دانشکده مهندسی کامپیوتر

پیاده‌سازی درگاه ارتباط با رابط‌های برنامه‌نویسی

پایان‌نامه برای دریافت درجه‌ی کارشناسی در رشته‌ی مهندسی کامپیوتر

میلاد ابراهیمی

استاد راهنما:

دکتر وصال حکمی

شهریور ۱۳۹۹

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه

نام دانشکده: مهندسی کامپیوتر

نام دانشجو: میلاد ابراهیمی

عنوان: پیاده‌سازی درگاه ارتباط با رابط‌های برنامه‌نویسی

تاریخ دفاع: شهریور ۱۳۹۹

رشته: مهندسی کامپیوتر

ردیف	سمت	نام و نام خانوادگی	مرتبه دانشگاهی	دانشگاه یا مؤسسه	امـضـــــا
۱	استاد راهنما	دکتر وصال حکمی	استادیار	دانشگاه علم‌وصنعت ایران	
۲	داور داخلی	دکتر وصال حکمی	استادیار	دانشگاه علم‌وصنعت ایران	

تأییدیه‌ی صحت و اصالت نتایج

باسمه تعالی

اینجانب میلاد ابراهیمی به شماره دانشجویی ۹۵۵۲۱۰۰۹ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری‌شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی‌صلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: میلاد ابراهیمی

تاریخ و امضا:

مجوز بهره‌برداری از پایان‌نامه

بهره‌برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

بهره‌برداری از این پایان‌نامه برای همگان بلامانع است.

بهره‌برداری از این پایان‌نامه با اخذ مجوز از استاد راهنما، بلامانع است.

بهره‌برداری از این پایان‌نامه تا تاریخ ..... ممنوع است.

نام استاد راهنما: دکتر وصال حکمی

تاریخ:

امضا:

چکیده

گاهی‌اوقات می‌خواهیم داده‌هایی را(چه به صورت متنی و چه به صورت تصویری) مخفیانه از جایی به جایی دیگر ارسال کنیم، به گونه‌ای که هیچ‌یک از درگاه‌هایی که در مسیر فرستادن داده‌ها قرار دارند، از «وجود» چنین داده‌های سری خبردار نشوند. در این مقاله شما با یکی از روش‌های انجام این کار، یعنی پنهان‌سازی داده‌های متنی و تصویری آشنا خواهید شد. تمرکز اصلی این مقاله، بر روی پنهان‌نگاری در «تصویر» است.

فهرست مطالب

فهرست تصاویر

فهرست جداول

۱ مقدمه

۱.۱ تاریخچه

گسترش روزافزون راه‌های ارتباطی از طریق شبکه و اینترنت، باعث ایجاد کسب‌وکارهای بسیاری شده است. این فعالیت‌ها در حوزه‌های مختلفی از جمله درخواست خدمات حضوری، ارائه‌ی خدمات مجازی، شبکه‌های اجتماعی و ارتباطی و ... انجام می‌شوند.

با توجه به محبوبیت روزافزون خدمات مجازی در بین مردم، کسب‌وکارها نیز بزرگ و بزرگ‌تر می‌شوند. از این رو، ارائه‌ی خدمات مطمئن، سریع و با کیفیت مطلوب بخشی از اولویت‌های صاحبان کسب‌وکارهاست.

در ابتدای گسترش فناوری‌های مربوط به خدمات مجازی، سامانه‌های یکپارچه و متمرکز، به راحتی جواب‌گوی درخواست‌های مشتریان بودند. ولی با توجه به رشد بسیار سریع تعداد مشتریان، راه‌حل‌های یکپارچه و متمرکز، کارایی خود را از دست داند.

از طرفی، با توجه به پیچیده‌شدن نرم‌افزارهای یکپارچه در طول زمان، گسترش گروه‌های مهندسی جهت ارائه‌ی خدمات بیش‌تر و ایجاد تغییرات در نرم‌افزار، سخت و سخت‌تر می‌شود.

لذا اولین اقدام جهت تاب‌آوری میزان بار و درخواست مشتریان، سعی در بهینه‌سازی نرم‌افزارها و سخت‌افزارها در تمام سطوح ممکن است. این بهینه‌سازی‌ها اغلب بسیار پیچیده‌اند. به طوری که هزینه‌ی نیروی انسانی جهت این فعالیت‌ها، اغلب بسیار بیش‌تر از سود کسب‌شده در قبال آن‌هاست.

۲.۱ معماری‌های جدید توسعه‌ی نرم‌افزار

مطرح‌شدن معماری‌های جدید توسعه نرم‌افزار، از جمله معماری مبتنی بر خدمت، معماری میکروسرویس و ... سعی در برطرف کردن این مشکلات دارند. مهم‌ترین ویژگی این معماری‌ها، توزیع‌پذیری، گسترش‌پذیری و غیر متمرکز بودن است.

هر کدام از معماری‌های مطرح شده، دارای مزایا و معایب و همچنین آسانی‌ها و سختی‌های مختص به خود هستند.

امروزه، معماری میکروسرویس در صنعت بسیار مورد استفاده قرار می‌گیرد. به طوری که بسیاری از کسب‌وکارهای بزرگ داخلی و خارجی از این معماری جهت توسعه محصول استفاده می‌کنند. همان‌طور که گفته شد این معماری دارای مزایا و معایب مختلفی است.

بررسی دقیق این معماری در قالب این گزارش نمی‌گنجد ولی جهت شفافیت هرچه بیش‌تر موضوع، در ادامه به برخی از معایب

این معماری می‌پردازیم.

۳.۱ معایب معماری میکروسرویس

۱.۳.۱ افزایش هزینه‌های توسعه و نگه‌داری

با توجه به پیچیدگی‌های این معماری نسبت به سامانه‌های یکپارچه، زمان و هزینه جهت تولید نرم‌افزارهایی که طبق این معماری ساخته شده باشند افزایش می‌یابد.

از طرفی، به دلیل ماهیت غیر متمرکز این معماری، نگه‌داری و تعمیرات نرم‌افزارهای مبتنی بر این معماری پرچالش‌تر و گران‌تر خواهد بود.

۲.۳.۱ بستر غیر قابل اطمینان شبکه

اجزای مختلف نرم‌افزارهای پیاده‌سازی شده بر اساس معماری میکروسرویس، نیازمند ایجاد ارتباط از انواع مختلف بین یکدیگر هستند. شبکه‌های کامپیوتری یکی از محبوب‌ترین و پراستفاده‌ترین بسترهای ارتباطی برای این منظور است. ولی با توجه به ماهیت این شبکه‌ها، نمی‌توان اطمینان کامل از برقراری ارتباط بین قسمت‌های مختلف نرم‌افزار حاصل کرد. به همین دلیل پیچیدگی‌هایی، جهت ایجاد اطمینان در عملکرد نرم‌افزارها، در پیاده‌سازی نرم‌افزار ایجاد می‌شود.

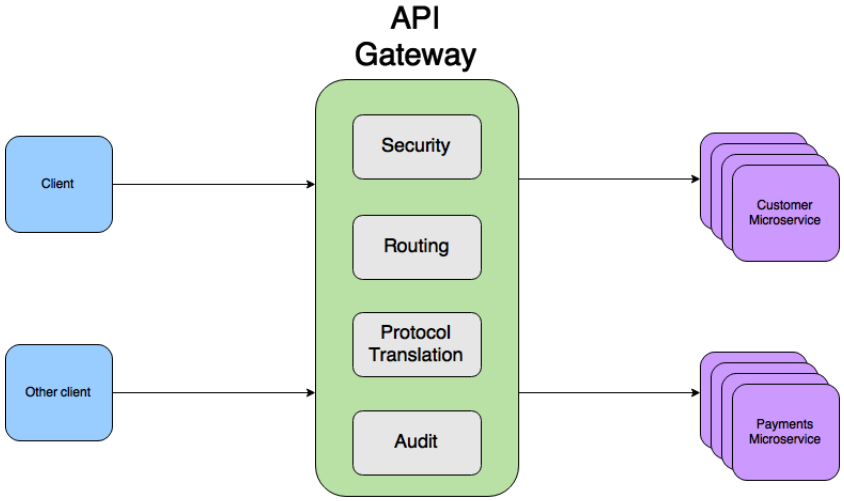
۳.۳.۱ سختی ارتباط کاربران با خدمات ارائه‌شده

در سامانه‌های یکپارچه، ارتباط کاربران و کسب‌وکار، از طریق یک راه ارتباط و تنها دو دستگاه برای کاربر و صاحب کسب‌وکار صورت می‌گرفت. ولی با توجه به ماهیت غیر متمرکز معماری میکروسرویس، کاربران باید با استفاده از چند راه ارتباطی، خدمات مورد نیاز خود را دریافت کنند. از طرفی به‌هنگام‌سازی دستگاه کاربر جهت ارتباط از طریق چند راه ارتباطی هزینه‌های هنگفتی را در بردارد. در این گزارش قصد بر آن است که یک راه حل برای این مشکل یافت‌شود و پس از پیاده‌سازی، نتایج آن بررسی شود.

۴.۱ الگوی درگاه ارتباط با رابط‌های برنامه‌نویسی

این الگو یک الگوی مطرح برای حل مشکل ذکر شده است. فسلفه‌ی این الگو بر این مبنا است که کاربران، مطابق گذشته و معماری نرم‌افزارهای یکپارچه، تنها از طریق یک راه ارتباطی با یک سامانه‌ی میانی ارتباط برقرار کنند. وظیفه‌ی این سامانه‌ی میانی دریافت اطلاعات از بخش‌های مختلف سامانه، به وسیله‌ی رابط‌های برنامه‌نویسی فراهم‌شده توسط هر قسمت، و منتقل کردن آن به کاربر است. شمای کلی این الگو مانند شکل ۱ است.

شکل ۱: شمای کلی الگوی درگاه ارتباط با رابط‌های برنامه‌نویسی



۲ مروری بر منابع

۱.۲ مقدمه

متن مقدمه

۲.۲ تعاریف، اصول و مبانی نظری

,

۱.۲.۲ تیتر

متن‌ها

۳.۲ مروری بر ادبیات موضوع

متن

۴.۲ نتیجه‌گیری

متن نتیجه گیری

۳ روش پیاده‌سازی

۱.۳ مقدمه

برای پیاده‌سازی یک سامانه، می‌توان دو رویکرد داشت. می‌توان از سامانه‌های از پیش موجود استفاده کرد و با حاصل کردن تغییرات لازم به مقصود رسید. و یا می‌توان با پیاده‌سازی سامانه از ابتدا، نیازهای را حل کرد.

از طرفی با توجه به ویژگی‌های ذکرشده در فصل ۲ ، راه حل انتخابی برای پیاده‌سازی یک درگاه ارتباط با رابط‌های برنامه‌نویسی باید معیارهای زیر را مدنظر خود قرار داده باشد:

- سرعت بالا
- کارایی بالا
- بهینه‌بودن راه‌حل
- گسترش‌پذیری
- تغییرپذیری
- سادگی و قابل فهم بودن

۲.۳ انتخاب روش و فناوری

استفاده از فناوری‌های موجود، مانند Nginx، HAProxy و ... و ایجاد تغییرات در آن‌ها جهت بر طرف کردن نیازهای مطرح شده در فصل ۲ ، می‌تواند یکی از راه‌حل های حل مسئله باشد. ولی با توجه به پایه‌ی پیاده‌سازی این محصولات بر اساس نیازهای گذشته، و همچنین عدم سازگاری این محصولات با محیط‌های ابری، جهت استفاده به عنوان درگاه ارتباط با رابط‌های برنامه‌نویسی، مشکلات زیادی برای حل مسئله وجود خواهد آمد. برخی از این مشکلات عبارتند از:

- اجبار به استفاده از راه‌حل‌های موقتی برای سازگاری این محصولات با فضای موجود
- پیچیدگی بیش از حد به دلیل وجود ویژگی‌های مرتبط با حوزه‌های دیگر

- احتمال عدم سازگاری تغییرات مورد نظر با ویژگی‌های اصلی محصول
- و ...

از این رو پیاده‌سازی یک درگاه، از ابتدا و با توجه به نیازهای جدید، راه حل منطقی تری به نظر می‌رسد.

با توجه به نیاز سامانه به دسترسی‌های سطح پایین، جهت ارتباط با لایه‌ی هفتم شبکه، و همچنین نیاز به سرعت بالا برای کنترل بار، زبان‌های برنامه‌نویسی محدودی مناسب پیاده‌سازی این محصول خواهند بود. برخی از این گزینه‌ها عبارتند از:

- زبان برنامه‌نویسی کامپایلری C/C++
- زبان برنامه‌نویسی کامپایلری Golang
- زبان برنامه‌نویسی مفسری Lua
- زبان برنامه‌نویسی Javascript (بستر Node.js)

معمولا زبان‌های برنامه‌نویسی سطح پایین‌تر، سرعت بالاتری نیز دارند. از این رو زبان‌های C و C++ از بالاترین سرعت برخوردار هستند. با این وجود، کاری‌هایی مانند مدیریت حافظه، مدیریت پردازه‌های هم‌روند و ... به عهده‌ی برنامه‌نویس است. این امر باعث پیچیدگی توسعه خواهد بود. از این رو، زبان برنامه‌نویسی Golang، که یک زبان برنامه‌نوسی سطح پایین محسوب می‌شود، به علت مدیریت حافظه خودکار و استفاده‌ی ساده از پردازه‌ها و ریسمان‌ها برای اجرای فرآیندهای هم‌روند، می‌تواند گزینه‌ی مناسبی جهت پیاده‌سازی باشد.

از زبان‌های مفسری نیز می‌توان برای توسعه‌ی نرم‌افزارهایی که در زمان اجرا نیاز به تغییر در خود دارند، استفاده کرد. ولی نقطه‌ی تاریک استفاده از زبان‌های برنامه‌نویسی مفسری، احتمال بالای رخداد خطاهای زمان اجرا خواهد بود. این ویژگی باعث از دست رفتن ثبات سامانه می‌شود.

نحوه‌ی انجام فرآیندهای هم‌روند نیز در انتخاب فناوری پیاده‌سازی این بخش از سامانه بسیار تاثیرگذار است. زیرا با توجه به نیاز به توان عملیاتی بالا، فناوری‌های تک رشته‌ای و یا تک پردازه‌ای باعث کاهش توان عملیاتی سامانه خواهند شد.

جدول ۱ شامل مقایسه‌ی زبان‌های برنامه‌نویسی ذکر شده با توجه به معیارهای ذکر شده است.

جدول ۱: مقایسه‌ی زبان‌های برنامه‌نویسی جهت پیاده‌سازی درگاه ارتباط با رابط‌های برنامه‌نویسی

زبان برنامه‌نویسی	سرعت	سادگی	بهینگی	توان عملیاتی	مدیریت حافظه	مدیریت فرآیندهای هم‌روند
C/C++	بسیار بالا	پایین	بسیار بالا	بسیار بالا	خیر	بسیار سخت
Golang	بالا	متوسط	بالا	بالا	بله	آسان
Lua	بالا	متوسط	بالا	متوسط	بله	سخت
Javascript	متوسط	بالا	پایین	متوسط	بله	سخت

با توجه به جدول ۱ ، روش انتخابی برای حل مسئله، پیاده‌سازی سامانه از ابتدا و با استفاده از زبان برنامه‌نویسی Golang است. زیرا با اینکه سرعت کمتری نسبت به زبان برنامه‌نویسی C یا C++ دارد، ولی با توجه به عدم نیاز به مدیریت حافظه، مدیریت آسان‌تر فرآیندهای هم‌روند و همچنین سادگی و بهینگی قابل قبول، می‌توان از اختلاف سرعت این دو زبان برنامه‌نویسی چشم پوشی کرد.

۳.۳ پیاده‌سازی

۱.۳.۳ ساختار

ساختار برنامه باید به شکلی تعیین شود که برنامه‌نویس را در روند توسعه برنامه محدود نکند. علاوه بر این، یک ساختار خوب می‌تواند زمینه را جهت معماری گسترش‌پذیر و منعطف فراهم کند.

یکی از ساختارهای مناسب، در محیط توسعه‌ی Golang ، ساختار بسته مبنا است. این ساختار توسط بسیاری از محصولات صنعتی و آکادمیک استفاده شده است. همچنین برخی از ابزارهای از پیش‌آماده در محیط توسعه‌ی این زبان، از سازگاری کامل با این ساختار برخوردار هستند.

۲.۳.۳ ساختار بسته‌مبنا

این ساختار، شامل سه پوشه‌ی اصلی cmd، internal و pkg است. پوشه‌های config و logging از جمله پوشه‌های دیگر این ساختار هستند که استفاده از آن‌ها توصیه شده است.

پوشه‌ی cmd حاوی برنامه‌ی اصلی و قابل اجرا است. این پوشه معمولا دارای برنامه‌های کوتاه و با منطق ساده هستند.

پوشه‌ی pkg حاوی بسته‌های مستقل و قابل استفاده توسط سایر برنامه هاست. بسته‌های موجود در این پوشه به تنهایی قابل استفاده اند و وابستگی آن‌ها به دیگر بسته ها بسیار کم است.

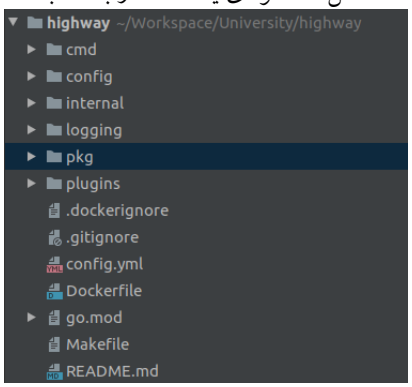
پوشه‌ی internal دارای بسته‌هایی است که از بسته‌های موجود در pkg استفاده کرده و منطق اصلی برنامه را پیاده‌سازی می‌کند. در این پوشه، رابط‌های ورودی و خروجی برنامه تعیین می‌شود.

بسته‌ی logging شامل برنامه‌های مربوط به ثبت اتفاقات سامانه است.

بسته‌ی config نیز دربردارنده‌ی برنامه‌های مربوط به پیکربندی و تنظیمات اجرای برنامه است. نحوه‌ی بارگذاری پیکربندی‌ها و همچنین استفاده از تنظیمات در قسمت‌های مختلف برنامه در این قسمت پیاده‌سازی می‌شود.

نمونه‌ی یک ساختار بسته‌مبنا در شکل ۲ قابل مشاهده است.

شکل ۲: نمونه‌ی یک ساختار بسته مبنا



۴.۳ مفاهیم

برای پیاده‌سازی یک درگاه ارتباط با رابط‌های برنامه‌نویسی، مفاهیم زیر پیشنهاد شده است:

- بطن
- خدمت
- متعادل‌کننده‌ی بار
- ضابطه

- میان‌افزار
- مسیریاب

در ادامه به توضیح هر یک از مفاهیم ذکر شده پرداخته می‌شود.

۱.۴.۳ بطن

یک بطن، کوچک‌ترین واحد در این سامانه است. هر بطن، یک داده ساختار شامل نام، آدرس، وزن و وضعیت است. این چهار ویژگی نشان‌دهنده‌ی یک نمونه از رابط‌های برنامه‌نویسی قابل استفاده توسط برنامه است. داده‌ساختار زیر، نشان‌دهنده‌ی بطن است.

```
type Backend struct {
    Name string
    Addr string
    Weight int8
    Status int
}
```

۲.۴.۳ خدمت

یک خدمت دارای یک نام، چند بطن و یک متعادل‌کننده‌ی توزیع بار است. معمولا بطن‌های یک خدمت کاملا مشابه یکدیگر عمل می‌کنند. با توجه به قسمت ۳.۴.۳ ، خدمت با استفاده از متعادل‌کننده‌ی بار درخواست‌های را بین بطن‌های خود توزیع می‌کند.

قطعه کد زیر نشان‌دهنده‌ی داده‌ساختار یک خدمت است.

```
type Service struct {
    Name     string
    Backends []Backend
    LB       LoadBalancer
}
```

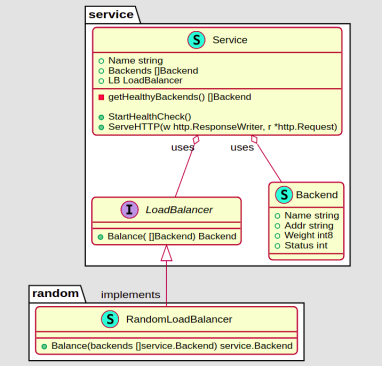
۳.۴.۳ متعادل‌کننده‌ی بار

این موجودیت در برنامه از نوع یک واسط است. این واسط بدون در نظر گرفتن منطق پیاده‌سازی توزیع بار، رفتارهای مورد نیاز برای این عمل را مشخص می‌کند.

این واسط به طور مستقیم قابل استفاده نیستند. بلکه نسخه‌های پیاده‌سازی شده‌ی این واسط را می‌توان به طور مستقیم استفاده کرد. برای مثال، الگوریتم توزیع تصادفی جهت توزیع بار در این برنامه پیاده‌سازی شده است. برنامه‌ی زیر نشان‌دهنده‌ی واسط متعادل‌کننده‌ی بار و شکل ۳ حاوی نمودار پیاده‌سازی نمونه‌ها از واسط هستند.

```
type LoadBalancer interface {
    Balance([]Backend) Backend
}
```

شکل ۳: نمودار UML بسته‌ی Service



متعادل‌کننده‌های بار می‌توانند از وزن‌های بطن‌های یک سرویس، برای توزیع وزن‌دار بار استفاده‌کنند.

۴.۴.۳ ضابطه

ضابطه‌ها داده‌ساختارهایی هستند که می‌توان از طریق آن‌ها به خدمات دسترسی پیدا کرد. در قالب این گزارش، ضابطه‌ها تنها برای پروتکل‌های HTTP و HTTPS طراحی شده‌اند. با توجه به ویژگی‌های این پروتکل‌ها، ضابطه‌های مختلفی می‌توان تعریف کرد. برخی از این ویژگی‌ها عبارتند از:

- نوع پروتکل (HTTP یا HTTPS)
- مسیر درخواست
- آدرس میزبان درخواست
- روش‌های استفاده از پروتکل

- سرتیترهای درخواست
- مولفه‌های پرس‌وجوی درخواست
- میان‌افزارها

کاربرد و نحوه‌ی استفاده از میان‌افزار ها در قسمت ۵.۴.۳ بررسی خواهد شد.

قطعه کد زیر، نمایانگر داده‌ساختار یک ضابطه است.

```
type Rule struct {
    Service     *service.Service
    Schema      string
    PathPrefix  string
    Hosts       []string
    Methods     []string
    Headers     map[string]string
    Queries     map[string]string
    Middlewares []middlewares.Middleware
    handler     http.HandlerFunc
}
```

۵.۴.۳ میان‌افزار

گاهی نیاز است قبل از رسیدن درخواست به خدمات، تغییراتی در درخواست ایجاد شود. و یا برخی از سیاست‌های کنترلی مانند احراز هویت و تایید دسترسی قبل از رسیدن درخواست به خدمت بررسی شود.

در برخی از مواقع نیز این نیاز مطرح است که از نتایج درخواست‌ها مطلع شد و از آن‌ها برای تولید گزارشات و تهیه‌ی داده‌های آماری استفاده کرد.

میان‌افزارها این نیازها را برطرف خواهد کرد. میان‌افزارها نیز به صورت یک واسط تعریف شده‌اند و نسخه‌های پیاده‌سازی شده‌ی مختلف آن‌ها نیازهای مختلف را رفع خواهند کرد. برنامه‌ی زیر نشان‌دهنده‌ی شمای این واسط است.

```
type Middleware interface {
    Process(handler http.HandlerFunc) http.HandlerFunc
}
```

در سامانه‌ی پیاده‌سازی شده، میان‌افزار ها دو قسم اند. برخی از آن‌ها،که معمولا جز میان‌افزارهای پرطرفدار در صنعت هستند، به صورت پیشفرض پیاده‌سازی شده‌اند. میان‌افزارهای از پیش آماده در این سامانه عبارتند از:

- میان‌افزار CORS
- میان‌افزار محدودکننده‌ی نرخ درخواست
- میان‌افزار Monitoring

محدودکننده‌ی نرخ درخواست با استفاده از الگوریتم Token Bucket باعث می‌شود بتوان نرخ ارسال درخواست‌های کاربران را تحت کنترل قرار داد.

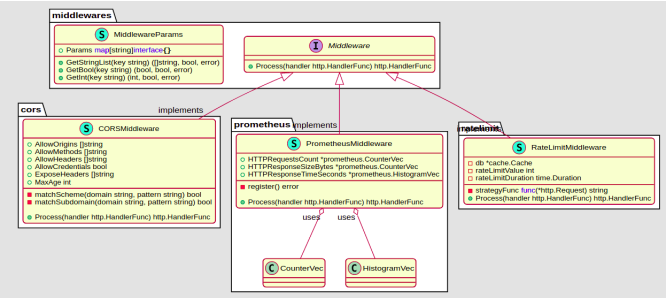
میان‌افزار CORS نیز با استفاده از استانداردهای تعیین‌شده برای درخواست‌های مبدا و مقصد مختلف، دسترس‌پذیری این سامانه را از منابع مختلف تعیین می‌کند.

میان افزار Monitoring نیز، طبق استاندارد نرم‌افزار Prometheus ، امکان بررسی و تحلیل اطلاعات را فراهم می‌کند.

گروه دیگری از میان‌افزارها نیز توسط کاربران استفاده‌کننده از سامانه تعریف و پیاده‌سازی شده و به داخل سامانه تزریق می‌شوند. این ویژگی باعث می‌شود، استفاده کنندگان از این سامانه، هیچ‌گونه وابستگی به توسعه‌دهندگان اصلی برنامه نداشته باشند و میان‌افزارهای مربوط به نیازمندی‌های خاص خود را پیاده‌سازی کرده و در سامانه تزریق کنند.

شکل ۴ نشان‌دهنده‌ی نحوه‌ی پیاده‌سازی میان‌افزارها با استفاده از واسط تعیین‌شده است.

شکل ۴: نمودار UML بسته‌ی Middleware



۶.۴.۳ مسیریاب

یک درگاه ارتباط با رابط‌های برنامه‌نویسی را می‌توان یک مسیریاب لایه‌ی هفتم شبکه در نظر گرفت. بدین صورت که درخواست‌های ارسالی از سمت کاربران را به پاسخ‌دهنده‌های مربوط به آن درخواست‌ها هدایت می‌کند.

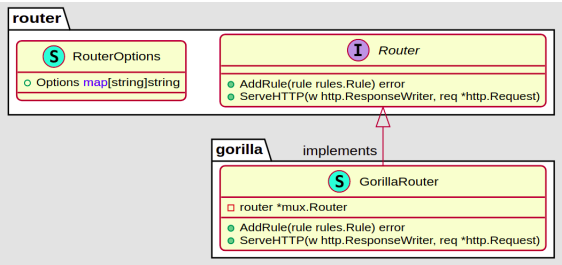
یک مسیریاب به طور واسط تعریف شده‌است و رفتارهای مورد نیاز آن، بدون توجه به منطق پیاده‌سازی آن مشخص شده است. برنامه‌ی زیر تعریف یک واسط مسیریاب را نشان می‌دهد.

```
type Router interface {
    AddRule(rule rules.Rule) error
    ServeHTTP(w http.ResponseWriter, req *http.Request)
}
```

مسیریاب‌ها با استفاده از ضابطه‌های تعریف‌شده، هر درخواست را به خدمت مورد نظر هدایت می‌کنند.

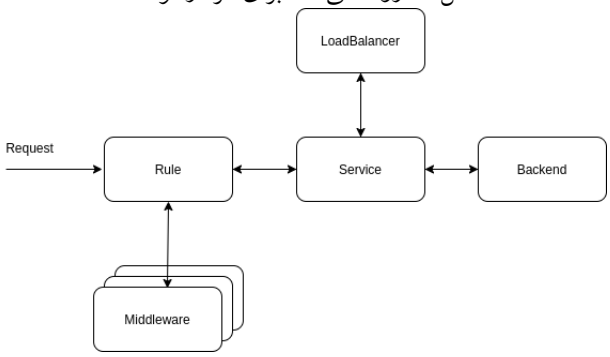
شکل ۵ نشان‌دهنده‌ی نحوه‌ی پیاده‌سازی یک مسیریاب با استفاده‌از واسط تعیین شده است.

شکل ۵: نمودار UML بسته‌ی Router



به طور کلی، شکل ۶ نشان‌دهنده‌ی روند طی‌شده برای درخواست کاربران را به طور خلاصه شرح می‌دهد.

شکل ۶: روند طی‌شده برای هر درخواست



۵.۳ ویژگی‌های سامانه

معماری ذکرشده در بخش ۴.۳ ، ویژگی‌های زیر را داراست:

- کمترین وابستگی ممکن
- آزمون پذیری بالا
- گسترش پذیری بالا
- مقیاس‌مندی بالا

۱.۵.۳ وابستگی

با استفاده درست از واسط‌ها و همچنین طراحی ماژولار، بخش‌های مختلف سامانه کمترین وابستگی به یکدیگر را دارند. این ویژگی باعث می‌شود تغییرات احتمالی در آینده به راحتی هرچه تمام‌تر انجام شود.

۲.۵.۳ آزمون‌پذیری

با توجه به وابستگی کم اجزای سامانه به یکدیگر،هر یک از اجزا می‌توانند به تنهایی مورد آزمون و ارزیابی قرار گیرند. هم‌چنین استفاده از واسط‌ها امکان ایستا کردن یک بخش و انجام آزمون واحد بر روی بخش دیگر را امکان‌پذیر می‌کند.

۳.۵.۳ گسترش‌پذیری

با توجه به استفاده به موقع از واسط‌ها، گسترش سامانه، از طریق پیاده‌سازی نسخه‌های جدید از واسط‌ها، بسیار ساده خواهد بود. همچنین امکان تزریق میان‌افزارهای پیاده‌سازی شده توسط کاربران استفاده کننده، بدون نیاز به کامپایل مجدد نرم‌افزار، خاصیت گسترش پذیری نرم‌افزار را افزایش می‌دهد.

۴.۵.۳ مقیاس‌مندی

با توجه به عدم ذخیره‌ی حالت در سامانه، می‌توان تعداد نسخه‌های در حال اجرای سامانه را به طور خطی برای افزایش میزان توان عملیاتی سامانه، زیاد کرد.

۴ نتایج و تفسیر آن‌ها

پس از پیاده‌سازی معماری شرح‌داده‌شده در فصل ۳ ، آزمایش کیفیت و نحوه‌ی عملکرد سامانه ضروری به نظر می‌رسد. در آزمایش‌های انجام‌شده، سامانه‌ی پیاده‌سازی شده با یکی از محبوب‌ترین نمونه‌های صنعتی، به نام Kong ، مورد مقایسه قرار گرفته شده‌است. در ادامه، به نحوه‌ی انجام آزمایش پرداخته شده است.

۱.۴ محیط آزمایش

با توجه به کاربرد وسیع درگاه‌های ارتباط با رابط‌های برنامه‌نویسی در محیط‌های ابری، انجام این آزمایش در محیط ابری به واقعیت نزدیک تر خواهد بود. از این رو تمام آزمایش‌ها در محیط ابری انجام شده‌است. آزمایش‌ها بر روی بستر OKD انجام شده‌اند.

برای ایجاد بار بر روی سامانه‌های مورد آزمایش از سکوی Locust استفاده شده است. میزان منابع مصرفی این سکو برای ایجاد بار به شرح زیر است:

- میزان حافضه‌ی موقت مصرفی: ۱۱ گیگابایت
- تعداد هسته‌های پردازشی: ۲۲
- تعداد ایجادکننده‌های بار: ۱۰

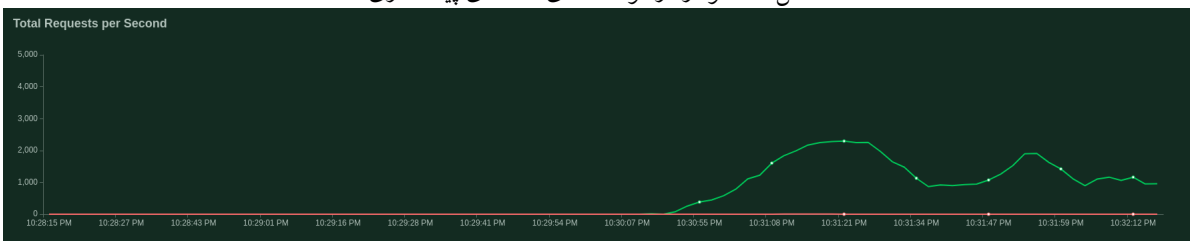
۲.۴ شرح آزمایش‌ها

۱.۲.۴ میزان تحمل بار

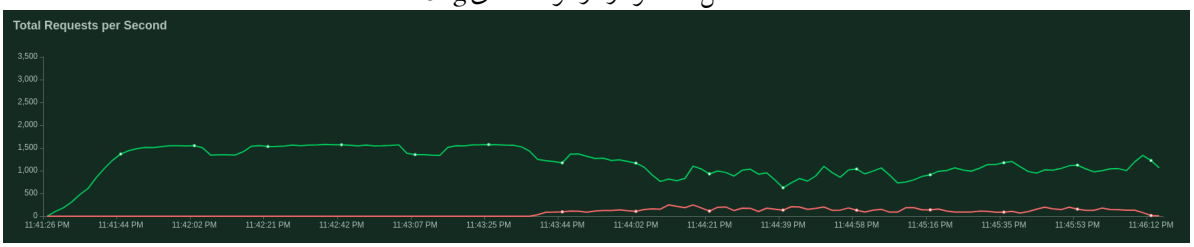
برای بررسی میزان تحمل بار، پس از ساخت دو بطن مشابه و بدون سربار بالا، در قالب یک خدمت و استفاده از یک ضابطه‌ی مناسب به انجام آزمایش مبادرت شده است. این فرآیند در هر دو سامانه‌ی مورد ارزیابی انجام شده است.

نتایج حاصل از تولید بار بر هر یک از سامانه‌ها، که در شکل‌های ۷ و ۸ قابل مشاهده است به شرح زیر است.

شکل ۷: نمودار درخواست‌های سامانه‌ی پیاده‌سازی شده



شکل ۸: نمودار درخواست‌های Kong



جدول ۲: مقایسه‌ی Kong و سامانه‌ی پیاده‌سازی شده

محصول مورد آزمایش	بیشینه مصرف حافظه‌ی موقت	بیشینه مصرف هسته‌های پردازشی	بیشینه میزان تحمل بار
سامانه‌ی پیاده‌سازی شده	۲۴۴ MB	۲	۲۳۳۰ rps
Kong	۲۰۴۸ MB	۲	۱۶۴۰ rps

با توجه به جدول شماره‌ی ۲ ، علی رغم کاهش ٪ ۸۸ ای میزان مصرف حافظه‌ی موقت، بیشینه میزان تحمل بار حدود ٪ ۴۲ افزایش یافته است. علاوه بر این طبق شکل شماره‌ی ۷ ، میزان درخواست‌های رد شده در سامانه‌ی پیاده‌سازی شده صفر است ولی در سامانه‌ی Kong شاهد رد شدن برخی از درخواست‌ها در اوج بار هستیم.

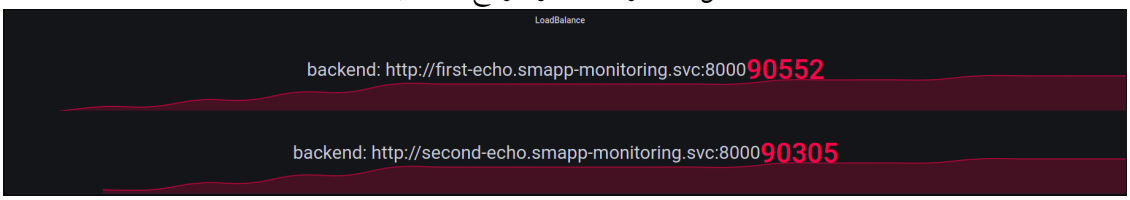
از این نظر، سامانه‌ی پیاده‌سازی شده بر Kong برتری کامل دارد.

۲.۲.۴ نحوه‌ی توزیع بار

با توجه به وجود دو بطن برای سرویس مورد آزمایش، نحوه‌ی عملکرد توزیع‌کننده‌ی بار نیز باید مورد آزمایش قرار گیرد. با فعال کردن میان‌افزار Prometheus بر روی یک ضابطه‌ی جدید و اجرای آزمایش بر روی آن و همچنین تحلیل نتایج حاصل، می‌توان نتایج را

در شکل شماره‌ی ۹ مشاهده کرد.

شکل ۹: نحوه‌ی عملکرد توزیع‌کننده‌ی بار

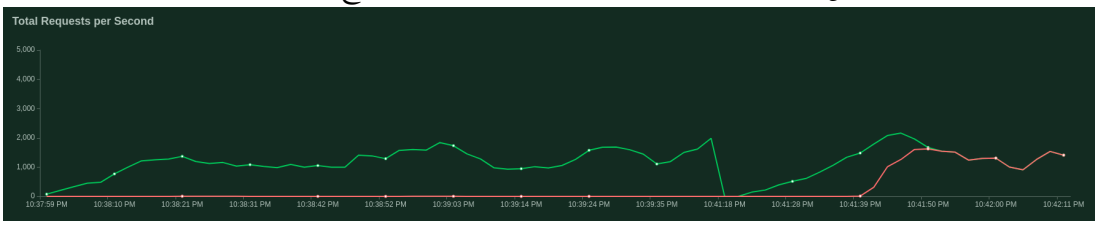


با توجه به شکل شماره‌ی ایکس، از مجموع ۱۸۰۸۷۵ درخواست ارسال شده به سامانه، ۹۰۵۵۲ به بطن اول و ۹۰۳۰۵ به بطن دوم هدایت شده اند. در مجموع ٪ ۵۰/۰۷ درخواست ها به بطن اول و ٪ ۴۹/۹۳ از درخواست‌ها به بطن دوم هدایت شده‌اند. با توجه به وزن یکسان هر دو بطن، نحوه‌ی عملکرد توزیع‌کننده بار، قابل قبول است.

۳.۲.۴ کنترل نرخ درخواست‌ها

یکی از میان‌افزارهای پیاده‌سازی شده، کنترل کننده‌ی نرخ درخواست‌ها است. برای ارزیابی این میان‌افزار و عملکرد درست آن، با ساخت یک ضابطه‌ی جدید و فعال‌سازی این میان‌افزار در آن،به اجرای دوباره‌ی آزمایش مبادرت شده است. نحوه‌ی پاسخ به درخواست‌ها توسط سامانه، در شکل ۱۰ قابل مشاهده است.

شکل ۱۰: نحوه‌ی رفتار سامانه در حضور کنترل کننده‌ی نرخ درخواست‌ها

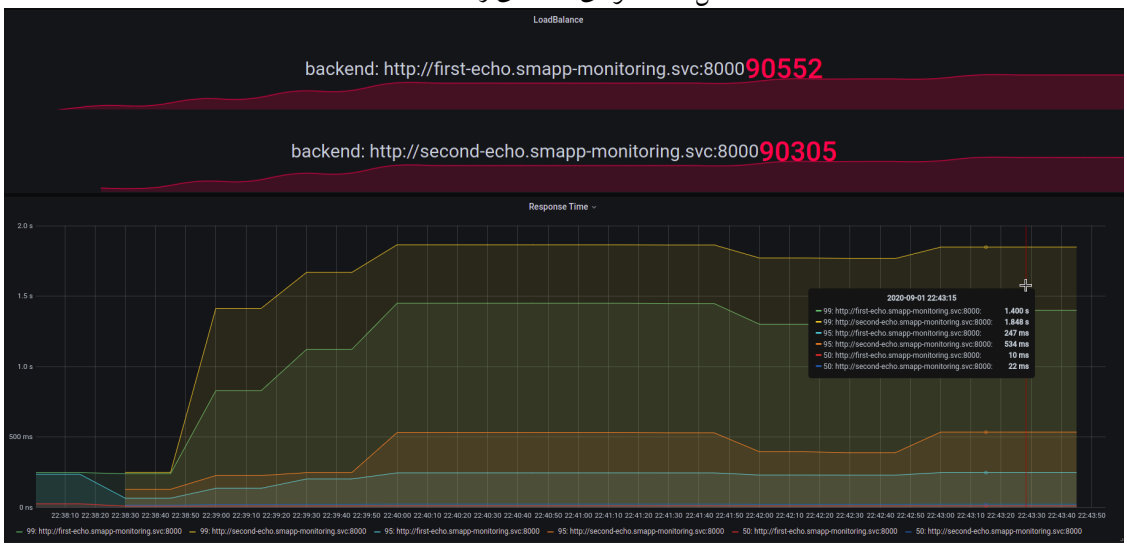


به توجه به شکل، پس از تمام شدن تعداد درخواست مجاز هر فرستنده در زمان ۱۰:۴۱:۳۹، سامانه شروع به رد کردن درخواست‌های جدید می‌کند. از این رو، میان‌افزار پیاده‌سازی شده به درستی عمل می‌کند.

۴.۲.۴ رصد و تحلیل سامانه

برای بررسی عملکرد مطلوب میان‌افزار Prometheus جهت رصد و تحلیل این سامانه، این میان‌افزار در یک ضابطه‌فعال شده و ارزیابی مجددا انجام شده است. در شکل شماره‌ی ۱۱ ، شاهد نمونه‌ی صفحه‌ی رصد سامانه هستیم. در این صفحه نحوه‌ی عملکرد توزیع‌کننده‌ی بار و همچنین Quantile های مختلف مدت زمان پاسخ‌گویی سامانه محاسبه شده است.

شکل ۱۱: نمونه‌ی صفحه‌ی رصد سامانه



۵ جمع‌بندی و پیشنهادها

۱.۵ مقدمه

متن مقدمه

۲.۵ محتوا

متن مقدمه

۱.۲.۵ جمع‌بندی

متن‌ها

۲.۲.۵ نوآوری

متن‌ها

۳.۲.۵ پیشنهادها

متن‌ها

Abstract

English Abstract Here

Computer Engineering Department

Implementing an API (Application Programming Interface) Gateway

Bachelor of Science Thesis in Computer Engineering

Milad Ebrahimi

Supervisor:

Dr. Vesal Hakami

September 2020