴-۶ می‌باشد. این برنامه بعد از نصب شدن دیگر قابل تغییر نمی‌باشد، به عبارت دیگر اگر بعد از نصب شدن بلاکچین بر روی سرور، تابعی به این کلاس اضافه شود، تابع اضافه شده اعمال نمی‌شود و قابل اجرا نخواهد بود.

جدول ۴-۶: نگاشت توابع قرارداد هوشمند

تابع	عملیات	توضیح نحوه عملکرد	جزئیات کد
queryContract()	واکشی یک بلوک	در این تابع با ارسال آیدی قرارداد، بلوک قرارداد مورد نظر که در بلاکچین ذخیره شده است واکشی می‌شود	پیوست ۴
addContract()	اضافه کردن قرارداد	در این تابع اطلاعات یک قرارداد جدید در یک بلوک قرار گرفته و در بلاکچین ذخیره می‌شود	پیوست ۵
queryAllContract()	واکشی تمام بلوک‌ها	در این تابع تمام بلوک‌های مربوط به قراردادها واکشی می‌شود	پیوست ۶

بعد از پیاده‌سازی قرارداد هوشمند، لازم است چارچوب Hyperledger Fabric بر روی سرور (یا چندین سرور) نصب شود. در این پروژه به دلیل محدودیت سرور، چارچوب مورد نظر بر روی یک سرور راه‌اندازی شده است. در ابتدا لازم است زنجیره بلاکچین به همراه کانال ارتباطی آن ساخته شود. سپس قرارداد هوشمند تعریف شده بر روی سرور نصب می‌شود. با اجرای فایل startFabric.sh تمام عملیات مورد نیاز برای نصب بلاکچین بصورت خودکار صورت می‌گیرد. لازم به ذکر است جهت شبیه‌سازی بلاکچین بر روی چند سرور، شبکه مجازی ایجاد شده شامل سه نود، Org1 Or2 Orderer1، است.

بعد از نصب چارچوب Hyperledger Fabric به توضیح جزئیات برنامه سرور می‌پردازیم. قبل از اجرا برنامه سرور لازم است برای دسترسی به قرارداد هوشمند یک کاربر و یک ادمین ثبت نام کنند تا از طریق این دو نوع کاربر اجازه دسترسی به بلاکچین داده شود. برای ثبت نام این دو کاربر، دو فایل enrollAdmin.js و registerUser.js اجرا می‌کنیم.

بعد از اجرای دو فایل ذکر شده، برنامه سمت سرور با اجرای فایل app.js شروع به کار می‌کند. در برنامه سمت سرور سه API جهت برقراری ارتباط با قرارداد هوشمند پیاده‌سازی شده است که در جدول ۵-۶ آمده‌اند. هر سه API در ابتدا لازم است به شبکه بلاکچین متصل شوند که به این منظور تابع configNetwork() پیاده‌سازی شده است.

```
async function configNetwork() {
  // load the network configuration
  const ccpPath = path.resolve(__dirname, '..', '..', 'test-network', 'organizations', 'peerOrganizations', 'org1.example.com', 'connection-org1.json');
  const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'));

  // Create a new file system based wallet for managing identities.
  const walletPath = path.join(process.cwd(), 'wallet');
  const wallet = await Wallets.newFileSystemWallet(walletPath);
  console.log(`Wallet path: ${walletPath}`);

  // Check to see if we've already enrolled the user.
  const identity = await wallet.get('appuser');
  if (!identity) {
    console.log('An identity for the user "appuser" does not exist in the wallet');
    console.log('Run the registerUser.js application before retrying');
    return;
  }

  // Create a new gateway for connecting to our peer node.
  const gateway = new Gateway();
  await gateway.connect(ccp, { wallet, identity: 'appuser', discovery: { enabled: true, asLocalhost: true } });

  // Get the network (channel) our contract is deployed to.
  const network = await gateway.getNetwork('mychannel');

  // Get the contract from the network.
  const contract = network.getContract('cayload');

  return {contract, gateway};
}
```

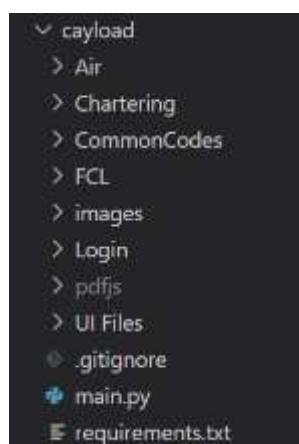
شکل ۵-۶: تابع configNetwork جهت برقرار ارتباط با شبکه بلاکچین

جدول ۵-۶: نگاشت توابع سمت سرور زیر خدمت Air

جزئیات کد	عملیات	API	
	دریافت یک بلوک	GET	query()
	واکشی تمام بلوک‌ها	GET	queryContract()
	اضافه کردن قرارداد	POST	addContract()

۳-۶- راهنمای فنی – سمت مشتری

در پیاده‌سازی سمت مشتری، زیر خدمت Air با دو زیر خدمت FCL و Chartering در بعضی از بخش‌ها متفاوت هستند که در ادامه به هر یک پرداخته می‌شود. در شکل ۳-۶ پوشه‌بندی پیاده‌سازی سمت مشتری نمایش داده شده است. کدهای مربوط به هر زیر خدمت در پوشه‌ی مربوط به خود که هم‌نام با زیر خدمت است، قرار گرفته‌اند. در پوشه CommonCodes و images فایل‌هایی قرار دارد که در هر سه زیر خدمت به صورت مشترک استفاده می‌شود. همچنین در پوشه Login فایل‌های مربوط به ورود به نرم افزار و پوشه‌ی UI Files شامل تمام UI های نرم افزار می‌باشد که به کمک نرم افزار QT Designer طراحی شده‌اند. فایل اصلی و اجرایی main.py می‌باشد. در آخر، تمام کتابخانه‌های استفاده شده در فایل requirements.txt قرار گرفته است.



شکل ۳-۶: پوشه‌بندی سمت مشتری

در ابتدا برای اجرا برنامه لازم است فرایند زیر اجرا شوند:

۱- نصب پایتون (نسخه ۳ و بالاتر)

۲- نصب کتابخانه virtualenv به کمک ابزار pip

۳- ساخت یک محیط مجازی به کمک دستور زیر جهت ایزوله بودن محیط پروژه از محیط اصلی سیستم جهت جلوگیری از تداخل کتابخانه‌های نصبی

```
pip install virtualenv
```

۴- فعال‌سازی محیط مجازی به دستور زیر (در محیط ویندوز):

```
cd "virtualenv name"\Scripts\activate
```

۵- نصب کتابخانه‌های مورد نیاز برنامه که در فایل requirements.txt قرار گرفته است. برای نصب این موارد می‌توان با اجرا دستور زیر در مسیری که فایل requirments.txt قرار دارد، تمام کتابخانه‌های مورد نیاز را نصب کرد:

```
pip install -r requirments.txt
```

۶- اجرای فایل main.py با اجرای دستور:

```
python main.py
```

هر فایل UI که شامل یک صفحه در نرم افزار است به کمک دستور زیر به فایل پایتون تبدیل می‌شود:

```
python -m PyQt5.uic.pyuic -x [FILENAME].ui -o [FILENAME].py
```

بنابراین در هر فایل، توابعی برای اجرای UI طراحی شده وجود دارد که مهم‌ترین آن `setupUi(self, landing_page)` و `retranslateUi(self, login_dialog)` می‌باشد. در ادامه به جزئیات بیشتر کد این بخش پرداخته شده است.

با اجرا شدن فایل `main.py`، سازنده‌ی مهم‌ترین کلاس با نام `Ui_lanfing_page()` اجرا خواهد شد. این کلاس صفحه‌ی اصلی نرم افزار (شکل ۳-۱) را نشان می‌دهد. طبق شکل ۴-۶، بعد از مقداردهی متغیرهای مورد نیاز،

تابع `show_login_dialog()` فراخوانی می‌شود که به کمک آن صفحه‌ی مربوط به ورود کاربران نمایش داده می‌شود (شکل ۴-۳).

```
def __init__(self):
    super().__init__()
    landing_page = self
    self.setupui(landing_page)
    self.is_login_window_open = False
    self.user_token = None
    self.user_public_key = None
    self.public_key = False
    app.processEvents()
    self.show_login_dialog(landing_page)
```

شکل ۴-۶: سازنده کلاس `Ui_landing_page()`

در این تابع (پیوست ۱۰)، سازنده کلاس مربوط به ورود به نرم افزار فراخوانی می‌شود. بعد از ورود موفقیت آمیز کاربر به نرم افزار، صفحه مربوط به ورود بسته می‌شود که این فرایند، فراخوانی تابع `closeEvent()` را به همراه دارد که جزئیات آن شکل ۵-۶ آمده است.

```
def closeEvent(self, event):
    if self.landingPage.user_token == None:
        self.landingPage.close()
    else:
        username = str(self.username_box.text()).upper()
        self.landingPage.store_keys(username)
        self.landingPage.lets_upload(self.landingPage)
        self.landingPage.show()
```

شکل ۵-۶: تابع اجرایی بعد از اتمام عملیات ورود کاربر

طبق این تابع، اگر ورود کاربر با موفقیت نباشد نرم افزار بسته می‌شود و وارد صفحه اصلی نمی‌شود. در غیر این صورت به ترتیب دو تابع `store_keys()` و `lets_upload()` اجرا می‌شود که فرایندهای اجرایی در جدول ۶-۶ آمده است. در انتها بعد از اجرا این دو تابع صفحه اصلی نرم افزار نمایش داده می‌شود.

جدول ۶-۶: نگاشت توابع اجرایی بعد از ورود موفقیت آمیز کاربر

تابع	نحوه‌ی عملکرد	جزئیات کد
<code>store_keys()</code>	ساخت کلید عمومی و خصوصی و ذخیره سازی آن	پیوست ۱۱

lets_upload()	اگر کاربر برای اولین بار وارد نرم افزار شود، کلید عمومی به سرور ارسال می شود. در غیر این صورت یکسان بودن کلید عمومی ذخیره شده در سرور و سیستم کاربر مورد بررسی قرار میگیرد	پیوست ۱۲
---------------	--	----------

همانطور که در شکل ۳-۱: نمایی از نرم افزار بلاکچین و شکل ۳-۹ مشاهده می کنید، سه دکمه برای سه زیر خدمت FCL، Chartering و Air فعال است. با کلیک کردن هر یک از این دکمه ها، تابع مربوط به آن فراخوانی می شود (جدول ۶-۷). هر یک از این توابع سازنده ی کلاس مربوط به زیر خدمت مورد نظر را فراخوانی می کند.

جدول ۶-۷: نگاشت توابع مربوط به دکمه های نرم افزار تحت دستکتاپ

نام دکمه	تابع فراخوانی شده بعد از کلیک شدن
FCL	fclJob()
Chartering	charteringJob()
Air Cargo	airJob()

فرایند اجرایی هر سه این توابع یکسان است و تنها در API های درخواستی متفاوت هستند. در این توابع، تمام قراردادهای کاربر در زیر خدمت مورد نظر از سرور دریافت می شود. که با توجه به اینکه تنها قراردادهایی که توسط طرفین قرارداد تایید شده اند قابل نمایش در نرم افزار است. در این بخش از فرایند صفحه ای جدید باز می شود که لیست قراردادهای تایید نهایی شده به کاربر نمایش داده می شود. طبق شکل ۳-۶ برای هر قرارداد یک دکمه تعبیه شده که با کلیک کردن بر روی آن جزئیات اطلاعات قرارداد در قالب PDF به کاربر نمایش داده می شود. جزئیات کد این بخش در پیوست ۱۳ آورده شده است که با توجه به یکسان بودن هر سه زیر خدمت در این بخش از کد، تنها کدهای مربوط به بهش Air آورده شده است.

همانطور که گفته شد، زمانی که کاربر بر روی دکمه جزئیات قرارداد کلیک می کند، صفحه ای جدید با محتوای جزئیات قرارداد مذکور در قالب PDF به همراه دکمه ای برای امضای دیجیتال کردن قرارداد باز می شود (شکل

۷-۳) (پیوست ۱۴. کاربر با زدن دکمه signature فرایند اجرایی در دو زیرخدمت FCL و Chartering با زیرخدمت Air متفاوت عمل خواهد کرد.

در دو زیرخدمت FCL و Chartering بعد از زدن دکمه signature، تابع implement اجرا می‌شود که در ابتدا برای اطلاعات قرارداد بلوکی ایجاد می‌شود که در این بین اطلاعات قرارداد با کلید خصوصی کاربر امضا می‌شود؛ سپس بلوک ساخته شده لازم است با الگوریتم اثبات کار استخراج شود. بعد از اینکه بلوک مورد نظر استخراج شد، آن را به زنجیره اضافه می‌کنیم. قبل از ارسال زنجیره جدید به سرور، لازم است زنجیره تولید شده با آخرین نسخه زنجیره ذخیره شده در سرور مقایسه شود تا اگر در بین فرایند امضا شدن بلوک جدید اضافه شده بود فرایند ساخت بلوک مجدد اجرا شود.

```
def implement(self):
    full_chain = self.get_full_chain()
    new_block = self.create_new_block(full_chain)
    full_chain.append(new_block)
    self.consensus(full_chain)
    self.finished.emit('', True)
```

شکل ۶-۶: تابع implement مربوط به زیرخدمت FCL و Chartering

جدول ۶-۸: نگاشت توابع فرایند امضا کردن قرارداد در زیرخدمت FCL و Chartering

توابع	نحوه‌ی عملکرد	جزئیات کد
create_new_block()	ساخت بلوک جدید و امضا کردن قرارداد با کلید خصوصی کاربر	پیوست ۱۵
consensus()	بررسی وضعیت زنجیره تولید شده و آخرین ورژن زنجیره ذخیره شده در سرور	پیوست ۱۶

در زیر خدمت FCL، بعد از زدن دکمه signature، تابع implement اجرا می‌شود که فرایند آن با دو زیر خدمت دیگر متفاوت است. از آنجایی که در این زیرخدمت از چارچوب Hyperledger Fabric استفاده شده تا عملیات ذخیره‌سازی قراردادها در این چارچوب صورت گیرد، دیگر فرایندهای قبلی اجرا نمی‌شود. در اینجا

تنها لازم است ابتدا اطلاعات قرارداد امضا شود؛ سپس داده‌ها طبق آنچه که تعریف شده در قالب دیکشنری قرار گیرد. در انتها دیکشنری مورد نظر به بلاکچین Hyperledger Fabric ارسال می‌شود.

```
def implement(self):
    # The file is in list format
    full_file = self.get_file()

    # Sign contract information by user's private_key
    server_data, blockchain_data = self.sign_contract_data(full_file)

    #Send data to Hyperledger Fabric
    try:
        self.send_data_to_blockchain(blockchain_data)

        self.finished.emit('', True)
    except BaseException as error:
        print(error)
```

شکل ۶-۷: تابع implement مربوط به زیر خدمت Air

جدول ۶-۹: نگاشت توابع فرایند امضا کردن قرارداد و ارسال به شبکه بلاکچین Hyperledger Fabric

توابع	نحوه‌ی عملکرد	جزئیات کد
sign_contract_data()	ایجاد دیکشنری و امضا کردن قرارداد با کلید خصوصی کاربر	پیوست ۱۷
send_data_to_blockchain()	ارسال دیکشنری اطلاعات به بلاکچین	پیوست ۱۸

این بخش مربوط به فریم ورک جدیدی هست که دارم اضافه میکنم در نتیجه کامل نیست:

مورد کاربرد ورود کاربر و ثبت نام مدیر سایت:

• ورود کاربر:

در این نرم افزار دو نوع کاربر، مدیر سایت و کاربر عادی، وجود دارد. در صورتی که کاربر مدیر سایت باشد، با ورود موفقیت آمیز به صفحه مدیریت سایت منتقل می‌شود. در صورتی که کاربر از نوع کاربر عادی باشد، با ورود موفقیت آمیز به صفحه‌ی نرم افزار منتقل می‌شود و در این صفحه قادر است سرویس‌های خود و قراردادهای مربوط به هریک را مشاهده کند.

ورود مدیر سایت: مدیر سایت جهت وارد شدن به صفحه مدیریت سایت لازم است نام کاربری و رمز عبور خود را وارد کند. همچنین لازم است از بین لیست شرکت‌های ثبت نام شده، شرکت خود را انتخاب کند. از آنجایی که کاربر مورد نظر مدیر است لازم است جعبه انتخاب را علامت بزند تا به عنوان مدیر سایت وارد نرم افزار شود. در نهایت با فشردن دکمه login عملیات مربوط به ورود کاربر توسط تابع login که در پیوست ۱۹ نشان داده شده است، اجرا خواهد شد.



شکل ۶-۸: صفحه ورود کاربران به نرم افزار

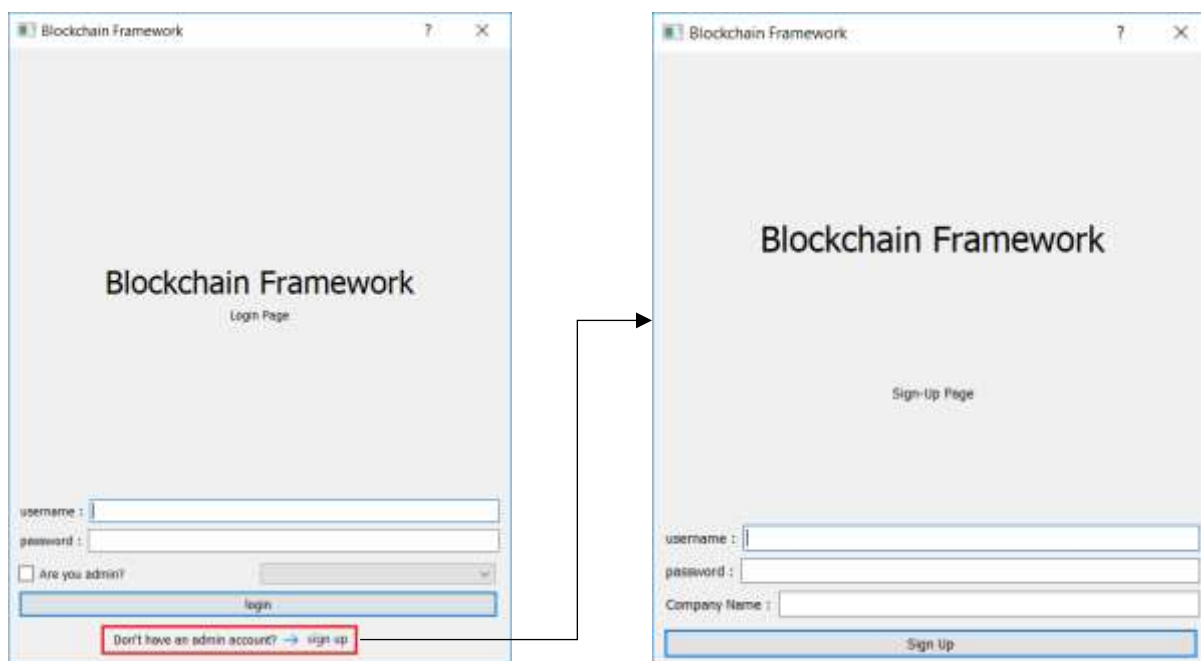
در صورتی که کاربر مورد نظر تمایل به ثبت نام داشته باشد می تواند با فشردن دکمه sign up به صفحه‌ی مربوط به ثبت نام کاربر منتقل شود. با فشردن این دکمه تابع زیر اجرا می شود که در آن از کلاس مربوط به صفحه‌ی ثبت نام یک شی ایجاد شده و آن را اجرا می کند.

```
def signup_page_show(self):
    global closeEvent
    closeEvent = "signup"
    self.close()
    self.signup_dialog = Ui_signup_dialog(self)
    self.signup_dialog.show()
```

شکل ۶-۹: تابع signup_page_show جهت نمایش صفحه ثبت نام مدیر سایت

- ثبت نام مدیر سایت:

در این برنامه لازم است سایت مورد نظر جهت استفاده از آن، ثبت نام کند. بنابراین شخصی به عنوان مدیر سایت می تواند در این نرم افزار ثبت نام و نام شرکت خود را ثبت کند. به این منظور همانطور که در مشاهده می نید، در صفحه ورود کاربران، لینک sign up طراحی شده است که در صورتی که کاربر تمایل داشته باشد می تواند با کلیک بر آن به صفحه‌ی ثبت نام انتقال یابد.



شکل ۶-۱۰: صفحه ورود کاربر و ثبت نام

جهت ثبت نام سایت، تنها لازم است نام آن را ثبت کنند. سایر اطلاعات در صفحه مدیریت از مدیر سایت دریافت می‌شود. زمانی که کاربر دکمه “Sign Up” را فشار دهد تابع signup که در شکل ۱۱-۶ آمده است اجرا می‌شود. در این تابع اطلاعات وارد شده در قالب دیکشنری به سمت سرور ارسال می‌شود. در صورتی که ثبت نام کاربر موفقیت آمیز باشد، سرور کد ۲۰۱ ارسال میکند؛ سپس پیام آن به کاربر نشان داده می‌شود و در نهایت به صفحه‌ی ورود کاربر منتقل می‌شود.

```
def signup(self):
    username = str(self.username_box.text()).strip().replace("\n", "")
    password = str(self.password_box.text())
    site_name = str(self.lineEdit.text())

    url = "http://127.0.0.1:8000/site/"
    json_data = {
        "siteName" : site_name,
        "adminUsername" : username,
        "adminPassword" : password
    }

    req = requests.post(
        url=url,
        json=json_data
    )
    if req.status_code == 201:
        self.close()
        self.signup_success = Ui_signup_message_dialog(self.login_page)
        self.signup_success.show()
    else:
        print("There is a problem")
```

شکل ۱۱-۶: تابع signup جهت ثبت نام مدیر سایت

لازم به ذکر است که نام شرکت ثبت شده باید یکتا باشد و یک شرکت نمی‌تواند دوبار با یک نام یکسان ثبت نام کند. در صورتی که یک سایت جهت ورود کاربران از چندین API استفاده می‌کند، لازم است برای هر API یک بار در نرم افزار و با نامی متفاوت ثبت نام کند.

مورد کاربرد مدیریت سایت:

زمانی که کاربر روی دکمه مربوط به هر سرویس میزنه:

سازنده کلاس service_list_dialog فراخوانی میشود که این کلاس در فایل service_contract_list.py قرار دارد. در این کلاس یک شی از کلاس GetContractListAndCheckSignedWorker ساخته میشود و متد run آن اجرا میشود.

در متد run تمام قراردادهای مربوط به کاربر مورد نظر از سرور دریافت می‌شود. قراردادهای دریافتی باید در قالب تعریف شده باشد که به شرح زیر است:

```
{
  "services": [
    /* ONE CONTRACT */
    {
      "id" : "INT",
      "blockchain_chain_file" : "BOOLEAN",
      "number_of_user" : "INT",
      "users" : {
        "user_(ID)" : {
          "signed" : "BOOLEAN", /* This field is fill by
application */
          "id" : "USER_ID"
        }
        /* Based on the number of users, there are user_(
dictionaries */
      },
      "date_created" : {
        "yaer": "",
        "month": "",
        "day": ""
      },
      "data" : {
        /**/
      },
    },
  ],
}
```

```

        "signed" : "(FULL OR SIGNED OR NOT)" /* This field is fill by
application */
    }
],
    "token" : "USER TOKEN"
}

```

زمانی که قراردادهای دریافت شد، به ازای هر قرارداد بررسی میشود که آیا تمام کاربران قرارداد مذکور را تایید نهایی کرده‌اند یا نه. در صورتی که قرارداد توسط تمام طرفین قرارداد تایید نهایی شده باشد باید مقدار متغیر blockchain_chain_file برابر True باشد. اگر مقدار آن False باشد در اینصورت این قرارداد در لیست قراردادهای کاربر نمایش داده نمی‌شود.

در صورتی که قرارداد قابلیت نمایش در لیست کاربر را داشته باشد، با توجه به تعداد منعقد کنندگان قرارداد و افرادی که آن را امضای دیجیتال کرده‌اند، به قرارداد مذکور متغیری تحت عنوان signed اضافه میشود که میتواند سه مقدار FULL، USER و NOT داشته باشد. مقدار FULL برای زمانی است که تمام افراد طرفین قرارداد آن را امضای دیجیتال کرده باشند، مقدار USER زمانی است که تمام افراد امضا نکرده‌اند ولی کاربر وارد شده به نرم افزار قرارداد مذکور را امضا کرده است و مقدار NOT زمانی است که تمام افراد و خود کاربر امضا نکرده باشند.

بعد از انجام این عملیات، لیست تمام قراردادهایی که قابلیت نمایش در لیست کاربر دارند به عنوان خروجی پاس داده میشود.

بعد از اجرای تابع run، تابع after_data_download اجرا می‌شود. در این متد با توجه به تعداد قراردادهای ردیف در لیست کاربر ایجاد می‌شود. در این تابع، به ازای هر قرارداد یک شی از کلاس MyQWidgetItem ساخته میشود که هر شی شامل اطلاعات قرارداد می‌باشد. در این کلاس برای هر قرارداد دکمه‌ای قرارداده شده است که با کلیک کردن بر روی آن اطلاعات مربوط به هر قرارداد به کلاس Ui_service_dialog پاس داده می‌شود؛ به عبارتی دیگر یک شی از این کلاس ساخته می‌شود و اطلاعات مربوط به قرارداد مذکور به

سازنده‌ی آن پاس داده می‌شود. سپس توابعی دیگری از جمله `init_ui`، `load_progress_bar`، `load_table` و `load_table_header` به ترتیب اجرا می‌شوند.

مراجع

پیوست

```
def valid_hash(chain):
    digest = hashes.Hash(hashes.SHA256())
    for contract in chain[1:]:
        block_data = json.dumps(contract['data'], sort_keys=True) + str(round(contract['timestamp'])) + str(
            contract['proof_of_work'])
        digest.update(block_data.encode())
        hashed = digest.finalize()
        if hashed.hex() == contract['hash']:
            return True
        else:
            return False
```

پیوست ۱. تابع valid_hash در سمت سرور

```
def chain_validity(chain):
    is_chain_valid = True
    for key in range(len(chain)):
        if key == 0:
            continue
        else:
            if chain[key]['previous_hash'] != chain[key - 1]['hash']:
                return False
    return is_chain_valid
```

پیوست ۲. تابع chain_validity در سمت سرور

```
def sign_validity(chain):
    for contract in chain[1:]:
        block_signature = contract['signature']
        cipher = contract['hash']
        sign = base64.b64decode(block_signature['sign'])
        public_key = public_key_load(block_signature['public_key'])
        is_sign_valid = True
        try:
            public_key.verify(
                sign,
                cipher.encode(),
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )
        except InvalidSignature:
            is_sign_valid = False
    return is_sign_valid
```

پیوست ۳. تابع sign_validity در سمت سرور

```

async queryContract(ctx, contractNumber) {
  const contractAsBytes = await ctx.stub.getState(contractNumber);
  if (!contractAsBytes || contractAsBytes.length === 0) {
    throw new Error(`${contractNumber} does not exist`);
  }
  console.log(contractAsBytes.toString());
  return contractAsBytes.toString();
}

```

پیوست ۴. تابع queryContract در قرارداد هوشمند

```

async addContract(ctx, contractNumber, data, data_signed, username, public_key) {
  console.info('===== START : Create contract =====');
  const contract = {
    data: data,
    docType: 'contract',
    signature: {
      data_signed: data_signed,
      username: username,
      public_key: public_key
    }
  };
  await ctx.stub.putState(contractNumber, Buffer.from(JSON.stringify(contract)));
  console.info('===== END : Create contract =====');
}

```

پیوست ۵. تابع addContract در قرارداد هوشمند

```

async queryAllContracts(ctx) {
  const startKey = '';
  const endKey = '';
  const allResults = [];
  for await (const {key, value} of ctx.stub.getStateByRange(startKey, endKey)) {
    const strValue = Buffer.from(value).toString('utf8');
    let record;
    try {
      record = JSON.parse(strValue);
    } catch (err) {
      console.log(err);
      record = strValue;
    }
    allResults.push({ Key: key, Record: record });
  }
  console.info(allResults);
  return JSON.stringify(allResults);
}

```

پیوست ۶. تابع queryAllContracts در قرارداد هوشمند

```

exports.query = async (req, res) => {
  try {
    const {contract, gateway} = await configNetwork();
    const result = await contract.evaluateTransaction('queryAllContracts');
    res.status(200).json({response: result.toString() });

    // Disconnect from the gateway.
    await gateway.disconnect();
  } catch (error) {
    console.error(`Failed to evaluate transaction: ${error}`);
    res.status(500).json({error: error});
    process.exit(1);
  }
};

```

پیوست ۷. GET API برای واکشی تمام بلوک‌ها

```

exports.queryContract = async (req, res) => {
  try{
    const contractNumber = req.params.contractNumber;
    const {contract, gateway} = await configNetwork();
    const result = await contract.evaluateTransaction('queryContract', contractNumber);
    res.status(200).json({response: result.toString() });

    // Disconnect from the gateway.
    await gateway.disconnect();
  }
  catch (error) {
    console.error(`Failed to evaluate transaction: ${error}`);
    res.status(500).json({error: error});
    process.exit(1);
  }
};

```

پیوست ۸. GET API برای واکشی یک بلوک خاص

```

exports.addContract = async (req, res) => {
  try{
    const key = 'CONTRACT' + req.body.key;
    const data = req.body.data;
    const data_signed = req.body.signature.data_signed;
    const username = req.body.signature.username;
    const public_key = req.body.signature.public_key;
    // console.log(req.body)
    const {contract, gateway} = await configNetwork();
    await contract.submitTransaction('addContract', key, data, data_signed, username, public_key);
    res.status(200).json({response: "Contract added"});

    // Disconnect from the gateway.
    await gateway.disconnect();
  }
  catch (error) {
    console.error(`Failed to evaluate transaction: ${error}`);
    res.status(500).json({error: error});
    process.exit(1);
  }
};

```

پیوست ۹. POST API برای اضافه کردن قرارداد جدید به شبکه بلاکچین

```
def show_login_dialog(self, landing_page):
    self.login_dialog = Ui_login_dialog(landing_page)
    self.login_dialog.exec_()
```

پیوست ۱۰. تابع `show_login_dialog` جهت اجرای صفحه ورود به نرم افزار تحت دسکتاپ

```
def store_keys(self, username):
    if self.user_token != None:
        with open('token.bin', 'wb') as token_file:
            token_file.write(self.user_token.encode())
        token_file.close()
        with open('user.bin', 'wb') as user_file:
            user_file.write(username.encode())
        user_file.close()
        ps, pu = self.generate_private_public_key()
        self.store_private_key(username, ps)
        self.store_public_key(username, pu)
```

```
def generate_private_public_key(self): ...
```

```
def store_private_key(self, username, ps): ...
```

```
def store_public_key(self, username, pu): ...
```

پیوست ۱۱. تابع `store_keys` و توابع مورد نیاز آن

```
def lets_upload(self, landing_page):
    """
    If the user is login to the software the first time,
    he should send his public key to the server.

    "user_public_key" variable is initialized when the user is login to the system.
    if it is the first time, the server returns "None" as a public key.
    """
    if self.check_public_key():
        if self.is_public_key_same():
            self.pu_key = True
        else:
            self.pu_key = False
            self.pv_warning_dialog = Ui_pv_warning_dialog(landing_page)
            self.pv_warning_dialog.exec_()
    else:
        self.send_public_key_to_server()
        self.pu_key = True

def check_public_key(self): --
def is_public_key_same(self): --
def send_public_key_to_server(self): --
```

پیوست ۱۲. تابع `lets_upload` و توابع مورد نیاز آن

پیوست ۱۳.

با کلیک کردن بر روی دکمه Air Cargo (یا سایر دکمه‌های مربوط دو زیرخدمت دیگر) تابع `airJob` اجرا می‌شود. که به دنبال آن تابع `run` اجرا می‌شود که در شکل زیر آورده شده است. در این تابع ابتدا تمام قراردادهای مربوط به Air دریافت می‌شود. با بررسی هر قرارداد، در صورت تایید نهایی قرار داد مورد نظر به لیست اضافه می‌شود تا در صفحه لیست قراردادها نمایش داده شود. همچنین در این تابع وضعیت قرارداد از جهت امضای دیجیتال شدن مورد بررسی قرار می‌گیرد.

```
def run(self):
    user_type = self.read_user_type()
    user_id = self.read_user_id()
    user_token = self.read_user_token()
    c_list = []
    if user_type == 'customer':
        url = "https://dev5.cayload.com/api/avl/air_cargo_customer_data_quotations/customer_list"
    elif user_type == 'provider':
        url = "https://dev5.cayload.com/api/avl/air_cargo_customer_data_quotations?provider={}".format(user_id)
    else:
        print('User Type is not correct.')

    headers = {
        "Content-Type": "application/json",
        "Authorization": "Token {}".format(user_token)
    }

    contracts = requests.get(
        url,
        headers=headers
    ).json()

    air_contracts = contracts['air_cargo_customer_data_quotations']

    if user_type == 'customer':
        for contract in air_contracts:
            if contract['blockchain_chain_file']:
                c_list.append(contract)
                chain_file_url = "https://dev5.cayload.com/" + str(contract['blockchain_chain_file']['file'])
                chain = self.get_chain(chain_file_url)
                correspond = contract['provider']['email']['key'] or \
                    contract['provider']['phone']['key']

                own = contract['service']['customer']['email']['key'] or \
                    contract['service']['customer']['phone']['key']
                correspond_sign = self.corresponds_sign(own, correspond, chain)
                if correspond_sign == 'Full':
                    contract['signed'] = 'Full'
                if correspond_sign == 'user':
                    contract['signed'] = 'signed'
                if correspond_sign == 'Not':
                    contract['signed'] = 'not'
            self.finished.emit(c_list)

    elif user_type == 'provider':
        for contract in air_contracts:
            if contract['blockchain_chain_file']:
                c_list.append(contract)
                chain_file_url = "https://dev5.cayload.com/" + str(contract['blockchain_chain_file']['file'])
                chain = self.get_chain(chain_file_url)
                correspond = contract['service']['customer']['email']['key'] or \
                    contract['service']['customer']['phone']['key']
                own = contract['provider']['email']['key'] or \
                    contract['provider']['phone']['key']
                correspond_sign = self.corresponds_sign(own, correspond, chain)
                if correspond_sign == 'Full':
                    contract['signed'] = 'Full'
                if correspond_sign == 'user':
                    contract['signed'] = 'signed'
                if correspond_sign == 'Not':
                    contract['signed'] = 'not'
            self.finished.emit(c_list)

def read_user_type(self):
    pass

def read_user_id(self):
    pass

def read_user_token(self):
    pass
```

در شکل زیر، تابع `corresponds_sign` وضعیت قرارداد مورد بررسی قرار می‌گیرد تا مشخص شود قرارداد مورد نظر بطور کامل توسط کاربران امضای دیجیتال شده، هیچ یک امضا نکرده‌اند یا امضا کننده قرارداد کاربر وارد شده می‌باشد.

```
def corresponds_sign(self, user, correspond, chain):
    """
    Checking the number of signs.
    If both sides of the contract sign it, the contract has been signed FULLY. (=Full)
    If one side of the contract sign it who is login, the contract has been signed by the user. (=user)
    If no one signs the contract, the contract has not been signed. (=Not)
    """
    count = 0
    user_flag = False
    # chain = chain[0]
    if len(chain) == 1:
        return 'Not'
    else:
        for contract in chain[1:]:
            if contract['signature']['user'].upper() == correspond.upper():
                count += 1
            if contract['signature']['user'].upper() == user.upper():
                count += 1
                user_flag = True
        if count == 2:
            return 'Full'
        elif user_flag:
            return 'user'
        else:
            return 'Not'

def get_chain(self, chain_file_url):...
```

پیوست ۱۴.

کاربر با کلیک کردن بر دکمه Contract Details، تابع run که جزئیات آن در شکل زیر آمده است اجرا می‌شود. در این تابع بعد بررسی‌های لازم، PDF مربوط به قرارداد با تابع create_contract_pdf ساخته می‌شود.

```
def run(self):
    is_contract_sign = 'False'
    is_bc_file_exist = False
    user = self.user.decode('UTF-8')
    chain = self.get_full_chain()
    for contract in chain[1:]:
        if contract['signature']['user'].upper() == user.upper():
            is_contract_sign = 'True'
    if self.is_bc_file_stored():
        fcl_data = 'bc exist'
        data = self.fetch_bc_data()
        self.create_contract_pdf(data)
        is_bc_file_exist = True
        local_chain = self.get_local_chain()

    if not self.is_bc_file_stored() and is_contract_sign == 'True':
        self.store_bc_file(chain)
        data = self.fetch_bc_data()
        self.create_contract_pdf(data)
        is_bc_file_exist = True
    else:
        fcl_data = 'bc not exist'
        self.create_contract_pdf('none')

    self.finished.emit(is_contract_sign, is_bc_file_exist)
```

```
def get_local_chain(self): ...
```

```
def is_bc_file_stored(self): ...
```

```
def store_bc_file(self, chain): ...
```

```
def fetch_local_contracts(self, type): ...
```

```
def create_contract_pdf(self, data): ...
```

```
def none_check(self, value): ...
```

```
def fetch_bc_data(self): ...
```

```
def decrypt_blockchain_file(self, file_name): ...
```

```
def decrypt_with_private_key(self, cipher_message): ...
```

```

def create_new_block(self, chain):
    user = self.read_user().decode('UTF-8')
    block_data, correspond_len = self.implement_block_data()
    nonce = 0
    epoch = time()
    chain_data = json.dumps(block_data, sort_keys=True)
    chain_data = chain_data + str(round(epoch))
    block_hash = self.hash_block(chain_data + str(nonce))
    if block_hash[-2:] != '00':
        while block_hash[-2:] != '00':
            nonce += 1
            block_hash = self.hash_block(chain_data + str(nonce))
    signature = self.signature(block_hash).decode('UTF-8')
    public_key = self.decrypt_public_key_file()
    block = {
        "index": len(chain) + 1,
        "timestamp": epoch,
        "data": block_data,
        "signature": {"sign": signature, "user": user, "public_key": public_key.decode('UTF-8')},
        "previous_hash": chain[-1]['hash'],
        "hash": block_hash,
        "proof_of_work": nonce
    }
    return block

def implement_block_data(self): ...

def public_key_load(self, public_key_pem): ...

def encrypt_with_public_key(self, public_key_load, key): ...

def hash_block(self, data): ...

def signature(self, hash): ...

def decrypt_public_key_file(self): ...

def create_symmetric_key(self, username): ...

```

پیوست ۱۵. ایجاد بلوک جدید و امضا کردن قرارداد در زیر خدمت FCL و Chartering

```

def consensus(self, chain):
    full_chain = self.get_full_chain()
    if (len(full_chain) > len(chain)) and chain[len(chain) - 1]['timestamp'] < full_chain[len(full_chain) - 1][
        'timestamp']:
        regenerated_block = self.create_new_block()
        full_chain.append(regenerated_block)
        user = self.read_user()
        symmetric_key = self.create_symmetric_key(user.decode())
        fernet = Fernet(symmetric_key)
        encrypted = fernet.encrypt(str(full_chain).encode())
        identify = str(self.data['id'])
        with open('chartering/contracts_bc/contract_' + str(self.data['id']) + '_' + self.id + '.bin', 'wb') as user_file:
            user_file.write(encrypted)
        user_file.close()
        list_cahin = []
        list_cahin.append(chain)
        url = "https://dev5.cayload.com/api/avl/chartering_customer_data_inquirys/{id}/blockchain_on_sign".format(
            id=identify)
        user_token = self.token.decode('UTF-8')
        r = requests.post(url, json={"contract": list_cahin, "passer": 5},
            headers={"Content-Type": "application/json",
                "Authorization": "Token {}".format(user_token)})
    else:
        user = self.read_user()
        symmetric_key = self.create_symmetric_key(user.decode())
        fernet = Fernet(symmetric_key)
        encrypted = fernet.encrypt(str(chain).encode())
        with open('chartering/contracts_bc/contract_' + str(self.data['id']) + '_' + self.id + '.bin', 'wb') as user_file:
            user_file.write(encrypted)
        user_file.close()

        user_token = self.token.decode('UTF-8')
        identify = str(self.data['id'])
        list_cahin = []
        list_cahin.append(chain)
        # print(list_cahin)
        # print(type(list_cahin))
        url = "https://dev5.cayload.com/api/avl/chartering_customer_data_inquirys/{id}/blockchain_on_sign".format(
            id=identify)
        r = requests.post(url, json={"contract": chain, "passer": 5},
            headers={"Content-Type": "application/json",
                "Authorization": "Token {}".format(user_token)})

```

پیوست ۱۶. بررسی وضعیت زنجیره تولید شده و زنجیره ذخیره شده در سرور

```

def sign_contract_data(self, latest_file):
    username = self.read_user().decode('UTF-8')
    raw_contract_data, contract_id = self.implement_data()
    index = str(len(latest_file) + 1)
    contract_data = json.dumps(raw_contract_data)
    public_key = self.decrypt_public_key_file()
    contract_data_signed = self.signature(contract_data).decode("utf-8")
    server_data = {
        "index": index,
        "key": contract_id + '_' + index,
        "data": raw_contract_data,
        "signature": {
            "sign" : contract_data_signed,
            "user": username,
            "public_key": public_key,
        },
        "previous_hash": "",
        "hash": "",
        "proof_of_work": "",
        "timestamp": time()
    }
    blockchain_data = {
        "key": str(contract_id + '_' + index),
        "data": raw_contract_data,
        "signature": {
            "data_signed" : str(contract_data_signed),
            "username": str(username),
            "public_key": public_key,
        }
    }
    return server_data, blockchain_data

```

```
def implement_data(self):...
```

```
def public_key_load(self, public_key_pem):...
```

پیوست ۱۷. امضا کردن قرارداد و ساخت دیکشنری اطلاعات آن در زیر خدمت Air

```

def send_data_to_blockchain(self, contract_data):
    url = 'https://b1.cayload.com:443/data/add'
    headers = {
        "Content-Type": "application/json",
    }
    result = requests.post(url, json=contract_data)
    return result

```

پیوست ۱۸. ارسال داده‌های قرارداد به بلاکچین در زیر خدمت Air

```

def login(self):
    username = str(self.username_box.text()).strip().replace("\n", "")
    password = str(self.password_box.text())
    site_name = str(self.company.currentText())
    if self.is_admin.isChecked():
        self.user_type = 'admin'
        url = 'http://127.0.0.1:8000/admin_login/'
        json_data = {
            "siteName" : site_name,
            "adminUsername" : username,
            "adminPassword" : password
        }
        status_code, result = self.check_connection(url=url, json=json_data, timeout=5, method="get")
        if status_code == 200 and result['exist'] == True:
            site_id = result['site_id']
            login_id = result['login_id']
            blockchain_id = result['blockchain_id']
            self.close()
            self.landingPage.show_admin_page(site_id, login_id, blockchain_id)
        else:
            self.exec_login_failed_message_dialog()

    else:
        self.user_type = 'user'
        url = 'http://127.0.0.1:8000/user_login/'
        json_data = {
            "siteName" : site_name
        }

        status_code, result = self.check_connection(url=url, json=json_data, timeout=5, method="get")
        if status_code == 200:
            login_url = result['url']
            login_json = result['data'].split(',')
            data = {
                login_json[0] : username,
                login_json[1] : password
            }
            status_code, site_result = self.check_connection(url=login_url, json=data, timeout=5, method='post')
            if status_code == 200:
                self.landingPage.user_token = site_result['token']
                self.save_token(site_result['token'])
                self.save_id(site_result['customer']['id'])
                self.close()
                self.landingPage.show_caution()
                self.landingPage.show_client_page(site_id=result['site_id'])
            else:
                print("wrong")

        else:
            print("wrong")
            self.exec_login_failed_message_dialog()
            self.close()

```

پیوست ۱۹. تابع login جهت ورود کاربران