تسک فنی مهندسی نرم‌افزار

سامان اسلامی نظری

۱۵ بهمن ۱۴۰۳

ساختار کلی

وبسایت `BitPin-Blog` به طور کلی متشکلی از ۳ برنامه[1] می‌باشد: ۱) `blog_posts`، ۲) `rate` و ۳) `users`. به طور کلی برنامه `blog_posts` شامل مدل‌ها و ویوهای مورد نیاز برای ایجاد و نمایش مطالب کاربران است؛ برنامه `rate` وظایف مربوط به امتیاز دادن و برنامه `users` برای مدیریت کاربران می‌باشد.

برای محدود کردن درخواست‌های ناگهانی و پشت هم، از دو الگوریتم Leaky Bucket و Exponential Moving Average استفاده شده که در بخش‌های آتی توضیح داده شده‌اند.

برنامه blog_posts

ویوهای این برنامه صرفا وظیفه ایجاد و نمایش مطالب کاربران را دارند. مدل‌ها و شرح دقیق‌تر ویوها به صورت زیر می‌باشد.

مدل‌ها

شامل سه مدل `BlogPost`، `BlogRatingLeakyBucket` و `BlogRatingEma` می‌باشد. مدل اول شامل فیلدهای اصلی برای ذخیره اطلاعات مطالب به همراه یک فیلد `average_rating` برای محاسبه میانگین امتیازات کاربران به این مطلب است. دلیل وجود این فیلد این است که هر بار در پی درخواست یک کاربر برای دریافت میانگین امتیاز یک مطلب، میانگین را از اول حساب نکنیم؛ با زیاد شدن تعداد امتیازها محاسبه هر باره میانگین زمان‌بر می‌شود؛ بنابراین برای جلوگیری از این اتفاق، هر بار پس از دریافت یک امتیاز جدید، میانگین بروزرسانی می‌شود.

دو مدل دیگر نیز مربوط به الگوریتم‌های محدود کردن تعداد درخواست امتیازهاست که در ادامه توضیحاتشان قابل مشاهده خواهد بود. دلیل اینکه این مدل‌ها در این برنامه قرار گرفته‌اند و نه در برنامه rate این است که خود مدل‌ها بیش‌تر به یک `BlogPost` مرتبط هستند و به مدل `BlogPost` ارتباط مستقیم دارند. با این حال الگوریتم‌هایشان هر بار در پی دریافت درخواست ایجاد امتیاز جدید اجرا می‌شوند و برای همین در بخش `rate/utils.py` قرار گرفتند.

- وجود فیلد `average_rating` به ما کمک می‌کند تا محاسبه و ارسال میانگین امتیازات در $O(1)$ انجام گیرد که در مقابل روش معمولی محاسبه میانگین با پیچیدگی زمانی $O(n)$، با اسکیل کردن سیستم زمان پاسخگویی تغیری نخواهد کرد.

---

[1] app

- مدل `User` که در خود جنگو موجود است دست نخورده و صرفا با اضافه کردن یک کلید خارجی[2] به مدل `BlogPost` و یک رابطه برعکس این مدل را پیاده‌سازی کردیم.

ویوها

این برنامه شامل سه ویو می‌باشد: `BlogPostCreateView`، `BlogPostsListView` و `BlogPostDisplayView`. وظیفه اولی ایجاد مطلب جدید است. وظیفه دومی نمایش تمام مطالب به کاربران است؛ دقت کنید که به طور معمول می‌توان از `ListAPIView` استفاده کرد تا از نوشتن کد تکراری جلوگیری کنیم؛ با این حال برای اینکه نمایش آن در مرورگر خواناتر باشد، از `APIView` و یک تمپلیت استفاده کردیم. ویو آخر نیز یک مطلب خاص را نمایش می‌دهد؛ در صورتی که کاربر به این مطلب امتیازی داده باشد، امتیاز کاربر به همراه فرم بروزرسانی این امتیاز به او نمایش داده می‌شود و در صورتی که امتیازی ثبت نکرده باشد، صرفا فرم ثبت امتیاز به او نمایش داده می‌شود.

برنامه rate

این برنامه شامل کارکردها و مدل `Rate` به منظور ثبت، تغییر و نمایش اطلاعات مربوط به امتیاز مطالب را شامل می‌شود.

مدل‌ها

شامل یک مدل `Rate` می‌باشد که شامل فیلدهای اصلی اطلاعات آن به همراه دو کلید خارجی یکی به `User` و دیگری به `BlogPost` مربوطه می‌باشد. روی ترکیب این دو کلید خارجی یک محدودیت همتا بودن[3] قرار داده شده تا مطمئن شویم هر کاربر برای هر مطلب فقط یک امتیاز ثبت می‌کند؛ البته این موضوع در ویو نیز بررسی شده. علاوه بر آن تابع `save` این مدل بازنویسی[4] شده تا هنگام ثبت یک امتیاز جدید، میانگین امتیازات `BlogPost` به صورت خودکار بروزرسانی شود.

- مدل‌های `BlogPost` و `User` تغییری نکرده و صرفا در این مدل یک کلید خارجی به این دو مدل اضافه شده.
- تابع `save` بازنویسی شده تا محسابه میانگین امتیاز مطالب به صورت خودکار انجام گیرد.

ویوها

شامل چهار ویو `RateCreateView`، `BlogRatingStatsView`، `UserRatingStatsView` و `UpdateUserRatingView` می‌باشد. ویو اولی به منظور ایجاد یک امتیاز جدید است؛ این ویو ابتدا بررسی می‌کند که کاربر قصد ایجاد امتیاز تکراری ندارد. سپس با توجه به تنظیمات اصلی، یکی از الگوریتم‌های EMA یا

---

[2] Foreign Key
[3] Unique Constraint
[4] Override

Leaky Bucket را انتخاب می‌کند؛ در صورتی که هر یک از این الگوریتم‌ها نتیجه دهند که درخواست فعلی باید محدود شود، امتیاز کاربر نادیده گرفته می‌شود. ویو دوم اطلاعات امتیاز یک مطلب و ویو سوم اطلاعات امتیاز یک کاربر برای یک مطلب را ارسال می‌کند. ویو آخر نیز صرفا امتیاز کاربر به یک مطلب را بروزرسانی می‌کند؛ دلیل جدا سازی بروزرسانی امتیاز با ثبت امتیاز ترجیح تعریف صریح نسبت به ضمنی است[5].

برنامه users

این برنامه صرفا شامل چند ویو برای مدیریت کاربران می‌باشد. منطق و مدل زیادی در آن وجود ندارد چرا که مقدار زیادی از آن‌ها به صورت پیشفرض در کتابخانه جنگو موجود می‌باشد.

الگوریتم‌های محدودسازی درخواست‌ها

دو الگوریتم محدودسازی درخواست‌های امتیازدهی پیاده‌سازی‌شده‌اند که در فایل `settings.py` می‌توان انتخاب کرد کدام یک از این دو الگوریتم اجرا شوند. به صورت پیشفرض الگوریتم Leaky Bucket فعال است. در محیط پروداکشن بهتر است این الگوریتم‌ها توسط یک نود ردیس یا دیگر فریمورک‌های کش کردن پیاده‌سازی شوند. با این حال برای ساده‌سازی این برنامه، اطلاعات مربوط به هر یک توسط مدل‌ها و در دیتابیس ذخیره می‌شوند.

الگوریتم Leaky Bucket

یک الگوریتم ساده با پارامترهای ثابت می‌باشد. روش کار این الگوریتم این است که یک ظرف[6] از درخواست‌ها را در نظر می‌گیرد. این ظرف یک ظرفیت (`bucket_size`) ثابت و یک نشتی ثابت (`leak_rate`) دارد. برای مثال در این برنامه به صورت پیشفرض، در هر ثانیه ۱۰ واحد نشتی داریم و ظرفیت کلی ظرف ۲۰ واحد است. نشتی تعیین می‌کند که حداکثر چند درخواست در هر ثانیه می‌توانیم داشته باشیم و ظرفیت تعیین می‌کند که در صورت بیش‌تر بودن تعداد درخواست‌ها از این محدودیت، تا چه تعداد قابل تحمل خواهد بود. مثلا اگر به مدت ۲۰ ثانیه، ۱۰ درخواست در ثانیه دریافت کنیم، درخواست بعدی رد خواهد شد.

این الگوریتم پارامترهای ثابت دارد که باعث می‌شود در بسیاری از شرایط مناسب نباشد. مثلا با توجه به سابقه یک مطلب و نویسنده آن ممکن است تعداد امتیازهای بیش‌تری را بخواهیم قبول کنیم، در حالی که تعداد درخواست زیاد برای یک مطلب ناشناس غیر معمول است. یک راه برای حل این مشکل استفاده از میانگین و انحراف معیار تعداد درخواست‌ها در پارامترهاست که این موضوع در الگوریتم بعدی پوشش داده شده.

از مزیت‌های این الگوریتم ساده بودن پیاده‌سازی آن است.

---

[5] Explicit is always better than implicit

[6] Bucket

الگوریتم EMA

این الگوریتم از فرمول زیر پیروی می‌کند:

$$Z = \frac{\mu - current\ rate}{\sigma}$$
$$\mu = request\ rate\ mean, \sigma = request\ standard\ deviation$$

روش کار این الگوریتم این است که سرعت درخواست جدید (اختلاف زمانی دریافت درخواست جدید با آخرین درخواست) را با میانگین سرعت موجود مقایسه می‌کنیم. سپس این مقایسه را نسبت به انحراف معیار نرمالایز می‌کنیم. استفاده از انحراف معیار باعث می‌شود تا این موضوع را در نظر بگیریم که آیا رفتار کاربران در گذشته در مقاطعی ناگهانی بوده و در مقاطعی ثابت. در صورتی که قبلا هم رفتار کاربران در مقاطعی ناگهانی بوده، مشکلی برای این درخواست جدید نباید پیش بیاید. در نهایت مقدار Z را نسبت به یک حد آستانه مشخص مقایسه می‌کنیم و در صورتی که سرعت درخواست‌های جدید بسیار زیاد بود آن‌ها را رد می‌کنیم. این الگوریتم در تعدادهای بالا همگرایی بهتری دارد. همچنین در صورت تداوم یک رفتار، مثلا بالا بودن سرعت درخواست‌ها برای مدت طولانی، به آن عادت کرده و درخواست‌ها را رد نمی‌کند.

برای پیاده‌سازی این الگوریتم ابتدا مدل `BlogRatingEma` را ساختیم که پارامترهای مورد نیاز برای محاسبه فرمول مورد نیاز را شامل می‌شود. این پارامترهای شامل `mean_request_rate`، `variation_sum` و `last_record` می‌باشند. میانگین تعداد درخواست‌ها به آسانی در هر بار دریافت درخواست جدید بروزرسانی می‌شود. با این حال بروزرسانی انحراف معیار کار آسانی نیست؛ برای این کار متغیر `variation_sum` را معرفی می‌کنیم که در اصل برابر $\sum(x_i - \sigma)^2$ می‌باشد (مجموع اختلاف سرعت رسیدن درخواست‌ها از میانگین). بنابراین در هر مرحله با دریافت درخواست جدید آن را بروزرسانی کرده و انحراف معیار جدید را محسابه می‌کنیم. روش دیگر محاسبه این بود که تمام درخواست‌ها را ثبت کنیم و هر متغیر را از ابتدا محاسبه کنیم؛ این روش بهینه نبوده و با زیاد شدن تعداد درخواست‌ها با مشکل بهینگی روبرو مواجه می‌شویم. با این روش محاسبه الگوریتم در $O(1)$ انجام شده و با اسکیل کردن سیستم به مشکلی نمی‌خوریم.

- با تغییر در محاسبات پیچیدگی زمانی از $O(n)$ به $O(1)$ کاهش یافت.
- الگوریتم نسبت به رفتار گذشته کاربران حساس بوده و پارامترهای آن ثابت نیستند.

تست‌ها

برای این دو الگوریتم در فایل `rate/tests.py` دو تست قرار گرفته. تست مربوط به الگوریتم Leaky Bucket زمان را توسط کانتکست منیجر `FreezeGun` نگه داشته و درخواست ثبت امتیاز توسط ۲۰ کاربر ارسال می‌کند. بدین صورت ظرفیت تعداد درخواست‌ها پر شده و ۲۱-امین درخواست با خطای `Too many requests` مواجه می‌شود.

برای الگوریتم EMA نیز ابتدا ۱۵ درخواست در بازه‌های ۵ ثانیه‌ای ارسال می‌کنیم و سپس ۶ درخواست در بازه‌های کوتاه‌تر ۵۰ میلی‌ثانیه‌ای ارسال می‌کنیم. ۶ درخواست آخر حداقل یکی باید با خطا مواجه شود. دلیل اینکه از عبارت

«حداقل یکی» استفاده می‌کنیم این است که محاسبه دقیق این که کدام درخواست با خطا مواجه می‌شود سخت بوده و صرفا مقابله با درخواست‌های ناگهانی مد نظر است.