پروژه مصاحبه

پیش‌نیازها:

- تسلط به زبان پایتون و فریم‌ورک جنگو
- تجربه کار با دیتابیس MongoDB
- آشنایی با اتصال به MongoDB از طریق MongoEngine یا PyMongo

کلیت پروژه:

در این پروژه قصد داریم دو سرویس را توسعه دهیم که به مشابه بخشی از زیبال عمل می‌کنند. فرض می‌شود که سیستم بر اساس معماری **Service-Oriented** پیاده‌سازی شده و سرویس‌ها از طریق REST API با یکدیگر ارتباط برقرار می‌کنند.
* قسمت دوم این پروژه اختیاری است و انجام آن میتونه درک بیشتری از نحوه پیاده‌سازی شما به ما بده و حتما در جمع‌بندی و نتیجه‌گیری ما تاثیر مثبتی داره.

قسمت اول - گزارش تراکنش‌ها:

هدف این بخش پیاده‌سازی یک **Rest API** است که تراکنش‌های درگاه پرداخت اینترنتی کاربران را در قالب نمودارهای روزانه، هفتگی و ماهانه نمایش دهد. این API باید از دو ورودی type و mode پشتیبانی کند که به شرح زیر است:

ورودی‌ها:

- type: نوع خروجی که می‌تواند یکی از مقادیر count یا amount باشد.
- mode: نوع دسته‌بندی گزارش که یکی از مقادیر weekly ،daily یا monthly است.
- merchantId: کاربر (اختیاری، در صورت عدم ارسال، باید اطلاعات تجمیعی برای تمامی کاربران نشان داده شود).

خروجی‌ها: خروجی این API به صورت یک لیست از آبجکت‌های دارای دو کلید key و value خواهد بود که به شرح زیر است:

Daily:

```
[
    {
        "key": "1403/04/04",
        "value": 0
    },
    {
        "key": "1403/04/05",
        "value": 0
    },
]
```

Weekly:

```
[
    {
        "key": "1403 سال 6 هفته",
        "value": 26450580002,
    },
    {
        "key": "1403 سال 7 هفته",
        "value": 14560551127,
    },
....
]
```

Monthly

```
[
    {
        "key": "1403 شهریور",
        "value": 77947308487,
    },
    {
        "key": "1403 مهر",
        "value": 59446838593,
    },
...]
```

با توجه به حجم بسیار بالای تراکنش‌ها، بعضا فراخوانی کوئری‌های مربوط به این سرویس، سربار محاسباتی زیادی به همراه خواهد داشت.
اگر تعداد تراکنش ها چند میلیونی باشد اجرای pipeline روی این تعداد رکورد با کندی مواجه می شود. برای افزایش سرعت و بهینگی این API نیاز به ذخیره کردن این مقادیر در یک کالکشن جدید داریم؛ تا بدون اجرای کوئری‌های قبل در زمان بسیار پایین‌تر نتیجه را به کاربر نمایش دهیم.
به همین منظور، یک Command جنگو بنویسید که مقادیر لازم را محاسبه کند در یک Collection دیگر به نام transaction_summary ذخیره کند.
همچنین یک API جدید با ورودی‌های مشابه قبلی بنویسید که اطلاعات را از کالکشن جدید بخواند.

کالکشن transaction جهت استفاده شما ارسال شده است.

قسمت دوم - پایگاه اعلان(اختیاری):

هدف از این بخش، پیاده‌سازی یک **سرویس اعلان مرکزی** است که امکان ارسال پیام‌ها به کاربران و ادمین‌ها از طریق **چندین کانال ارتباطی** (SMS، ایمیل، Push Notification، تلگرام و ...) را فراهم کند. این سرویس باید به گونه‌ای طراحی شود که منطق ارسال پیام برای هر کدام از این کانال‌ها (Medium) به صورت مستقل و سفارشی باشد.

یکی از ویژگی‌های ضروری این سرویس، پشتیبانی از ارسال **پیام‌های Asynchronous** است، چرا که ممکن است پاسخ‌دهی از **Provider** (مانند کاوه‌نگار برای ارسال پیامک) با تأخیر مواجه شود. این تأخیر نباید باعث توقف یا بلاک شدن برنامه شود، و سیستم باید به صورت موازی و بدون وقفه سایر عملیات را ادامه دهد.

علاوه بر این، باید مکانیزمی برای **مدیریت ارسال‌های ناموفق** وجود داشته باشد. به عنوان مثال، ممکن است ایمیل‌ها با شکست مواجه شوند و سیستم باید قادر باشد که **سیاست ارسال مجدد** (retry policy) را پیاده‌سازی کرده و اطمینان حاصل کند که هر پیام **یک‌بار و فقط یک‌بار** ارسال می‌شود.

همچنین، نیاز به **گزارش‌دهی** ارسال پیام‌ها داریم. باید اطلاعات دقیق از پیام‌های ارسال شده در یک کالکشن ذخیره شود، به طوری که بتوانیم تاریخ و ساعت ارسال، نوع **Medium** و شخص دریافت‌کننده پیام را مشاهده کنیم. برای هر پیام، ممکن است چندین کانال ارتباطی در نظر گرفته شود؛ به عنوان مثال، گزارش روزانه تراکنش‌ها ممکن است به صورت همزمان از طریق پیامک، ایمیل و تلگرام ارسال شود.

پیام‌ها به صورت یک رشته متنی در نظر گرفته می‌شوند و باید بتوانند با قالب‌های مختلف سازگار شوند تا برای هر نوع **Medium** به درستی ارسال شوند.

لیست Mediumها:

- ارسال پیامک
- ارسال ایمیل
- ارسال از طریق بات تلگرام

توجه: قرار نیست منطق ارسال برای تمام **Medium** ها پیاده‌سازی شود. هدف این است که **زیرساخت** به گونه‌ای طراحی شود که امکان افزودن **Medium** های جدید به راحتی وجود داشته باشد و منطق ارسال آن‌ها به سادگی قابل پیاده‌سازی باشد.

سیستم باید قادر باشد پیام‌ها را به طور همزمان از طریق چندین مدیوم (مثلاً پیامک، ایمیل و تلگرام) ارسال کند.. مدل ارسال داده‌ها به هر کدام از **Thirdparty ها** (مانند APIهای مختلف) ممکن است متفاوت باشد. این قالب‌ها باید به گونه‌ای طراحی شوند که به هیچ‌وجه **Hardcode** نباشند و قابلیت انعطاف‌پذیری بالا برای افزودن و تغییر در آینده را داشته باشند.

نکته مهم:

این بخش **اختیاری** است، اما انجام آن به شما **امتیاز بیشتر** می‌دهد چرا که مهارت‌های شما در **مدیریت سیستم‌های ارسال پیام‌های Asynchronous**، **پیاده‌سازی سیاست‌های ارسال مجدد**، و **گزارش‌دهی دقیق** را نشان می‌دهد. علاوه بر این، طراحی ساختار باز برای افزودن **کانال‌های ارتباطی جدید** می‌تواند نشان‌دهنده توانایی شما در توسعه سیستم‌های مقیاس‌پذیر باشد.

نکات فنی که باید مدنظر قرار بدین:

- پیاده‌سازی با **Python + Django**
- ترجیحاً از **Django Rest Framework** استفاده شود.
- ذخیره‌سازی تمامی داده‌ها در **MongoDB**.
- معماری **Database** و مدل‌های رد و بدل شدن **REST API** به عهده شماست.
- تمرکز بر ارتباط بین سرویس‌ها نباشد، فرض می‌شود که ارتباط سرویس‌ها از طریق **REST API** برقرار می‌شود.
- نیازی به پیاده‌سازی **Authentication** نیست؛ فرض می‌شود که این سرویس قبلاً پیاده‌سازی شده است.

- تمامی سرویس‌ها باید **Dockerized** شوند.
- لزوماً نیازی نیست که هر سرویس در قالب یک **App** جداگانه باشد؛ می‌توانید هر دو سرویس را در یک **App** توسعه دهید.
- در ارسال اعلان‌ها، باید **Exactly Once** بودن رعایت شود. پیشنهاد ما استفاده از **Celery Queue** است.
    - ذخیره‌سازی اطلاعات اعلان‌ها نیز بهتر است با استفاده از سیستم صف انجام شود.
    - در صورت استفاده از **Celery**، از **Result Backend** استفاده کنید.

نکات تکمیلی:

- **زمان تخمینی برای انجام پروژه:** با در نظر گرفتن مطالعات و پیچیدگی‌ها، حدود **7-8 ساعت** خواهد بود.
- برای پروژه **Git Repository** بسازید.
- خروجی **postman** برامون حتما ارسال کنید.
- یک **چک‌لیست** در انتهای پیپر برای راحتی کار شما قرار داده‌ایم.
- در صورت بروز هر گونه ابهام، با ما تماس بگیرید و ایمیل ارسال کنید.

TODO LIST:

- ☐ ایجاد دیتابیس و import داده‌های **transaction**
- ☐ ایجاد **Git Repository**
- ☐ ستاپ اولیه و ساخت پروژه + **Docker**
- ☐ اتصال **Django** به **MongoDB**
- ☐ پیاده‌سازی **Aggregation Pipelines** که از `mode` و `type` پشتیبانی کنند
- ☐ پیاده‌سازی **API** که خروجی مرحله قبل را در قالب ذکر شده برگرداند
- ☐ پیاده‌سازی یک **Command** برای کش کردن اطلاعات موجود در **transaction**
- ☐ پیاده‌سازی یک **API** که همان خروجی ذکر شده را با استفاده از داده‌های کش‌شده بهینه‌تر برگرداند
- ☐ پیاده‌سازی کلاس‌های مربوط به اعلان‌ها و **API**های مربوطه
- ☐ پیاده‌سازی توابع ارسال اعلان برای هر **Medium** - در نهایت صرفاً اطلاعات اعلان به همراه نوع **Medium** چاپ شود
- ☐ پیاده‌سازی منطق **تعریف قالب پیام**
- ☐ ساخت **Postman** برای تست APIها
- ☐ ارسال پروژه برای ما