دانشگاه شهید بهشتی

دانشکده مهندسی و علوم کامپیوتر

پیاده‌سازی سرویس ابری مدیریت دستگاه‌های موبایلی[1]

پروژه کارشناسی مهندسی کامپیوتر
گرایش نرم افزار

دانشجو:
عباس یزدان‌مهر

استاد راهنما:
استاد مهران علیدوست‌نیا

تابستان ۱۴۰۳

[1] MDM = Mobile Device Management

چکیده

در پروژه حاضر، هدف اصلی پیاده‌سازی بخش سرویس ابری سمت سرور در چرخه مدیریت دستگاه‌های موبایلی (MDM) است. این پیاده‌سازی به منظور نظارت، مدیریت و کنترل کلاینت‌های‌موبایلی صورت می‌گیرد و شامل دسترسی امن و مدیریت متمرکز به برنامه‌ها و دادن دسترسی به کاربران می‌شود. از جمله ویژگی‌های اصلی این پروژه، احراز هویت کاربران از طریق سرویس‌های ابری، نظارت بر دسترسی‌ها و قابلیت تنظیم مرکزی تنظیمات و اجازه‌ها برای کلاینت‌ها است. چالشی که وجود دارد وجود یک سامانه یکپارچه و ساده با قابلیت انعطاف بالا و توسعه‌پذیری بالا همراه با داشتن تعداد کاربران زیاد است که این در این پروژه سعی بر حل آن داریم. موارد مشابه این پروژه برخی یکپارچه نیستند و برخی خیلی انعطاف بالایی ندارند و همچنین در این پیاده سازی تلاش شده است تا نسبت به موارد مشابه، سادگی در توسعه در نظر گرفته شود و همچنین سادگی در انجام مدیریت‌ها هم صورت گیرد و تمام فرآیندهای‌ممکن بصورت خودکار و حتی گاها زمان‌بندی‌شده صورت گیرد. در این پروژه با استفاده از معماری نرم افزاری که ارائه می‌شود تلاش در حل این مشکلات و نیازها داریم. به کمک این پیاده‌سازی، امکان بهبود امنیت و بهره‌وری در محیط‌های مدیریتی فراهم می‌شود، که به بررسی جزئیات این فرآیند و اهمیت آن در محیط‌های سازمانی می‌پردازیم.

واژگان کلیدی: Mobile, Operating System, API, Cloud, Backend, Device Management

فهرست مطالب

فهرست شکل‌ها

فصل اول: کلیّات

۱-۱ مقدمه

در این بخش یک معرفی اولیه از پروژه انجام شده آورده شده است. در ادامه پس از بیان مسئله پروژه و توضیح کلی به معرفی نوع روش حل مسئله می پردازیم.

۱-۲ بیان مسئله

در دنیای امروزی که استفاده از دستگاه‌های موبایل و تکنولوژی‌های اینترنت اشیاء[۲] به سرعت در حال گسترش است، مدیریت امنیت و دسترسی به این دستگاه‌ها از اهمیت بسیاری برخوردار است. این مسئله بیشترین تأثیر خود را بر سازمان‌ها و بخش‌های صنعتی که بسترهای پیچیده‌ای از دستگاه‌های متصل به شبکه مدیریت می‌کنند، به ویژه در حوزه‌های حساس مانند بهداشت[۱]، تولید و امنیت دارد.

این مسئله اساساً در مرحله پیاده‌سازی و اجرای سیستم‌های MDM و خدمات ابری سمت سرور برای اتصال به دستگاه‌های IoT رخ داده است. با ظهور فناوری‌های جدید و افزایش تعداد و تنوع دستگاه‌های متصل، نیاز به راه‌حل‌هایی که امنیت این ارتباطات را تضمین کنند، چالش‌های بیشتری به وجود آمده است.

این مسئله از سال‌های اخیر با افزایش نیاز به استفاده از دستگاه‌های موبایلی و تکنولوژی‌های اینترنت اشیاء برای بهبود عملکرد، کاهش هزینه‌ها و افزایش بهره‌وری، به شدت به نمایش در آمده است. این نیاز باعث شده است که سازمان‌ها به دنبال راه‌حل‌هایی برای مدیریت امنیت و دسترسی به دستگاه‌های متصل باشند، به ویژه در مواجهه با چالش‌هایی همچون مدیریت مرکزی دسترسی‌ها و انطباق با معماری‌های مختلف دستگاه‌ها.

راه‌حل‌های پیشنهادی کنونی اغلب با محدودیت‌هایی مانند عدم یکپارچگی، پیچیدگی در تنظیمات و مدیریت دسترسی‌ها، و عدم انعطاف و توسعه‌پذیری روبه‌رو هستند. بهبود این راه‌حل‌ها از طریق ارائه پیاده‌سازی سیاست‌های دسترسی پویا و هوشمند و همچنین پیاده‌سازی ساده با اسناد کافی امری ضروری است تا عملکرد بهتری در این حوزه و برای نیازمندی هایی که بیان می شود فراهم شود.

[۲] Internet of Things

۱-۳ کلیات روش پیشنهادی

با استفاده از بررسی موارد موجود که بصورت متن‌باز وجود دارند و پیدا کردن مشکلات آنها سعی می‌کنیم که ابتدا یک معماری نرم‌افزاری مناسب با قابلیت انعطاف بالا و سادگی طراحی کنیم و سپس به پیاده‌سازی و تست آن خواهیم پرداخت.

۱-٤ ساختار پروژه

همانطور که گفته شد در ادامه ابتدا موارد موجود و پروژه‌های مشابه را بطور کامل بررسی می‌کنیم و مشکلات آنها را پیدا می‌کنیم، سپس معماری نرم‌افزار موردنیاز و یکپارچه خودمان را با توجه به نیازها و قابلیت پیاده‌سازی و سادگی طراحی می‌کنیم و در نهایت به پیاده‌سازی و تست نرم‌افزار می‌پردازیم.

فصل دوم: مفاهیم پایه و کارهای مرتبط

۲-۱ مقدمه

۲-۱-۱ تعاریف و مفاهیم

مدیریت دستگاه‌های موبایلی شامل یک سری مفاهیم پایه است که در امنیت و مدیریت دستگاه‌های موبایل و تجهیزات اینترنت اشیاء بسیار حیاتی هستند. در زیر به برخی از این مفاهیم پایه اشاره می‌شود:

- احراز هویت[۳]: احراز هویت به فرآیندی اطلاق می‌شود که در آن هویت واقعی کاربر یا دستگاه تایید می‌شود. در MDM، این فرآیند برای اطمینان از اینکه دستگاه‌های متصل به شبکه سازمانی، همان دستگاه‌هایی هستند که ادعا می‌کنند و دارای حق دسترسی به منابع سازمانی هستند، اساسی است.
- مدیریت دستگاه[۴]: مدیریت دستگاه شامل این است که سازمان‌ها قادر باشند دستگاه‌های متصل به شبکه را از راه دور مدیریت و کنترل کنند. این شامل دسترسی دادن به برخی تجهیزات در دستگاه موبایلی، تغییر برخی تغییرات، ذخیره سازی لاگ‌های کاربران و نظارت بر آنها و ... می‌شود.
- رمزنگاری[۵]: رمزنگاری به فرآیند تبدیل اطلاعات به صورت ناخوانا به یک فرد غیر مجاز اشاره دارد. در مدیریت دستگاه‌های موبایلی، رمزنگاری برای محافظت از داده‌های حساس و اطلاعات شخصی کاربران در دستگاه‌ها ضروری است.
- نظارت[۶]: به فرآیند نظارت بر وضعیت و عملکرد دستگاه‌ها می‌پردازد. این شامل نظارت بر مصرف باتری، فضای ذخیره‌سازی، نصب نرم‌افزارها، استفاده از داده و سایر ویژگی‌های دستگاه می‌شود.
- پاک کردن اطلاعات[۷]: این قابلیت به سازمان‌ها اجازه می‌دهد که در صورت دزدیده شدن دستگاه یا گم شدن آن، به صورت از راه دور تمامی داده‌های موجود در دستگاه را از بین ببرند. این کار باعث حفظ امنیت اطلاعات حساس سازمانی می‌شود.

---

[۳] Authentication
[۴] Device Management
[۵] Encryption
[۶] Monitoring
[۷] Data Wipe

- خط مشی‌های دسترسی[۸]: سیاست‌های دسترسی تعیین می‌کنند که کاربران یا دستگاه‌های متصل به شبکه چه نوع دسترسی‌ها و مجوزهایی به منابع سازمانی دارند. این سیاست‌ها بر اساس نقش‌ها، گروه‌ها و سطوح امنیتی تعیین می‌شوند.

۲-۲ تحلیل نقاط قوت و ضعف منابع غیرپژوهشی مشابه

پروژه‌های نسبتا خوبی در موضوع مدیریت دستگاه‌های موبایلی تا به حال وجود دارد که قابلیت‌های خوب و قابل‌قبولی دارند اما بصورت رایگان در اختیار نیستند و بعضی هم که بصورت رایگان در اختیار هستند بصورت متن‌باز نیستند هر چند که پروژه‌های معدود متن‌باز هم بصورت عمومی وجود دارند. در این قسمت پروژه‌هایی در این زمینه که بصورت متن‌باز هستند را بررسی می‌کنیم، تا بتوانیم از آنها، نیازمندی‌هایمان را همراه با قابلیت پیاده سازی بررسی کنیم و آنها را بهبود ببخشیم در ضمن اینکه ابزارها و چارچوب‌های مناسب را برای پیاده‌سازی کامل‌تر و سریعتر پیدا کنیم.

۲-۲-۱ پروژه Flyve MDM Agent for Android وGLPI Plugin

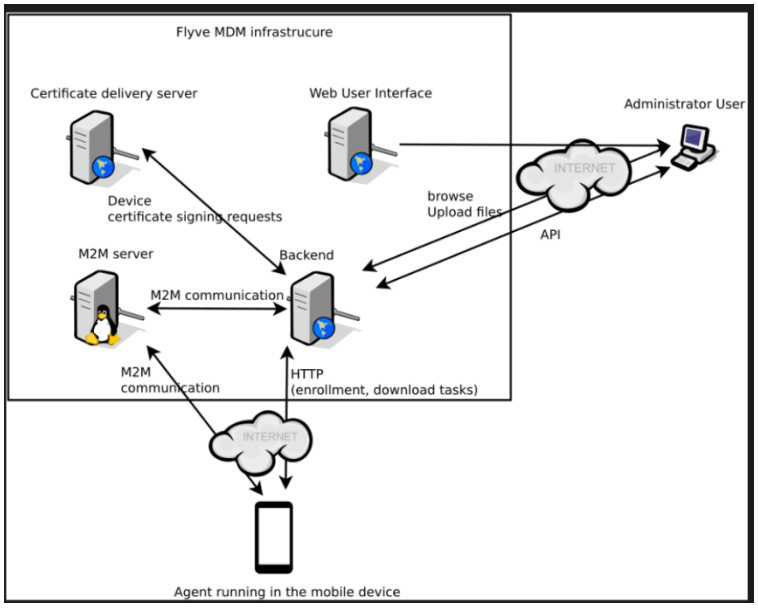
پروژه‌ی Flyve MDM Android Agent یک برنامه‌ی MDM متن‌باز برای سمت کلاینت و بطور خاص برای بستر اندروید است که توسط پروژه‌ی Flyve MDM توسعه داده شده است. البته پروژه Flyve MDM علاوه بر اندروید برای بسترهای دیگر برنامه هایی دارد و در سایت GitHub کدها برای بسترهای windows، iOS، blackberry و web قابل مشاهده است، اما به میزان برنامه اندروید آن توسعه نیافته‌اند. البته اینجا چون پروژه‌ی ما با سمت سرور کار دارد جزئیات برنامه‌های کلاینت برای ما خیلی مهم نیست و صرفا نیاز داریم که نحوه ارتباطات، پروتکل ها، مدل‌ها و کنترل‌های ممکن را مورد بررسی قرار دهیم.

این پروژه برای برنامه سمت سرور از پروژه GLPI که یک برنامه نرم افزاری برای مدیریت منابع IT و متن‌باز و رایگان است استفاده می کند و عملا پروژه Flyve پروتکل ها و مدل‌ها را به گونه ای بکار میگیرد تا با یکپارچه‌سازی با نرم افزار GLPI یک نرم افزار MDM تشکیل دهد.

[۸] Access Policies

در واقع پروژه GLPI خیلی کلی تر از چیزی است که ما نیاز داریم و تمرکز خاصی روی موضوع‌ما ندارد و می‌تواند در پروژه‌ها و کاربردهای بسیار زیادی استفاده شود. پروژه Flyve برای استفاده از GLPI به عنوان برنامه سرور خود یک افزونه برای آن درست کرده است و برخی پروتکل‌های مورد نیاز مثل پروتکل‌های موردنیاز برای ارتباط بی‌درنگ با استفاده از MQTT را خودش به پروژه GLPI اضافه کرده است.

معماری کلی این پروژه برای ارتباط بین دستگاه و سرورها به گونه زیر می‌باشد:



شکل ۱ - معماری کلی ارتباطات Flyve

همانطور که در شکل مشخص است این پروژه از فناوری M۲M استفاده می‌کند به این معنا که نیازی به ارتباط یک انسان برای انجام عمل نیست و دو ماشین یا سیستم بطور خودکار می‌توانند به راحتی با هم ارتباط برقرار کنند و این ارتباط که به آن ارتباط مستقیم هم می گویند، ارتباطی است که هر طرف می‌تواند بدون اینکه طرف مقابل درخواستی داده باشد، درخواست خود را بدهد و عملا یک کانال یا صف بین این دو ماشین یا سیستم برقرار است که هر دو هم می‌توانند پیام خود را در آن بگذارند و هم می توانند پیام خود را بردارند. البته این نکته حائز اهمیت است که هر دو سمت باید یک برنامه یا نخی[۹] را در هر دو سمت داشته باشند برای اینکه کانال ارتباطی را دنبال و یا گوش کنند.

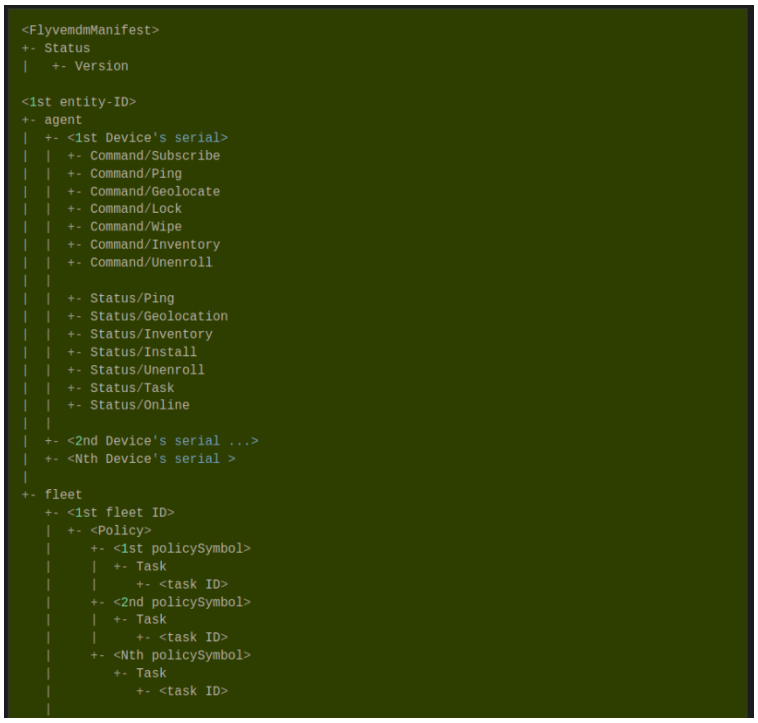
[۹] Thread

برای پیاده سازی M۲M این پروژه از چارچوب MQTT[۱۰] استفاده کرده است. MQTT در واقع یک صف است که همانطور که گفته شد یک طرف پیام خود را در صف می‌گذارد و طرف دیگر که صف را از قبل دنبال می‌کند پیامی که مربوط به او است را برمیدارد. صف‌ها معمولا قابلیت درست کردن موضوعات متفاوت را دارند که به این معناست که هر موضوع عملا صف جدایی می‌شود، MQTT نیز به خوبی این امکان را دارد و در این پروژه هم از این ویژگی تا حد امکان استفاده شده است. موضوعات MQTT بصورت درختی و سلسله مراتبی و با استفاده از علامت "/" مشخص می‌شوند.

صف‌های پروژه Flyve شامل موارد اصلی پیکربندی پروژه (FlyvemdmManifest) و موجودیت (Entity) می‌شود که هر موجودیت شامل یک عامل[۱۱] و یک کنترل‌کننده[۱۲] عامل است. هر عامل می‌تواند در چندین دستگاه مختلف فعال باشد. زیرموضوعات عامل شامل دستورات[۱۳] که درخواست هستند و وضعیت‌ها[۱۴] که پاسخ درخواست هستند می‌شود و کنترل‌کننده عامل شامل زیر موضوع خط‌مشی می‌شود که خط‌مشی‌ها در قالب وظایف[۱۵] در صف قرار می‌گیرند.

---

[۱۰] Message Queue Telemetry Transport
[۱۱] Agent
[۱۲] Fleet
[۱۳] Command
[۱۴] Status
[۱۵] Tasks



شکل ۲ - موضوعات MQTT

هر کدام از صف‌ها قالب پیام مخصوص به خود را دارند و پیامی غیر از قالب خود را نمی‌پذیرند. مثلا قالب پیام دو صف درخواست و پاسخ برای یافتن مختصات دستگاه بصورت زیر است:

Geolocation query

Sub topic `Command/Geolocate`

```
{
    "query" : "Geolocate"
}
```

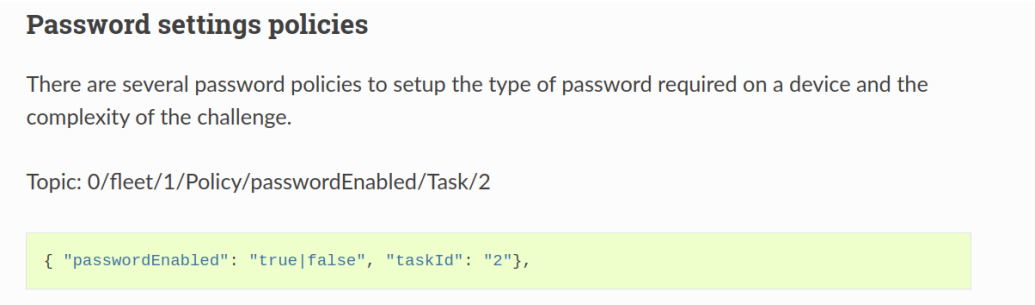
Expected answer:

Sub topic `Status/Geolocation`

```
{"latitude":48.1054276,"longitude":-1.67820699,"datetime":1476345332}
```

شکل ۳ - قالب یکی از موضوعات MQTT

یک مثال از نام کامل موضوع:

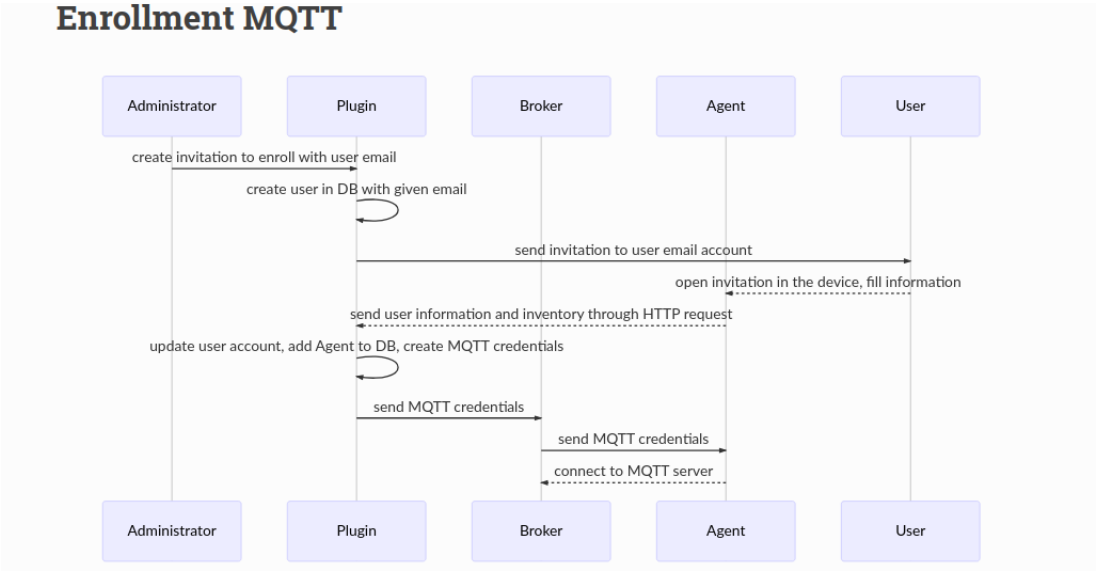


شکل ۴ - یک مثال از یک نام کامل موضوع MQTT

یکی دیگر از موارد برای پیاده‌سازی بهتر M۲M استفاده از FCM یا همان Firebase است که کاربرد اصلی آن فرستادن و مدیریت اعلان‌ها بصورت بلادرنگ و یا زمانبندی شده برای کاربر می‌باشد و یکی از مهم‌ترین قابلیت‌های آن این است که می‌توان وقتی برنامه غیرفعال است ولی کاربر به شبکه متصل است پیام یا اعلان خود را فرستاد و عملا ارتباط آن با دستگاه برقرار می‌شود و نه برنامه کلاینت. این برنامه علاوه بر MQTT از این فناوری هم برای ارتباط مستقیم استفاده کرده است.

فناوری MQTT برای زمانی است که برنامه فعال و در حال استفاده است در حالی که کاربرد اصلی FCM برای زمانی است که برنامه فعال و در حال کار نیست و می‌خواهیم پیامی را به اطلاع کاربر برسانیم.

یکی از مهم‌ترین قسمت‌های این پروژه نحوه ارتباط و شیوه انجام فرآیندها بین موجودیت‌های پروژه است. موجودیت‌های کلی پروژه شامل کاربر، عامل، صف، افزونه GLPI و مدیر می‌شود. که برای هر مورد کاربردی استفاده از این برنامه یک نحوه ارتباطی وجود دارد. مثلا برای مورد کاربردی فرآیند ثبت‌نام نحوه ارتباط به شکل زیر است:



شکل ۵ - یک نمونه ارتباطی بین موجودیت‌ها

مستندات باقی فرآیندها نیز بطور کامل در مستندات این پروژه موجود است. [۲]

ابزارها و فناوری‌ها

- این پروژه برای برنامه مرکزی خود از زبان برنامه نویسی php بصورت خام و بدون چارچوب خاصی استفاده کرده است و صرفا از کتابخانه‌های موجود برای این زبان مثل jquery و phpmailer استفاده می‌کند.
- برای مدیریت پکیج‌های php از composer استفاده می‌کند.
- برای احراز هویت کاربران از تلفیق با LDAP استفاده می‌کند.
- برای وب سرور از apache و بصورت جایگزین از nginx استفاده می‌کند.
- برای پایگاه داده از mariadb یا mysql استفاده می‌کند.
- برای ارسال اعلان‌ها و دستورات از FCM و MQTT استفاده می‌کند.
- برای frontend از زبان javascript به همراه html و css استفاده می‌کند.
- این برنامه هم بصورت مستقیم بر روی لینوکس و هم بصورت مجازی سازی و کانتینرسازی با ابزار داکر قابلیت نصب و استفاده دارد.

امکانات و نقاط قوت

- ثبت‌نام و احراز هویت در همه‌ی پروتکل‌های استفاده شده
- بررسی فعال بودن برنامه کاربر
- فرستادن خط‌مشی ها بصورت بلادرنگ
- دانلود فایل‌های مخصوص دستگاه
- امکان رمزگذاری برای فایل‌های دستگاه
- امکان پاک‌سازی دستگاه (Data Wipe)
- امکان قفل کردن دستگاه از راه دور
- نظارت بر مکان دستگاه

مشکلات و نقاط ضعف

- مقیاس‌پذیری پایینی دارد که یعنی این برنامه در بیشتر قسمت‌های خود قابلیت داشتن کاربر و اطلاعات زیاد را ندارد.
- توسعه‌پذیری پایینی دارد که یعنی فرآیند نصب و تست و ساخت برنامه شامل پیچیدگی‌هایی در وابستگی‌ها و نصب آنهاست.
- نیازمندی به افزونه‌های زیاد برای هر کاربرد که گاهی موجب تضاد بین نسخه‌های مختلف می‌شود.
- پیکربندی پیچیده‌ای دارد.

۲-۲-۲ پروژه Headwind MDM

پروژه‌ی Headwind MDM یا H-MDM یک راه‌حل متن‌باز برای مدیریت دستگاه‌های موبایل (MDM) و برای بستر اندروید است. این پروژه شامل یک سرور مرکزی است که به عنوان هسته مرکزی مدیریت دستگاه‌ها عمل می‌کند

و وظیفه پردازش درخواست‌ها، اجرای دستورات و مدیریت خط‌مشی‌ها را به عهده دارد و از طرفی شامل پیاده‌سازی برنامه کلاینت اندروید که به همین سرور مرکزی متصل می‌شود، می‌باشد.

این پروژه یک رابط کاربری برنامه‌نویسی برای دریافت اطلاعات دارد که از طریق پروتکل HTTP صورت می‌گیرد. مثل پروژه Flyve این پروژه از FCM و MQTT بصورت همزمان استفاده می‌کند و کاربردها هم تقریبا مشابه همان پروژه است یعنی FCM برای فرستادن اعلان‌ها مخصوصا وقتی برنامه کلاینت برخط نیست و MQTT هم برای وقتی برنامه فعال است.

معماری کلی این برنامه به این صورت است که یک سرور مرکزی داریم که با چارچوب Spring توسعه داده شده شامل یک بخش برای REST API، یک بخش برای زمانبندی کارها و فرستادن دستورات و یک داشبورد می‌شود. این پروژه افزونه‌محور است و موردهای کاربردی مدیریت دستگاه‌ها در بخش‌های مختلف بصورت افزونه‌های مختلف پیاده‌سازی شده‌اند.

افزونه‌های اصلی که برای این پروژه وجود دارند شامل افزونه‌هایی برای گرفتن اطلاعات دستگاه، مدیریت لاگ‌های دستگاه، مدیریت و کنترل پیام‌رسانی و فرستادن پیام می‌شود.

هر کدام از افزونه‌ها شامل چند بخش کلی هستند که تقریبا در همه‌ی آنها باید پیاده‌سازی شود که شامل بخش rest که مربوط به API است، بخش persistence که مربوط به MQTT است و بخش task که مربوط به بررسی و مدیریت کارها در آن افزونه است، می‌شوند.

ابزارها و فناوری‌ها

- استفاده از زبان جاوا و چارچوب Spring برای سرور مرکزی و REST API
- maven برای مدیریت پکیج‌ها و ساخت برنامه
- Angular برای توسعه درگاه مدیریتی وب
- MQTT برای ارتباط بلادرنگ دوطرفه
- FCM برای فرستادن اعلان‌ها
- JWT برای مدیریت احراز هویت

امکانات و نقاط قوت

- قابلیت کارکردن با برنامه بدون نیاز به ثبت‌نام
- قابلیت سفارشی‌سازی محیط کار در دستگاه اندرویدی
- استقرار برنامه بصورت خودکار با استفاده از درگاه وب
- پیکربندی و کنترل تنظیمات مختلف شامل GPS, WiFi, Bluetooth
- قابلیت توسعه افزونه‌های دلخواه برای برنامه
- بررسی و کنترل لاگ‌های برنامه

نقاط ضعف

- فقط برای دستگاه‌های اندرویدی توسعه داده شده است.
- پیکربندی و نگهداری آن سخت است.
- مستندات کافی برای این پروژه وجود ندارد.
- ویژگی‌های محدودی دارد.

۳-۲-۲ پروژه MicroMDM

پروژه‌ی MicroMDM یک راه‌حل متن‌باز برای MDM به‌ویژه برای دستگاه‌های اپل (macOS و iOS) است. این ابزار توسط جامعه‌ی توسعه‌دهندگان منبع‌باز توسعه داده شده و هدف آن فراهم کردن یک راه‌حل کارآمد و مقیاس‌پذیر برای مدیریت دستگاه‌های اپل در سازمان‌ها و شرکت‌ها است. البته MicroMDM به عنوان یک نرم‌افزار کامل نیست و عملا یک بستر است که قالب و چارچوب را با استفاده از رابط برنامه نویسی کاربری بستر اپل برای توسعه فراهم می‌کند. که برای استفاده از آن حتما باید به نوعی که کسب و کار ما می‌طلبد آن را توسعه و تغییر داد تا به عنوان یک محصول قابل استفاده باشد.

ابزارها و فناوری‌ها

- این پروژه از زبان برنامه نویسی GoLang و کتابخانه‌های آن برای برنامه مرکزی استفاده کرده است.
- برای وب سرور از خود کتابخانه‌های Go استفاده کرده است.
- برای پایگاه داده از پایگاه داده Postgresql استفاده کرده است.
- برای فرستادن اعلان‌ها و کامندها از APN و Apple MDM Protocol استفاده می‌کند.
- برای استقرار برنامه از docker استفاده می‌کند.
- همچنین برای ساخت و تست برنامه از makefile استفاده می‌کند.

امکانات و نقاط قوت

- امکان مدیریت و کنترل دستگاه‌های MacOS و iOS از راه دور.
- اعمال سیاست‌های امنیتی مانند رمزگذاری دستگاه، قفل دستگاه، و پاک کردن اطلاعات از راه دور.
- نصب و به‌روزرسانی نرم‌افزارها به صورت متمرکز و از راه دور.
- تنظیمات مختلف مانند پروفایل‌های شبکه، ایمیل، و VPN.
- مشاهده وضعیت دستگاه‌ها، از جمله اطلاعات سخت‌افزاری و نرم‌افزاری.
- ارائه API های RESTful برای ارتباط با دستگاه‌ها و ارسال دستورات.
- امکان ادغام با ابزارهای دیگر مانند Ansible و Puppet برای اتوماسیون بیشتر.

نقاط ضعف

- تمرکز زیادی بر روی دستگاه‌های بستر اپل (MacOS - iOS) دارد و عملا برای اندروید کاربرد ندارد.
- پیکربندی پیچیده‌ای دارد.

۲-۳ جمع‌بندی

در این فصل با مفاهیم و سپس با پروژه های موجود در این زمینه آشنا شدیم و همچنین نیازمندی‌های پروژه را جمع‌آوری کردیم. در فصل بعدی به معرفی سرویس ها و معماری ارتباطی بین آنها، فناوری مورد استفاده برای پیاده سازی هر سرویس و ارتباط بین آنها و نحوه پیاده سازی آنها می پردازیم.

فصل سوم: روش پیشنهادی و نتیجه‌گیری

۳-۱ مقدمه

در این فصل بیشتر به طراحی برای پیاده سازی و خود نحوه پیاده سازی بصورت عملی تر و کاربردی تر می پردازیم.

۳-۲ ساختار روش پیشنهادی

۳-۲-۱ مهندسی نیازمندی

با توجه به آنچه که بررسی کردیم و دیدیم حال نیازمندی‌های خودمان را جمع‌آوری کرده و سپس به طراحی و پیاده‌سازی می‌پردازیم. نیازهای کلی ما شامل موارد زیر است:

- مدیریت دسترسی‌ها به دستگاه‌ها
- مدیریت لاگ‌ها
- نظارت و کنترل دستگاه‌ها

برای برآورده کردن نیازهای بالا نیازهای پیاده‌سازی زیر را داریم:

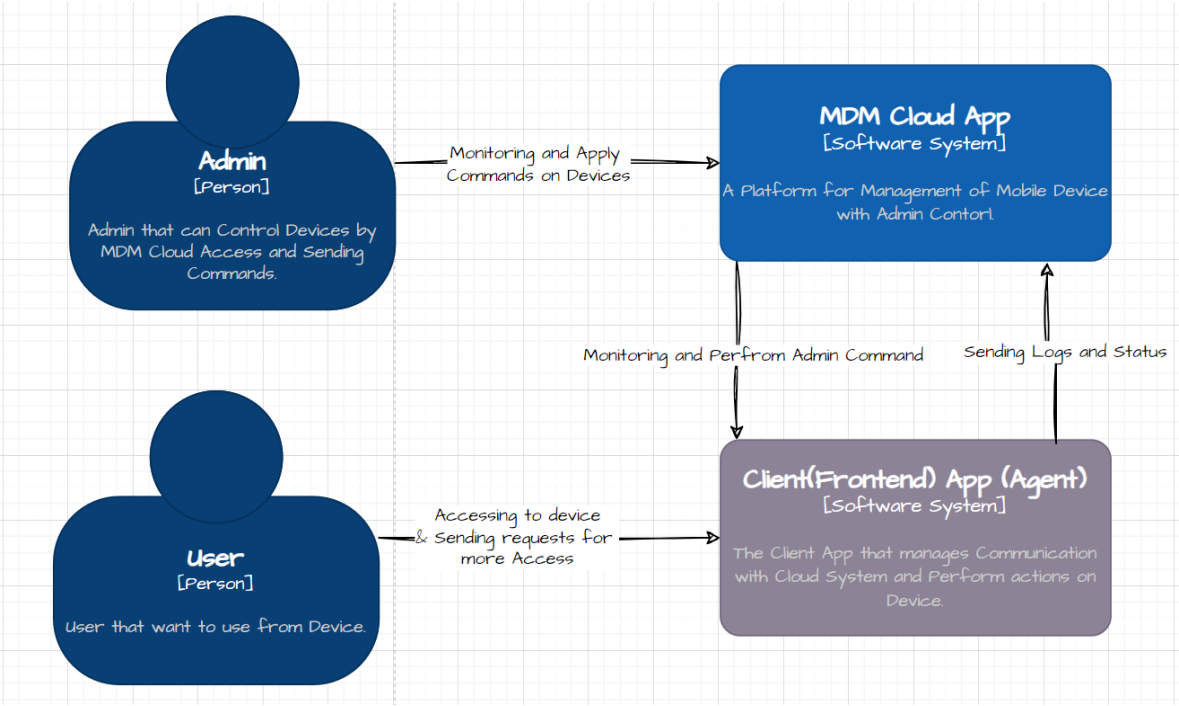
۱. یک سیستم مرکزی برای کنترل و مدیریت سرویس‌های مختلفی که داریم
۲. یک پایگاه داده SQL برای داده‌های موجودیت‌ها و ارتباط برنامه
۳. یک سرویس REST API برای ارتباط HTTP - stateless
٤. یک سرویس برای ارسال اعلان‌ها
٥. یک سرویس برای ارتباط بلادرنگ
٦. یک سرویس برای دریافت لاگ‌های کاربران
۷. یک پایگاه داده NoSQL برای ذخیره‌سازی تعداد زیاد لاگ‌ها
۸. یک داشبورد برای مدیریت[۱۶] همه‌ی موارد بالا

با توجه به نیازمندی انجام شده در ادامه می‌توان معماری و مدل پایگاه داده را طراحی کرد.

[۱۶] Administration

برای فهمیدن دقیق نیازمندی‌ها و پیاده‌سازی آنها از روش C4 استفاده می‌کنیم. در این روش از بیرونی‌ترین قسمت سیستم به سمت جزئیات داخلی سیستم پیش می‌رویم.

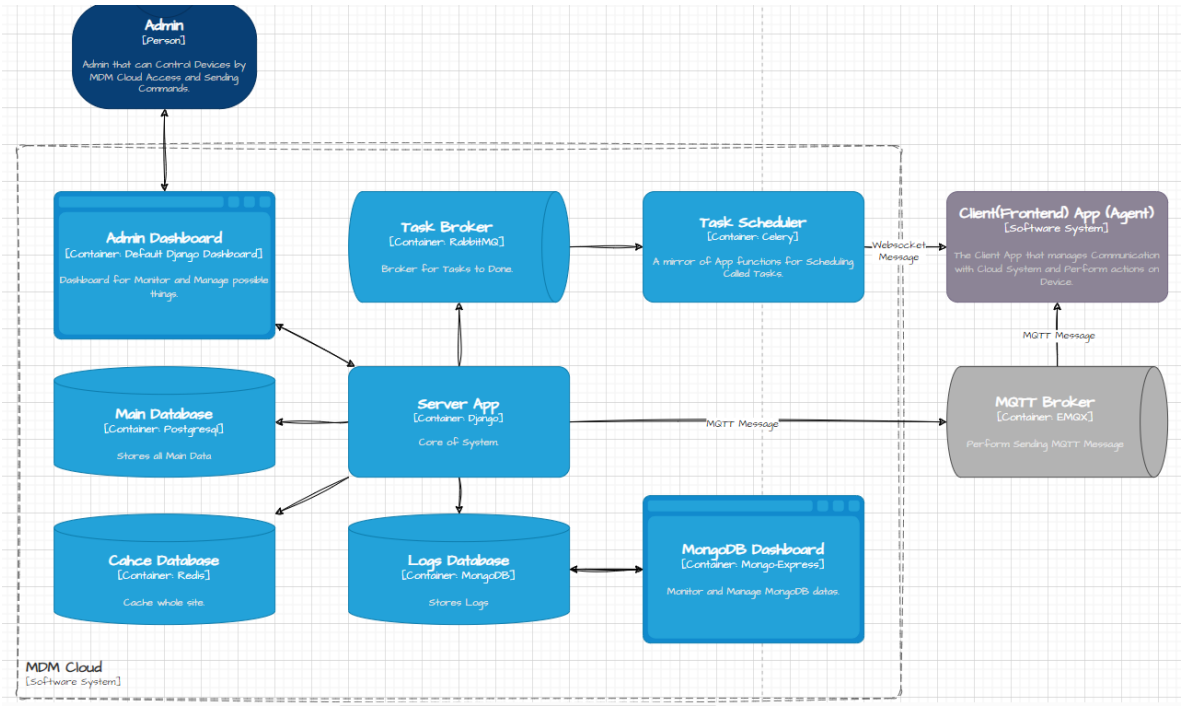
ابتدا نمودار Context را رسم می‌کنیم:



شکل ۶- نمودار Context سیستم

همانطور که مشاهده می‌شود سیستم کلی شامل یک برنامه ابری و یک برنامه سمت کلاینت و شامل دو نوع کاربر مدیر و کاربر دستگاه می‌باشد. که سیستم ما صرفا بخش ابری از این سیستم کلی است.

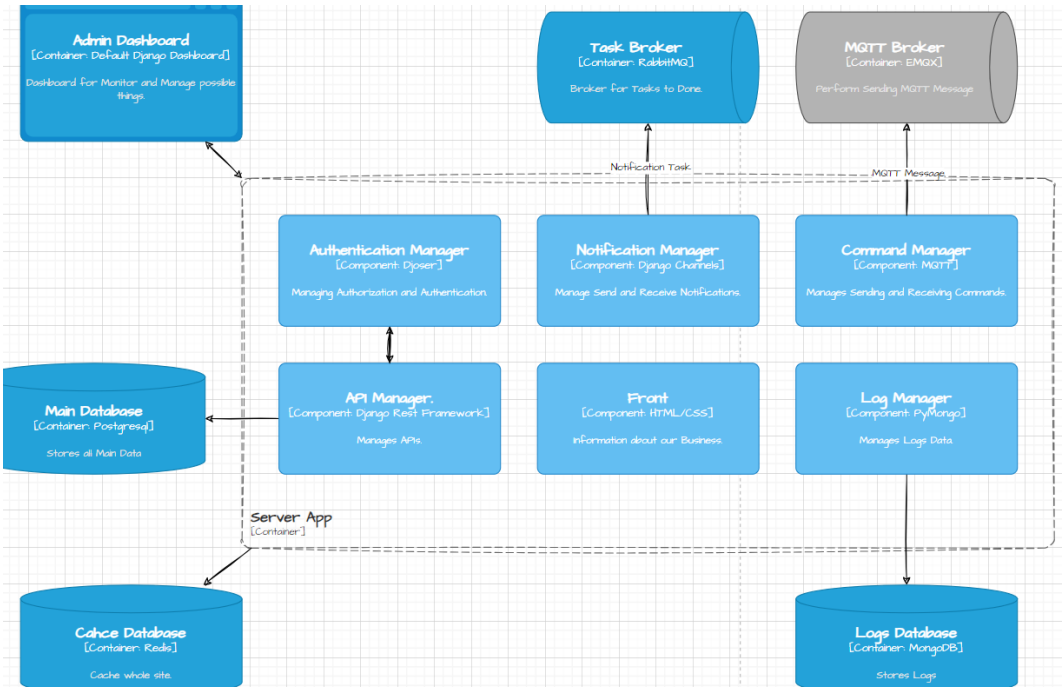
حال باید طبق نیازمندی‌هایی که بدست آوردیم و ذکر کردیم نمودار Container را برای سیستم ابری خود رسم کنیم:



شکل ۷- نمودار Container

همانطور که مشاهده می‌شود در این نمودار تمام بخش‌های اصلی را می‌توان دید که شامل پایگاه‌های داده برای ذخیره سازی اطلاعات اصلی، کش کردن سایت و ذخیره‌سازی لاگ‌ها می‌باشد و همچنین یک داشبورد و یک بخش برای مدیریت وظایف. در این مرحله فناوری‌ها هم انتخاب شده‌اند که در ادامه پس از رسم معماری کامل سیستم آنها را بطور دقیق‌تر توضیح می‌دهیم.

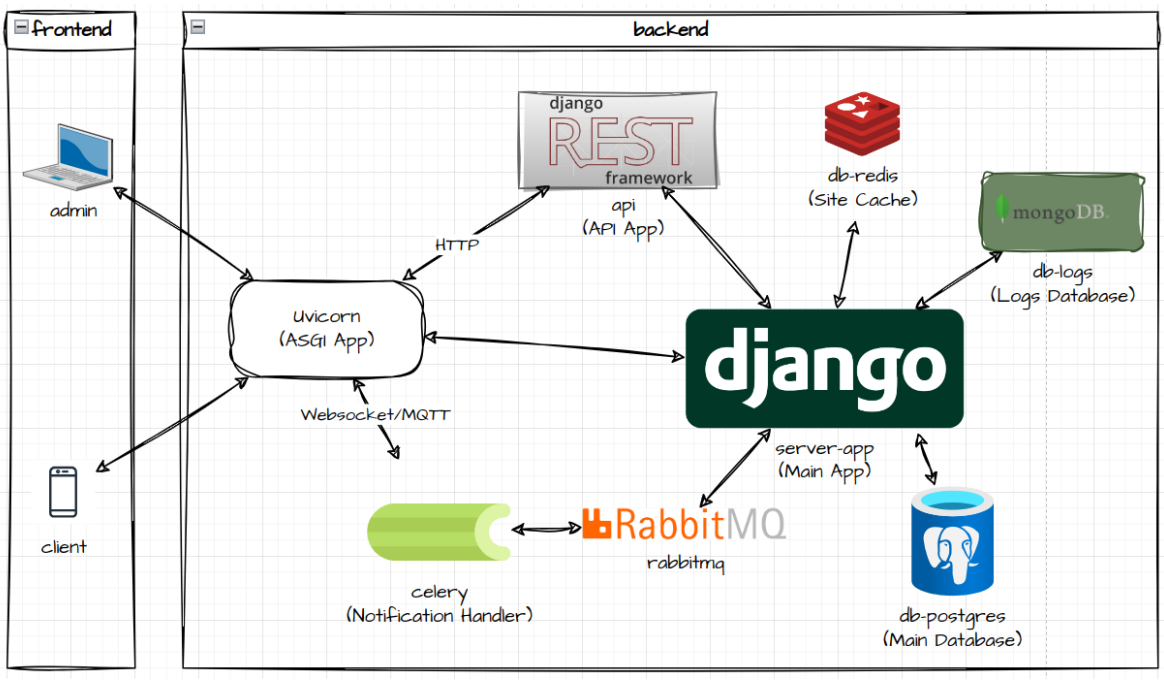
حال می‌توانیم نمودار Component را هم رسم کنیم:



شکل ۸- نمودار Component

طبق نیازمندی‌هایی که انجام دادیم توانستیم بخش‌های اصلی برنامه و فناوری‌های موردنیاز خودمان را به خوبی پیدا کنیم و حال ابتدا معماری پروژه را بطور کلی رسم می‌کنیم و علت انتخاب فناوری‌ها را هم شرح می‌دهیم:

۳-۲-۲ معماری پروژه



شکل ۹ - معماری پروژه

توضیحات اجزای معماری

Django یک چارچوب سطح بالا به زبان پایتون است که توسعه وب‌سایت‌ها و برنامه‌های کاربردی تحت وب را با ارائه یک ساختار قدرتمند و قابل تنظیم ساده‌تر می‌کند.

Django Rest Framework افزونه‌ای برای Django است که قابلیت ایجاد APIهای RESTful را فراهم می‌کند و با استفاده از آن می‌توان سرویس‌های وب مناسب برای اپلیکیشن‌های موبایل و SPAها (Single Page Application) ایجاد کرد.

PostgreSQL یک سیستم مدیریت پایگاه داده رابطه‌ای قدرتمند و بازمتن است که به عنوان پایگاه داده اصلی برای ذخیره‌سازی داده‌های ساختارمند در این معماری به کار می‌رود.

MongoDB یک پایگاه داده NoSQL مبتنی بر اسناد است که برای ذخیره‌سازی لاگ‌ها و داده‌های غیردانشور استفاده می‌شود و امکان جستجوی سریع و انعطاف‌پذیر را فراهم می‌کند.

Django Channels افزونه‌ای برای Django است که امکان پشتیبانی از ارتباطات WebSocket و پروتکل‌های غیر همزمان (asynchronous) را فراهم می‌کند.

Django Admin Dashboard بخشی از Django است که یک رابط کاربری خودکار برای مدیریت داده‌ها و مدل‌های پایگاه داده فراهم می‌کند و معمولاً برای مدیریت داده‌ها و پیکربندی استفاده می‌شود.

Celery یک کتابخانه قدرتمند برای مدیریت و اجرای وظایف زمان‌بندی شده و غیرازامنی (asynchronous tasks) در برنامه‌های پایتونی است.

PyMongo یک کتابخانه پایتون برای اتصال به MongoDB و تعامل با داده‌های موجود در آن به روشی ساده و موثر است.

RabbitMQ یک سیستم صف‌سازی پیام (message broker) است که به عنوان واسطه بین تولیدکنندگان و مصرف‌کنندگان پیام‌ها (وظایف) در Celery عمل می‌کند.

EMQX یک سرور MQTT قوی و پرکاربرد است که پیام‌های MQTT را بین کلاینت‌های مختلف هدایت و مدیریت می‌کند.

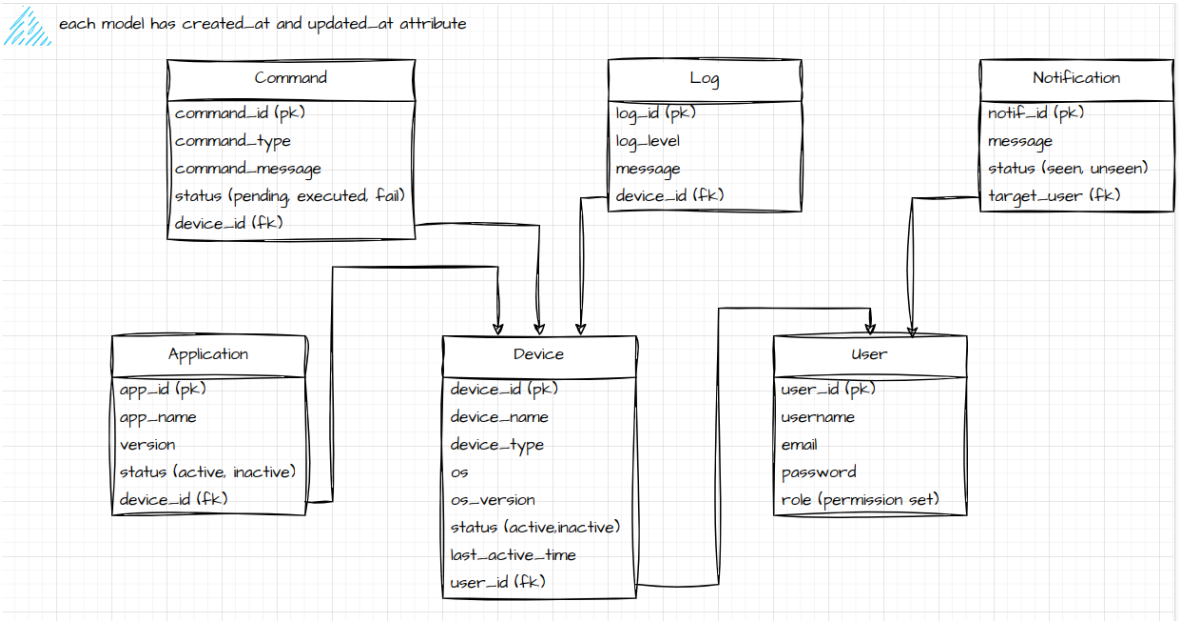
Redis for Site Caching یک پایگاه داده درون حافظه‌ای (In-Memory) است که برای ذخیره‌سازی داده‌های موقت و کشینگ در Django استفاده می‌شود تا عملکرد سایت بهینه‌تر شود.

ارتباط بخش‌ها

در این معماری، Django به عنوان هسته اصلی سیستم عمل می‌کند و از Django Rest Framework برای ایجاد APIها استفاده می‌کند. داده‌های اصلی در PostgreSQL ذخیره می‌شوند، در حالی که لاگ‌ها در MongoDB از طریق PyMongo ذخیره و مدیریت می‌شوند. برای وظایف زمان‌بندی شده و غیرازامن، Celery با استفاده از RabbitMQ به عنوان کارگزار پیام، وظایف را اجرا می‌کند. ارتباطات همزمان و WebSocketها با

Django Channels مدیریت می‌شوند. پیام‌های MQTT با استفاده از paho-mqtt در Django ارسال و دریافت می‌شوند، و EMQX به عنوان کارگزار MQTT برای هدایت این پیام‌ها عمل می‌کند. در نهایت، Redis برای کشینگ داده‌ها و افزایش سرعت پاسخگویی سایت استفاده می‌شود. Django Admin Dashboard نیز به عنوان ابزار مدیریت و نظارت بر داده‌ها و پیکربندی‌ها عمل می‌کند.

۳-۲-۳ مدل پایگاه داده



شکل ۱۰ - مدل پایگاه داده

۳-۳ پیاده‌سازی روش پیشنهادی

برای پیاده سازی مدل پیشنهادی به این شکل عمل می‌کنیم که در گام ابتدایی سعی می‌کنیم زیرساخت پروژه را فراهم کنیم و در ادامه توسعه را روی این زیرساخت ادامه دهیم و مدل‌ها و ارتباطات را به راحتی اضافه کنیم.

از سرور مرکزی شروع می‌کنیم. همانطور که در معماری هم مشخص است از چارچوب جنگو استفاده می‌کنیم. ابتدا باید پایتون را نصب کنیم. پس از نصب پایتون با استفاده از مدیریت بسته‌های نصبی پایتون (pip) شروع به نصب کتابخانه‌های اصلی پروژه می‌کنیم. این کتابخانه‌ها و چارچوب‌ها عبارتند از:

چارچوب جنگو

این بسته[۱۷] هسته مرکزی برنامه ماست. در چارچوب جنگو یک قسمت مرکزی داریم که با نصب بسته چارچوب جنگو اضافه می‌شود. سپس باید قسمت‌های مختلف مورد نیاز را در قالب برنامه‌هایی که توسط این هسته مدیریت می‌شوند به برنامه اضافه کنیم.

مدل معماری کلی جنگو هم به این گونه است که هر برنامه یک بخش داده (مدل) داریم که با استفاده از view های مختلف می‌توان روی آنها عملیات‌های مختلف شامل خواندن، نوشتن، تغییرات و حذف را با استفاده از method های پروتکل HTTP انجام داد.

این چارچوب شامل یک بخش برای تنظیمات تمام پروژه است، تمام برنامه‌ها و تنظیمات کلی آنها برای نیازمندی‌ها در اینجا تعریف و متصل می‌شوند، پایگاه‌های داده مختلف اینجا تعریف و متصل می‌شوند، تنظیمات امنیتی مهم در این فایل ایجاد می‌شود، میان‌افزارها در این قسمت اضافه می‌شوند و ارتباطات کلی پروژه هم در همین فایل مدیریت می‌شود.

سپس بخش آدرس‌های پروژه را داریم که آدرس دسترسی‌ها را تعیین می‌کنیم.

اضافه کردن برنامه رابط کاربری برنامه‌نویسی

اولین برنامه را برای رابط‌های پروژه می‌سازیم و سپس بسته بعدی را نصب می‌کنیم که برای تعریف رابط‌های پروژه برای کاربران است. این بسته بصورت خودکار برای در آدرس‌های معین‌شده رابط می‌سازد که کاربر با استفاده از پروتکل HTTP می‌تواند به این آدرس ها دسترسی پیدا کند.[۳]

مهم‌ترین قسمت هر برنامه موجودیت‌های آن است که در یک بخش مخصوص آنها به همراه ارتباط آنها که در مدل پایگاه داده در شکل آورده شده بود پیاده‌سازی می‌شوند.

اضافه کردن کتابخانه مدیریت احراز هویت

---

[۱۷] Package

کتابخانه djoser را که مخصوص مدیریت کردن احراز هویت است اضافه می‌کنیم. این کتابخانه برای ما آدرس‌هایی را بصورت خودکار می‌سازد که با روش‌های پروتکل HTTP می‌توان به آنها درخواست داد.

ساخت بستر اضافه کردن پروتکل‌های دیگر به جز HTTP

یکی از مهم‌ترین ویژگی‌های این پروژه این است که می‌خواهیم پروتکل‌های دیگر علاوه بر HTTP مثل Websocket یا MQTT را داشته باشیم. این قابلیت با استفاده از یک کتابخانه پایتون که به خوبی با چارچوب جنگو سازگار است ممکن می‌شود. این کتابخانه که uvicorn نام دارد قابلیت ASGI یا Asynchronous Server Gateway Interface را برای جنگو فراهم می‌کند که به این معناست که برنامه جنگو ما می‌تواند از این به بعد از پروتکل‌های غیرهمزمان نیز پشتیبانی کند. برای داشتن چندین پروتکل در بخش ASGI که ورودی تمام درخواست‌ها به سرور است، باید یک آدرس‌دهی با توجه به پروتکل درخواست و آدرس‌های متناظر که به برنامه‌های متناظر برمی‌گردند را تعریف کنیم.

asgi.py 1 ×

server-app > config > asgi.py > ...

```python
import os

from channels.routing import ProtocolTypeRouter, URLRouter

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings')

http_application = get_asgi_application()

from django_channels_jwt_auth_middleware.auth import JWTAuthMiddlewareStack

from . import urls

application = ProtocolTypeRouter(
    {
        "http": get_asgi_application(),
        "websocket": JWTAuthMiddlewareStack(
            URLRouter(urls.websocket_urlpatterns)
        )
    }
)

```

شکل ۱۱ - آدرس‌دهی پروتکل‌های مختلف

ساخت برنامه مدیریت اعلان‌ها

تاکنون یک برنامه داریم که پروتکل HTTP بر روی سرور را به خوبی روی مدل‌ها مدیریت می‌کند. حال باید برنامه‌ی جدیدی برای مدیریت اعلان‌ها اضافه کنیم. این برنامه را می‌سازیم و مدل اعلان را به آن اضافه می‌کنیم.

یک view برای تست اعلان بر روی بستر وب می‌سازیم و سپس شروع به پیاده‌سازی پروتکل Websocket می‌کنیم.

پیاده‌سازی پروتکل Websocket

برای پیاده کردن پروتکل Websocket باید کتابخانه channel از پایتون را نصب کنیم و سپس می‌توان کانال‌هایی بسازیم که قرار است همان ارتباط مستقیم برنامه با کلاینت‌ها را برقرار کند. در برنامه مدیریت اعلان یک کلاس مصرف‌کننده برای کانال می‌سازیم که همان عاملی است که باید با کلاینت‌ها با استفاده از ارتباط مستقیم صحبت کند.

مصرف‌کننده شامل تابع اتصال که اولین لحظه ارتباط که به آن دست‌تکانی هم می‌گویند را مدیریت می‌کند و سپس تابع دریافت که پیام‌های ورودی از بعد از اتصال را مدیریت می‌کند و تابع قطع اتصال است که لحظه قطع اتصال را مدیریت می‌کند و یک تابع که ما آن را اضافه می‌کنیم که برای فرستادن پیام دلخواه ما به سمت کلاینت است که این تابع را باید در جایی که می‌خواهیم اعلان ارسال کنیم صدا کنیم.

consumer.py

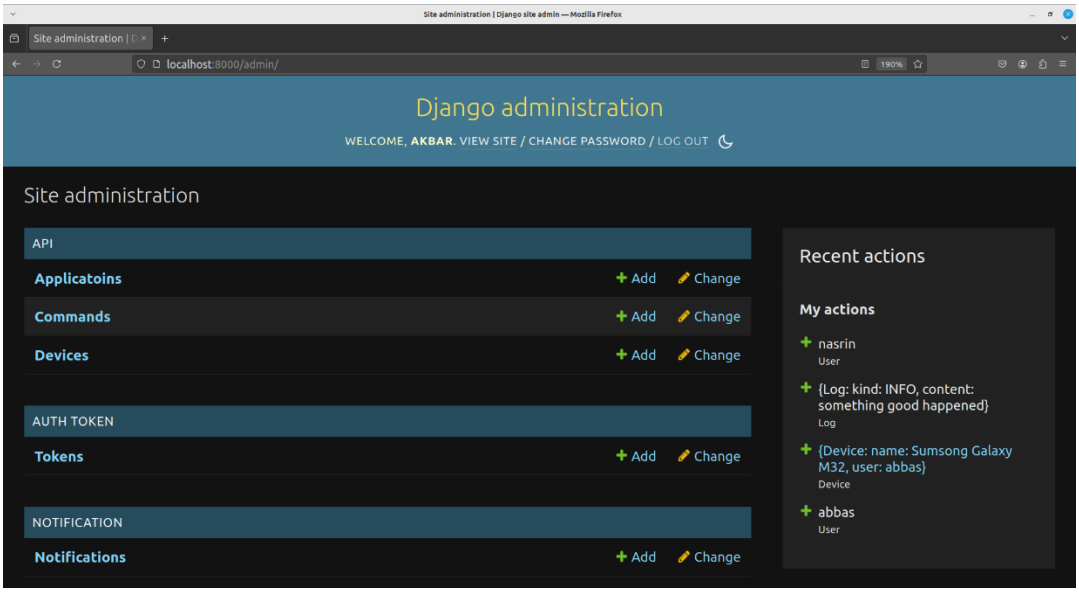
server-app > notification > consumer.py > ...

```python
class NotificationConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        await self.accept()
        print('client connected')

    async def disconnect(self, code):
        await self.channel_layer.group_discard("all", self.channel_name)
        print('client disconnected')

    async def receive(self, text_data, bytes_data=None):
        user = self.scope['user']
        if (user_id := user.id):
            await self.channel_layer.group_add(user.username, self.channel_name)
            await self.send(text_data="Success")
        else:
            await self.send(text_data="Unathorized Access")
            await self.disconnect()

    async def send_notification(self, event):
        message = event["message"]

        template = Template('<div class="notification"><p>{{message}}</p></div>')
        context = Context({"message": message})
        rendered_notification = template.render(context)

        await self.send(
            text_data=json.dumps(
                {
                    "type": "notification",
                    "message": rendered_notification
                }
            )
        )

```

شکل ۱۲ - مصرف‌کننده Websocket

پنل مدیریت جنگو

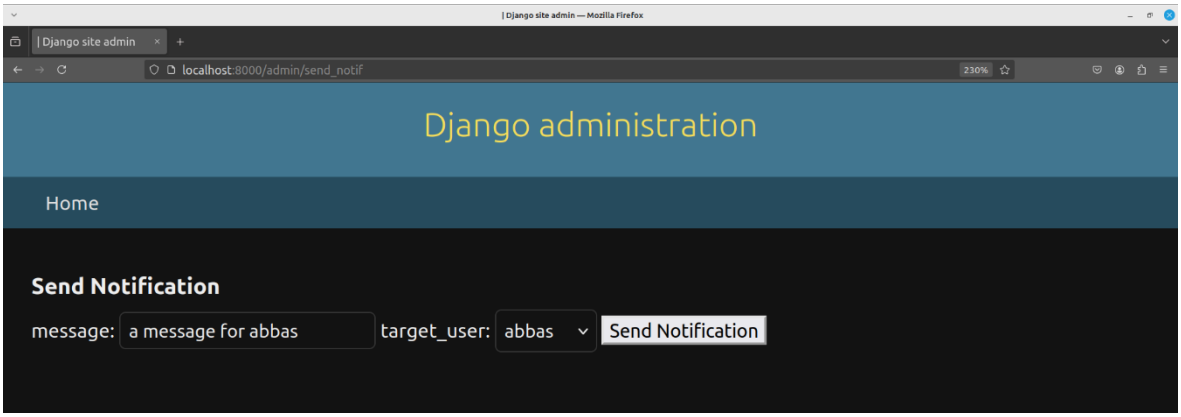
یکی از مهم‌ترین ویژگی‌های چارچوب Django داشتن یک درگاه خوب برای مدیرها است که می‌توانند در آن تمام مدل‌ها را بصورت خودکار مدیریت کنند، داده‌های ورودی را ببینند و ورود و خروج‌ها را بررسی کند.



شکل ۱۳ - درگاه مدیریت Django

پیاده‌سازی درگاه ارسال اعلان‌ها

در درگاه مدیریت جنگو همانطور که گفتیم می‌توان هر کاری با مدل‌ها کرد و مثلا می‌توان یک نمونه اضافه کرد. برای ارسال اعلان‌ها از همین قسمت استفاده می‌کنیم و در قسمت اضافه کردن یک نمونه اعلان جدید ارسال را انجام می‌دهیم برای این کار باید درگاه اضافه کردن اعلان در این قسمت را شخصی‌سازی کنیم که خروجی به شکل زیر است:



شکل ۱۴ - درگاه ارسال اعلان

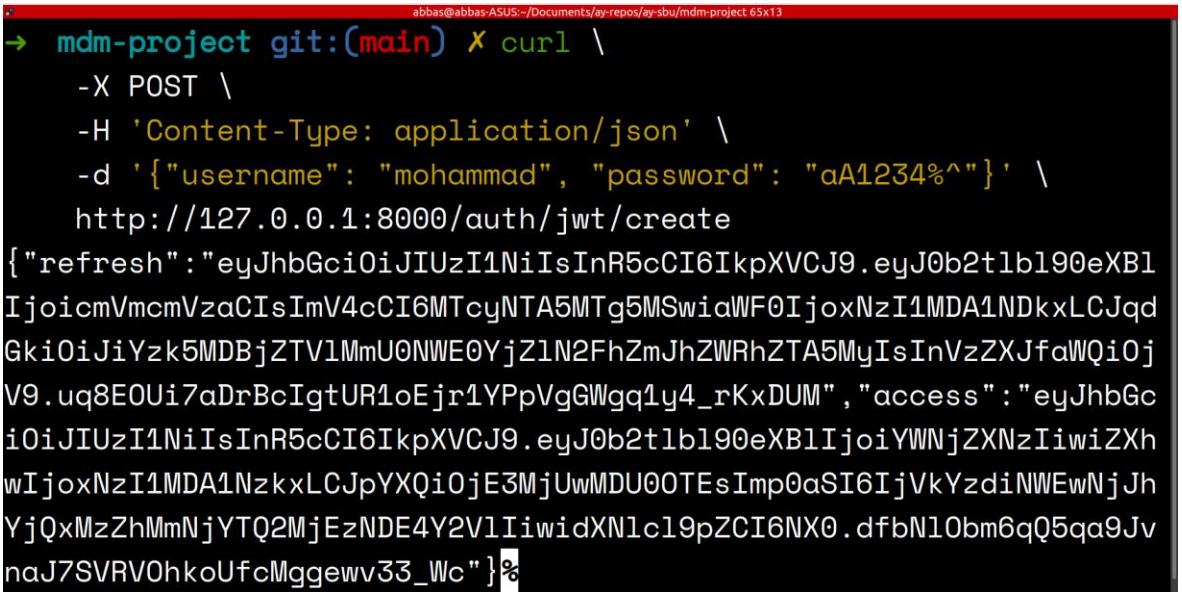
با توجه به فرمی که تعریف کردیم حالا باید تابع ارسال را استفاده کنیم که با کد زیر این کار را می‌کنیم:

```
                channel_layer = get_channel_layer()
                async_to_sync(channel_layer.group_send)(
                    selected_user.first().username,
                    {
                        "type": "send_notification",
                        "message": message
                    }
                )
```

شکل ۱۵ - کد ارسال اعلان

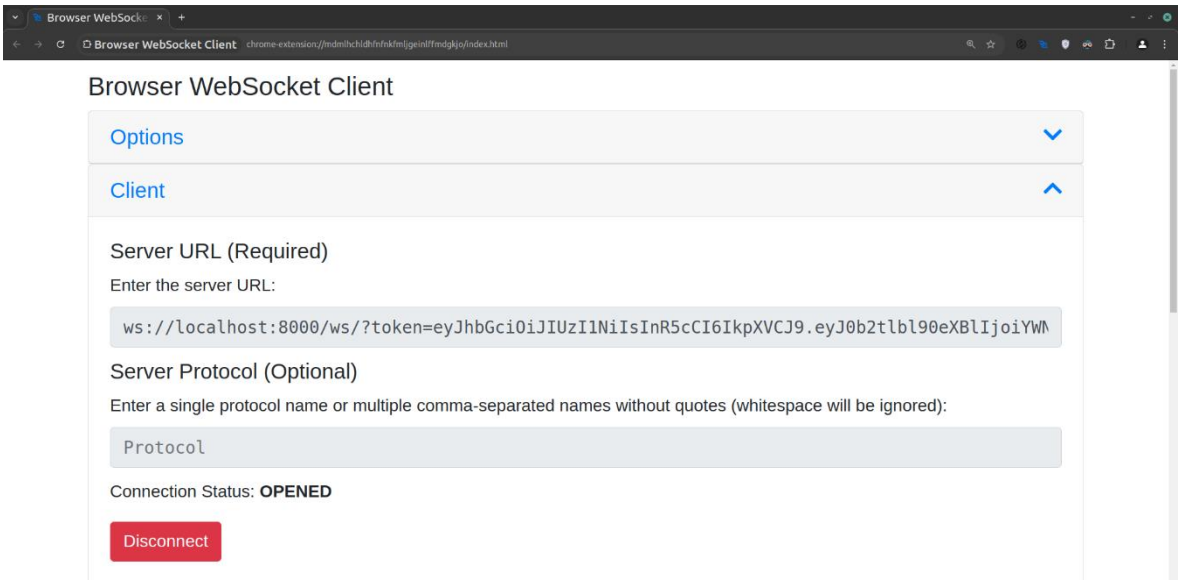
یک مورد کاربردی برای ارسال اعلان

ابتدا برنامه کلاینت باید یک توکن بگیرد:



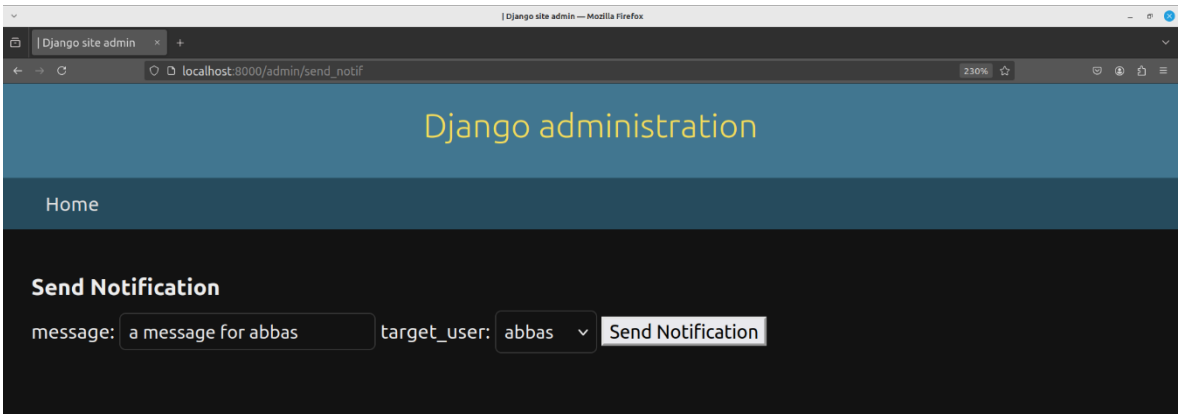
شکل ۱۶ - نحوه گرفتن توکن

سپس باید متصل شود:



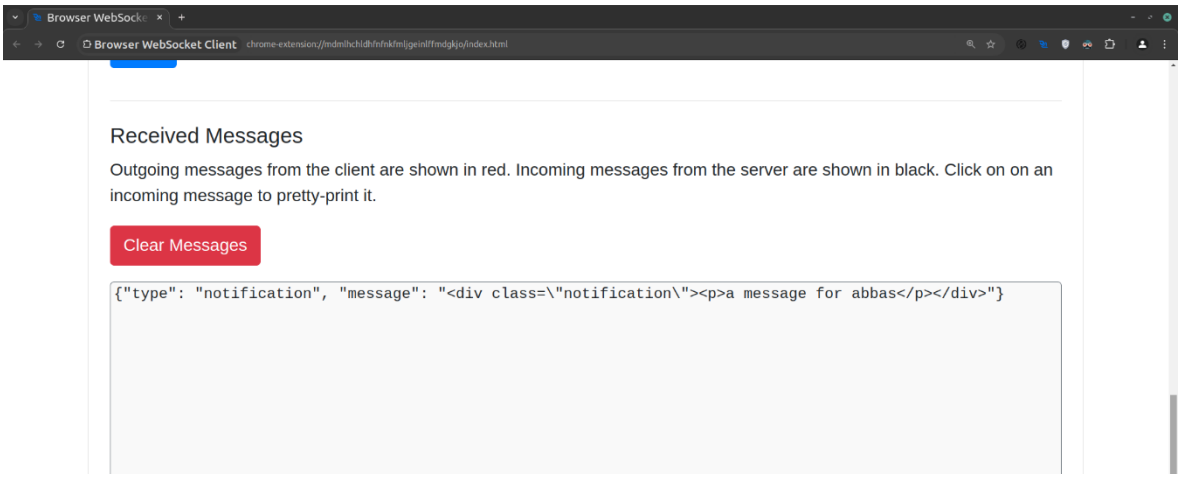
شکل ۱۷ - اتصال کلاینت Websocket

حال مدیر اعلان را از سمت سرور می‌فرستد:



شکل ۱۸ - درگاه ارسال اعلان

برنامه کلاینت (اگر فعال باشد) بصورت بلادرنگ آن را دریافت می‌کند:



شکل ۱۹ - دریافت اعلان توسط کلاینت

اگر برنامه کلاینت فعال نباشد در نوبت بعدی اتصال، پیام‌های دریافت نکرده را بصورت یکجا دریافت می‌کند.

اضافه کردن کتابخانه مدیریت وظایف

پروژه ما به خصوص در بخش ارسال اعلان نیازمند یک عامل مدیریت وظایف است که مطمئن باشیم که وظیفه مورد نیاز انجام شده است. برای این نیازمندی از کتابخانه Celery استفاده می‌کنیم. این کتابخانه یک صف زمانبندی شده است که برنامه ما را بسیار کاربردی و مقیاس‌پذیر می‌کند و باعث می‌شود مدیریت اعلان‌ها را به بهترین وجه انجام شود. [٤]

برای اضافه کردن این برنامه ابتدا تنظیمات اساسی را در بخش مرکزی برنامه انجام می‌دهیم. سپس باید برای این برنامه وظیفه تعریف کنیم. اولین وظیفه‌ای که تعریف می‌کنیم ارسال اعلان است و قرار است اجرای همان کدهای قبلی را به این برنامه منتقل کنیم. که به صورت زیر این کار را انجام می‌دهیم:

شکل ۲۰ - تعریف وظیفه برای Celery

حال کدهای قبلی را تغییر می‌دهیم:

```
34          notification = Notification.objects.create(message=message, target_user=selected_user.first())
35
36          # DEBUG: NOT USING CELERY
37          # channel_layer = get_channel_layer()
38          # async_to_sync(channel_layer.group_send)(
39          #     selected_user.first().username,
40          #     {
41          #         "type": "send_notification",
42          #         "message": message
43          #     }
44          # )
45
46          # USING CELERY
47          send_notification_task.delay(message, selected_user.first().username)
48
49          return HttpResponseRedirect("../{}/".format(notification.pk))
```

شکل ۲۱ - ارسال اعلان با Celery

مجازی‌سازی برنامه با استفاده از Docker

حال زیرساخت پروژه ما در حال کامل شدن است و باید آخرین گام‌ها را برداریم یکی از این گام‌ها مجازی‌سازی برنامه است که با Docker این کار را انجام می‌دهیم. با این کار می‌خواهیم در ادامه هر خدمت جدیدی که خواستیم را به راحتی به برنامه اضافه کنیم و فرآیند ساخت و آزمون برنامه را به راحت ترین و سریع ترین حالت ممکن بر روی هر دستگاهی برسانیم.

برای مجازی‌سازی برنامه اصلی چون که شامل کد است، باید برای آن یک Dockerfile بنویسیم که تمام فرآیند نصب و اجرای برنامه را بصورت خودکار انجام دهد.

اضافه کردن سرویس‌های معماری طراحی‌شده

برای اضافه کردن سرویس‌های جدید و اجرای آن‌ها بصورت یکپارچه نیازمند به نوشتن یک سند docker-compose.yml هستیم که بتوان با استفاده از آن همه‌ی خدمات معماری کلی را به یکباره اجرا و آزمون و خطا کرد و پس از اضافه کردن پیکربندی هر خدمت به سند docker-compose سپس باید پیکربندی‌های مناسب در برنامه مرکزی را نیز در سند تنظیمات انجام دهیم.

اضافه کردن پایگاه داده اصلی

برای پایگاه داده اصلی که بهتر است رابطه‌ای باشد از PostgreSQL استفاده می‌کنیم چون عملکرد خوبی دارد و در ضمن با برنامه ما سازگار است.

اضافه کردن پایگاه داده لاگ‌ها

برای ذخیره‌سازی لاگ‌ها چون حجم بالایی دارد و در ضمن ممکن است شمای آن تغییر کند، باید از یک پایگاه داده غیررابطه‌ای استفاده کنیم که از MongoDB استفاده می‌کنیم.

اضافه کردن پایگاه داده برای Caching

برخی اطلاعات و آدرس‌ها بارها درخواست داده می‌شوند این گونه درخواست‌ها را با استفاده از پایگاه داده Redis که در حافظه اصلی داده‌ها را ذخیره می‌کند که بسیار سریع است، استفاده می‌کنیم.

اضافه کردن درگاه برای هر کدام از خدمات

هر کدام از خدماتی که در سند docker-compose اضافه کردیم می‌توانند درگاه جدا داشته باشند و بصری‌سازی شوند که این کار را نیز انجام می‌دهیم. مثلا پایگاه داده PostgreSQL درگاه PGAdmin را دارد یا عامل Celery درگاه Flower را دارد که نظارت بر مدیریت وظایف را بسیار راحت می‌کند.

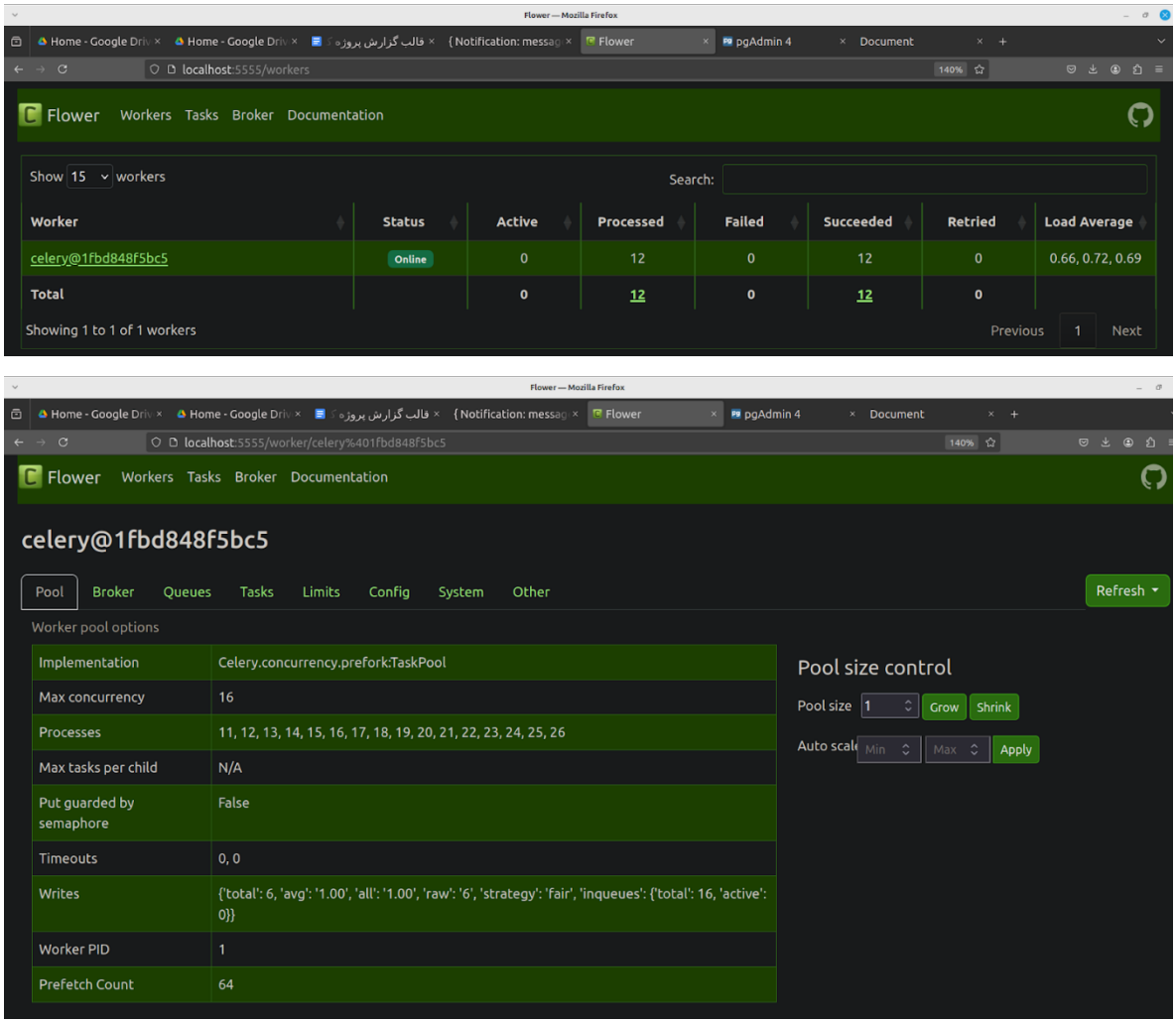
اضافه کردن ساختار صف برای ارتباط میان خدمات

برای اینکه خدمات بتوانند با یکدیگر ارتباط برقرار کنند یک راه مناسب استفاده از صف است که وقتی یک خدمت در حال انجام است پیام در صف بماند تا سرور وقت پاسخ‌دهی داشته باشد. برای صف هم از rabbitmq استفاده کرده‌ایم که یکی از کاربردهایش این است که وظایف را از سرور اصلی به Celery منتقل می‌کند.

۳-٤ نتایج

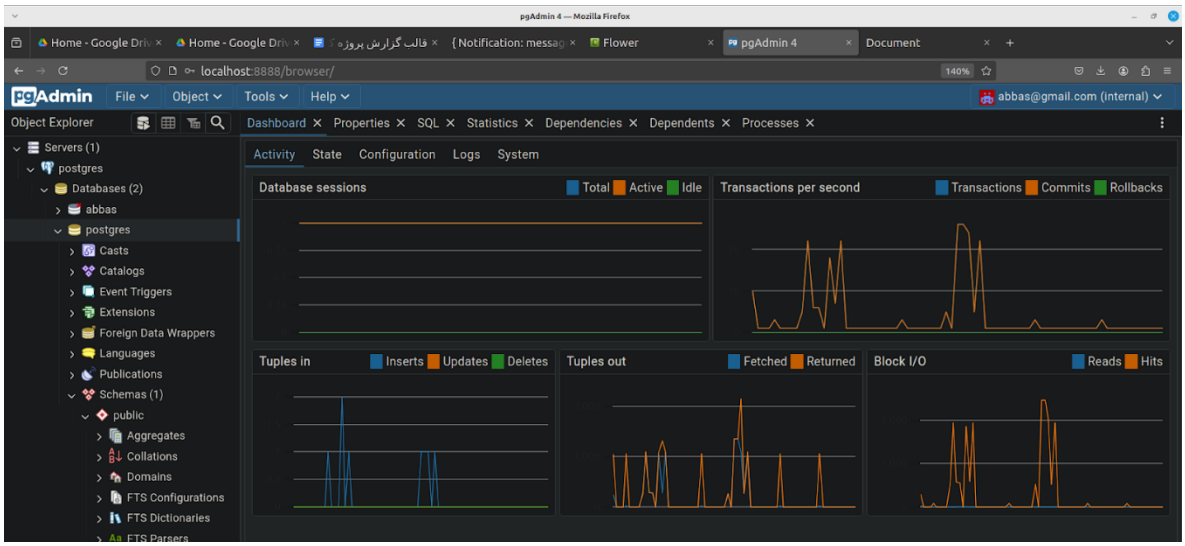
همانطور که گفتیم هر کدام از خدمات ما دارای یک درگاه جداگانه برای بصری‌سازی هستند. که مثلا در یک سناریو که شامل ارسال ۶ اعلان از سمت دو مدیر به دو کاربر به صورت جداست می‌توان نتایج زیر را مشاهده کرد.

نتایج انجام وظایف Celery



شکل ۲۲ - نتایج Celery

نتایج خروجی Postgres



شکل ۲۳ - نتایج Postgres

۳-۵ جمع‌بندی

این پروژه به پیاده‌سازی مدیریت دستگاه‌های موبایلی از طریق سرویس ابری پرداخته است و هدف اصلی آن ایجاد سیستمی یکپارچه و انعطاف‌پذیر برای نظارت، مدیریت، و کنترل دستگاه‌های موبایلی در محیط‌های سازمانی بود. موارد مهمی در این پروژه به آن دست یافتیم:

۱. **احراز هویت کاربر** :این پروژه با استفاده از روش‌های جدید برای احراز هویت کاربران، امنیت دسترسی به سیستم را بهبود بخشیده است.

۲. **نظارت و مدیریت متمرکز:** یکی از چالش‌های مهم این پروژه، توسعه یک سامانه متمرکز و ساده با قابلیت توسعه‌پذیری بالا است که بتواند با وجود تعداد بالای کاربران، نظارت و مدیریت موثری را فراهم کند.

۳. **سادگی در توسعه و استفاده:** در این پروژه سعی شده تا علاوه بر انعطاف‌پذیری، فرآیندهای مدیریتی به سادگی انجام شوند و حتی برخی از آنها به‌صورت خودکار صورت گیرند.

٤. **بررسی و بهبود نمونه‌های مشابه:** این پروژه با بررسی پروژه‌های مشابه و شناسایی نقاط ضعف و قوت آنها، سعی در ارائه راهکاری بهینه‌تر داشته است.

با این رویکردها، پروژه توانسته است به یک سامانه کارآمد و قابل‌اعتماد برای مدیریت دستگاه‌های موبایلی دست یابد که نه تنها امنیت بالایی دارد، بلکه توسعه و استفاده از آن نیز بسیار ساده است.

همانطور که مشاهده کردیم ما در این پروژه سعی کردیم که یک ساختار خوب و کاربردی برای زیرساخت توسعه مدیریت دستگاه‌های موبایلی ارائه کنیم و آن را پیاده‌سازی کردیم. هرچند که این پروژه تا اینجا صرفا زیرساخت و خام است و برای استفاده باید جزییات بیشتری با توجه به مهندسی نیازمندی بیشتر در سمت کلاینت و بررسی انجام‌پذیری ویژگی‌ها به آن اضافه شود. برای مثال در ادامه حتما می‌توان همزمان با داشتن پروتکل Websocket، پروتکل‌های دیگری مثل MQTT را اضافه کنیم.

منابع

[1] A. S. Jat and T.-M. Grønli, "Harnessing the Digital Revolution: A Comprehensive Review of mHealth Applications for Remote Monitoring in Transforming Healthcare Delivery," Lecture Notes in Computer Science. Springer Nature Switzerland, pp. 55–67, 2023. doi: 10.1007/978-3-031-39764-6_4.

[2] Flyve MDM plugin documentation (no date) *Flyve MDM plugin Documentation - Flyve MDM documentation*. Available at: https://flyvemdm-doc.readthedocs.io/en/latest/ (Accessed: ٢٤ August ٢٠٢٤).

[3] Christie, T. (no date) Django rest framework, Home - Django REST framework. Available at: https://www.django-rest-framework.org/ (Accessed: 24 August 2024).

[4] Distributed task queue (no date) Celery. Available at: https://docs.celeryq.dev/en/stable/ (Accessed: 24 August 2024).

پیوست‌ها

لینک گیتهاب پروژه

https://github.com/ay-sbu/mdm-project

Implementation of Cloud Service for Mobile Device Management

Abstract

In the present project, the main objective is to implement the server-side cloud service component within the Mobile Device Management (MDM) cycle. This implementation is designed for monitoring, managing, and controlling mobile clients and includes secure access, centralized management of applications, and user access permissions. Key features of this project include user authentication via cloud services, access monitoring, and centralized configuration and permission settings for clients. The challenge lies in creating an integrated, simple system with high flexibility and scalability while supporting a large number of users, which we aim to address in this project. Similar projects often lack integration or do not offer high flexibility, and in this implementation, we strive to achieve ease of development and simplify management tasks, ensuring that all possible processes are automated and, in some cases, scheduled. Through the proposed software architecture in this project, we aim to solve these issues and meet these needs. This implementation will enable improved security and efficiency in management environments, with a detailed examination of this process and its significance in organizational settings.

Keywords: **Mobile, Operating System, API, Cloud, Backend, Device Management**

Shahid Beheshti University

Faculty of Computer Science and Engineering

Implementation of Cloud Service for Mobile Device Management

By:

Abbas Yazdanmehr

A THESIS SUBMITTED

FOR THE DEGREE OF

BACHELOR OF SCIENCE

Supervisor

Dr. Mehran Alidoostnia

August ٢٠٢٤