به نام خدا

مصطفی مرادی

مستند چالش فنی بیت پین

تمام نیازمندی های چالش پوشش داده شده.

- دیدن لیست کانتنت ها به همراه میانگین امتیاز آنها
- دیدن جزئیات یک کانتنت (در این حالت اگر شما احراز هویت کرده باشید و توکن ارسال کنید امتیازی که وارد کرده اید برای آن کانتنت را خواهید دید، در غیر این صورت امتیاز به شما نشان داده نخواهد شد و نال خواهد بود، معمولا سمت فرانت هندل میشه)
- امتیاز دهی کانتنت ها
  - در این حالت حتما باید احراز هویت کرده باشید.
  - یک عدد ۰ تا ۵ ارسال میکند.
  - برای حل چالش حملات در زمان محدود از یک ایده استفاده شده که جلوتر توضیح داده میشه.
- امکان ثبت نام و گرفتن توکن احراز هویت هم وجود داره.

چالش حملات در زمان محدود:

برای حل این مسئله از این ایده استفاده کردم که یک threshold برای زمان و تعداد اقدامات برای ریت دادن یک کانتنت استفاده میکنیم. این مقادیر با توجه به تجربه حملات قابل حدس خواهند بود و اعداد استفاده شده در این پروژه فرضی هستند که در فایل .env میتونید تغییر بدید که برنامه متناسب با اون کار کنه.

عملکرد به این شکل هست که هر بار درخواست ریت دادن یه کانتنت داشته باشیم اون رو حتما کش میکنیم و براش یک counter در نظر میگیرم. اگر در یک بازه زمانی محدود که خودمون مشخص میکنیم، مثلا ۳۰ ثانیه، تعداد درخواست های ریت دهی برای اون کانتنت از threshold ما بالاتر رفت این یک حالت بحرانی حساب میشه و حدس بر این زده میشه که اون کانتنت زیر حمله هست. کاری که میکنیم اینه که برای یک بازه زمانی محدود که باز خودمون مشخص میکنیم چقدر باشه امکان ریت دهی رو برای اون کانتنت غیر فعال میکنیم. وقتی این بازه تمام بشه کانتنت از حالت قفل شدن بیرون میاد و دوباره میتونن کاربرا بهش ریت بدن.

پیاده سازی فنی:

1. **مدیریت کش (Redis):**

برای مدیریت شمارش تعداد درخواست های ریت و قفل کردن موقت کانتنت ها از Redis استفاده کردم. این کار باعث شده سیستم بتونه با سرعت بالا حملات را شناسایی و کنترل کنه.

2. **دیتابیس (PostgreSQL):**

اطلاعات کاربران، کانتنت ها و امتیازها در دیتابیس PostgreSQL ذخیره میشه. با این حال، عملیات سنگین

نوشتن در دیتابیس به حداقل رسیده است. در آینده میشه پایپ‌لاین هایی طراحی کرد که داده های کش رو بهینه تر به دیتابیس منتقل کنه.

3. **بهینه سازی محاسبه میانگین امتیاز:**

برای محاسبه میانگین امتیاز کانتنت ها، به جای اینکه هر بار تمام امتیاز ریت ها از دیتابیس خونده بشه و عملیات aggregation انجام بشه از یک فیلد average vote در مدل کانتنت استفاده شده است. این فیلد با هر امتیاز جدید، به صورت خودکار و در پشت صحنه توسط سیگنال (Signal) آپدیت می‌شود تا تاثیری کمتری روی ریسپانس تایم درخواست داشته باشه.

4. **استفاده از Celery Worker:**

برای کاهش بار پردازشی از سیستم، یه بخشی از مدیریت کش و پردازش ریت ها به Celery Worker سپرده شده.Middleware فقط درخواست ها رو به صف میفرسته و Celery وظیفه پردازش اونا رو به عهده میگیره.

نتایج تست عملکرد:

برای تست عملکرد سیستم، از ابزار **Locust** برای شبیه سازی کاربرا و بررسی رفتار سیستم تحت بار بالا استفاده شده. این تست نشان داد که:

- سیستم توانایی مدیریت تعداد بالای درخواست‌ها را دارد.
- محدودیت‌ها و قفل‌ها به درستی اعمال می‌شوند و مانع از حملات می‌گردند.
- زمان پاسخ‌ده ی درخواست ها پایدار بوده و در شرایط بحرانی نیز مدیریت مناسبی انجام شده است.
- البته باید در نظر داشت که این تست روی سیستم لوکال من انجام شده و طبیعتا منابع سرور خیلی بیشتر هست و تعداد نود های پاسخگو هم بیشتر هستن که باعث میشه نتایج بهتری داشت.

لطفا فایل README.md پروژه رو حتما مطالعه کنید. توضیحات دقیق تر اونجا موجود هست.