بسم الله الرحمن الرحیم

توضیحات پروژه google\_scraper

این برنامه به صورت backend یک سرور با استفاده از فریمورک falcon نوشته شده است.

ابتدا یک پوشه به نام google\_scraper ایجاد می کنیم و درون آن فایل خالی \_\_init\_\_.py را ایجاد می کنیم تا پوشه تبدیل به ماژول شود.

سپس فایل app.py را ایجاد کرده و درون آن شیء های سه Resource به نام های Images و search و task را ایجاد کرده و برای هر کدام یک آدرس url اختصاص می دهیم.

برای اجرای برنامه باید دستور python app.py را در یک command prompt وارد کنیم.

در فایل ImageResource.py برای get صفحه image‌ را پیاده سازی می کنیم به این صورت که در querystring ها مطلبی که می خواهیم در گوگل ایمیج سرچ شود را با query= دریافت می کنیم. این query= اجباری است در غیر این صورت به خطای BadRequest دچار می شویم. می توانیم دو کوئری دیگر به نام های num و page نیز داد که برای pagination خروجی استفاده می شود. در حالت پیشفرض صفحه ۱ و تعداد ۲۰ پاسخ برگردانده می شود.

سپس با استفاده از تابع get\_collection\_data که در فایل database وجود دارد داده ها را دریافت کرده و اگر تعداد داده ها مخالف صفر بود نتایج را بر می گردانیم. اگر تعداد داده ها برابر صفر بود نشان می دهد که ما در دیتابیس خود این کوئری را قبلاْ نداشته ایم پس حال باید این اطلاعات را از گوگل دریافت کنیم به همین دلیل با استفاده از celery‌ یک task تعریف کرده و آیدی آن را به کاربر برمی گردانیم.

تابع onget در حالت کامنت این طور پیاده سازی شده است که اگر تعداد داده ها صفر بود همانجا داده ها را از گوگل دریافت کرده و سپس برای کاربر ارسال می کنیم. این کار باعث می شود که سرعت پاسخ ما زیاد شود.

فایل SearchResource.py کاملا مانند ImageResource.py نوشته شده است با این تفاوت که هنگام کوئری زدن به گوگل سرچ کوئری زده می شود و نه گوگل ایمیج. البته این تفاوت در scrape.py مشخص می شود.

در فایل scrape.py از پنج تابع استفاده شده است.

تابع google\_image\_search یک کوئری را دریافت می کند و آن را درون url گوگل ایمیج گذاشته و صفحه html‌ را به عنوان خروجی بر می گرداند.

تابع result\_from\_image\_page صفحه html ای که از تابع قبل دریافت کرده را می گیرد و اطلاعات عنوان؛ آدرس پیج آن عکس؛ و اطلاعات آن عکس را ذخیره می کند. منظور از اطلاعات عکس دو حالت است در ۲۰ عکس اول کد base64 عکس ذخیره می شود. البته این کد ایراد دارد چرا به صورت کامل در صفحه لود نشده است و فقط یک خط است به همین دلیل قابلیت تبدیل به عکس را ندارد. اما حالت دوم آدرس url عکس است که می توان این عکس را دانلود کرد. به عنوان ورودی رشته کوئری نیز گرفته می شود تا به هر کدام از نتایج در انتها اضافه شود.

تابع google\_search یک کوئری را دریافت کرده و آن را درون url گوگل سرچ می گذارد و صفحه html آن را به عنوان خروجی بر می گرداند.

تابع result\_from\_page صفحه html حاصل از تابع قبل را گرفته و اطلاعات هر نتیجه سرچ را ذخیره می کند. عنوان و توضیحات و لینک نتیجه ذخیره می شود. به عنوان ورودی رشته کوئری نیز گرفته می شود تا به هر کدام از نتایج در انتها اضافه شود.

تابع fill\_database ابتدا نام collection را که می تواند image و search باشد (مطابق با چیزی که در mongodb از قبل ساخته ایم) را به همراه کوئری دریافت می کند سپس با توجه به نام collection مشخص می شود که باید از توابع image استفاده کند یا search‌ سپس نتایج را ابتدا با شیء database در دیتابیس ذخیره کرده و سپس آن را بر می گرداند.

در فایل database.py یک کلاس database ساخته می شود که در init آن به دیتابیس وصل شده و collection‌ آن در self.collection ذخیره می شود.

در تابع get\_collection\_data کوئری به همراه شماره صفحه و تعداد گرفته شده و نتیجه بازگردانده می شود.

در تابع add\_documents لیست ورودی ها گرفته می شود و با استفاده از insert\_many همه آن ها به collection مد نظر اضافه می شوند.

در فایل tasks.py به celery متصل شده و کوئری و نام collection‌ را گرفته و scrape.fill\_database را صدا می زنیم سپس نتایج را باز می گردانیم.

فایل TasksResource.py نیز برای آدرس task‌ نوشته شده است. در حالت get می آید وضعیت task و نتیجه آن را به کاربر بر می گرداند.

برای امتحان کردن کد در یک command prompt باید دستور python app.py را وارد کنیم تا این فرآیند حالت سرور را بگیرد.

در یک command prompt‌ دیگر باید دستور http localhost:8000/search?query=”salam” را بزنیم (به جای search می توان image نیز نوشت و به جای salam نیز هر چیز دیگری می توان نوشت. می توان کوئری های num و page را نیز اضافه کرد.).

در یک command prompt دیگر باید دستور celery -A tasks worker -l info --pool=solo را وارد کنیم تا بخش celery بتواند به درستی کار کند. البته برای آنکه celery بتواند درست کار کند باید redis نیز بر روی سیستم نصب و بالا باشد.

اگر دستور دوم در دیتابیس وجود داشته باشد نتایج برگردانده می شود و اگر نباشد یک taskid برگردانده می شود. حال باید دستور دیگری را در command prompt دوم وارد کرد. دستور این است http localhost:8000/tasks/{taskid} که به جای taskid آیدی مورد نظر را قرار می دهیم. اگر task به اتمام رسیده باشد نوع پاسخ SUCCESS می شود. و اگر عملیات هنوز به اتمام نرسیده باشد نوع پاسخ PENDING می شود.

بیرون از پوشه google\_scraper فایل main وجود دارد که دارای دو تابع قابل توجه است.

در تابع download\_web\_page یک url داده می شود به همراه متغیر پرچم threaded که این متغیر دوم باعث می شود که کار با چند نخ انجام بشود یا خیر. در این تابع یک صحفه از وبسایت با استفاده از ماژول pywebcopy دانلود می شود و در پوشه path ذخیره می شود. اطلاعات hostname و path ای که به خود صفحه راه دارد و همچین datetime ای که این صفحه دانلود شده است با هم اسم پوشه ای که صفحه در آن دانلود می شود را می سازد (استفاده از زمان به این دلیل است که شاید یک صفحه دوباره خواست دانلود شود و باید اسم جدید داشته باشد)

در تابع download\_image\_of\_query نام کوئری را دریافت کرده و نتیجه آن کوئری را که از تابع result\_from\_image\_page را می گیریم را نیز دریافت کرده و شمارنده ای را نیز می توانیم به تابع بدهیم. در این تابع می خواهیم عکس هایی از گوگل ایمیج که url‌ دارند را دانلود کرده و در پوشه جدیدی که به نام کوئری ثبت می شود قرار دهیم. به دلیل آنکه لینک هایی که برای عکس ها هستند فاقد نام مناسب هستند مجبوریم از اعداد برای نام فایل ها استفاده کنیم به همین دلیل اگر بخواهیم برای یک کوئری دوبار تابع را صدا بزنیم لازم داریم که شمارنده را تغییر دهیم در غیر این صورت عکس های جدید روی عکس های قبلی می نشیند.