Os primeiros passos

O primeiro passo foi ler a documentação do projeto e entender o objetivo do sistema. Depois disso, escolhi o tema das notícias com as quais iria trabalhar e, como sou um gamer assíduo, não teria tema melhor senão “Games”. Após a escolha do nicho, meu próximo passo foi saber quais são os sites mais populares dessa categoria. Então, fiz uma breve pesquisa e decidi que os melhores sites seriam: IGN Brasil, Game vício e Adrenaline.

Em seguida, dediquei-me a visualizar como seria o projeto final, quais tecnologias usaria e como tudo se integraria. Optei por usar a linguagem de programação Python, juntamente com a ferramenta Selenium, pois já tive experiências com ambas as tecnologias. Sabia que em algum momento iria usar a biblioteca pandas para tratar os dados obtidos e salvar em formato csv, então já deixei ela importada também. Porém, mais adiante, depois de tentar usar o csv com o Docker, percebi que não seria tão interessante utilizar essa arquitetura. Então decidi trabalhar com outra estrutura de banco de dados, que é o MongoDB.

Depois de visualizar a solução na minha mente, percebi que já estava utilizando muito do meu tempo arquitetando tudo. Então, decidi colocar a mão na massa e comecei a fazer os primeiros testes. Abri o meu Spyder, importei o Selenium e o pandas, coloquei a URL da IGN pra rodar e fazer a captura dos primeiros elementos. Depois de inspecionar as tags HTML e fazer alguns testes, descobri que o identificador único para as notícias, utilizando css selector, é encontrado utilizando a palavra NEWS, "[class *= 'NEWS']". Com este selector, consegui selecionar apenas os elementos que continham as notícias.

Escrevendo as primeiras linhas de código

Com a referência de todas as notícias visíveis na página, criei uma lista com todas as urls das notícias para usar posteriormente na minha raspagem. Abri a primeira url de teste e comecei a caçar no código fonte os elementos de título, sub título, nome do autor, data de publicação e quantidade de comentários. Todos foram bem fáceis de conseguir retornar o elemento, com exceção da quantidade de comentários, pois se encontrava em outro *iframe*, o que me deu uma dor de cabeça grande. Mas, depois de algumas horas pesquisando no stack overflow e pela inter webs, descobri como solucionar este problema. Caso alguém queira entender melhor como funciona o processo de iframes e trocar entre eles, vou deixar o link abaixo de onde aprendi: https://www.techbeamers.com/switch-between-iframes-selenium-python/

Quando eu achava que tudo já estava tranquilo, rodei o código mais uma vez e crashou. Depois de um tempo verificando, percebi que o id e o nome do *iframe* que possui o elemento mudam sempre que é aberta, ou seja, não é um id único (vai entender, né?!). Então, precisava de uma nova estratégia para lidar com isso. Depois de um tempo fuçando, decidi criar um laço de repetição que procura em todos os *iframes* por um determinado elemento, coloquei um “ try catch” e uns prints, e vuala! tudo certo! Consegui retornar o elemento com as quantidades dos comentários e, com um sorriso no rosto, parti para a próxima etapa. Fiz uma bateria de testes automatizados para conferir se estava tudo ok, mas, ao chegar na url da notícia, descobri que os *iframes* ficam visíveis apenas quando rolamos a página até o fim, ou seja, eles são criados dinamicamente e não estão presentes logo quando entramos na página, o que me gerou mais algumas linhas de código para contornar essa situação.

A solução que eu encontrei foi executar um comando no selenium para fazer o *scroll down* da página, porém, só um *scroll down* não bastava para aparecer o *iframe* dos comentários. Então, precisava fazer duas vezes para que ficasse tudo redondinho. Até o momento, estava fazendo testes em apenas uma página de artigo. Então, decidi começar a fazer testes com todos os links que eu havia retornado em uma lista. Fiz um laço de repetição pra fazer a raspagem em cada link. Rodando o código, percebi que, quando havia um erro, ele parava, e não era isso que eu queria. Então, coloquei dentro do laço um tratamento de exceção, ou seja, caso aconteça um erro durante o scraping de uma página, o programa pula para o próximo link, deixando o programa muito mais tolerante a falhas. Depois de rodar todos os links, o programa volta e roda apenas os links que deram erro na primeira tentativa. Eis o resultado:

No caso do IGN, assim que o driver abre a página, consegue retornar apenas alguns links das notícias, caso você queira mais, precisa rolar a pagina pra baixo, pois a IGN trabalha exatamente como o Instagram, utilizando infinite scroll. Sabendo disso, bolei uma estratégia para pegar as notícias de pelo menos um mês inteiro, rolei a página para baixo até chegar na data de um mês atrás e percebi que a quantidade fica em torno de 250 notícias. Para evitar uma execução tão demorada, decidi pegar até 50 notícias e raspar apenas 5 por execução.

Armazenando os dados

Após fazer scraping nas páginas, armazenei os dados em uma lista e depois criei um *dataframe* com o pandas. Porém, quando fui escrever o *dataframe* em um csv, percebi que teria que pensar direitinho qual delimitador usar, pois estamos tratando de texto, e os principais separadores como ',', ';' e '|' estavam aparecendo nos textos do título. Resolvi escolher o pipe('|') como delimitador do meu csv, mas, antes de escrever, precisava fazer uma varredura no *dataframe* e substituir tudo que era pipe por um traço('-'). Dessa forma, asseguro-me de que não terei problemas com delimitador. Além disso, aproveitando que estava fazendo uma limpeza nos dados, formatei o campo da quantidade de comentários também, deixando apenas a quantidade em um formato int e sem a string comentário, isso será útil posteriormente quando fizermos as análises. Além disso, o armazenamento vai ser mais eficiente por ocupar menos espaço em disco.

Fiz a escolha de salvar em um csv para poupar tempo e diminuir as dependências do projeto, mas, em um ambiente de produção, provavelmente um banco NoSQL como MongoDB ou até um SQL como MySQL seria melhor para estruturar o sistema. Caso o projeto precise ser escalado, sugeriria até pensar na possibilidade de utilizar um ecossistema Hadoop com o banco em um sistema de arquivos distribuídos (HDFS), utilizando HIVE e pySpark para fazer as consultas. Porém, botando o pé no chão, pois temos apenas alguns dias para entregar o projeto, vamos manter o csv.

Com relação a estratégia incremental de inserção de novos dados, em minhas experiencias, sempre fiz um left join com a base para inserir apenas os dados que não se encontram na base, porém, neste caso das raspagens dos artigos, seria muito custoso para o algoritmo fazer toda a raspagem e, só no final, verificar o que é novo para base. Então, decidi fazer uma função (returnOnlyNewLinks()), que, antes de raspar os links, ele verifica na base quais já estão lá e me retorna apenas as URLs que não estão, fazendo com que o processo fique muito mais rápido e não desperdice processamento. Depois de raspar apenas os artigos necessários, o algoritmo faz um union(pd.concat()) entre os dados que já estavam na base e os novos.

O Mongo DeuBom

A Estratégia Incremental acima era utilizada quando eu ainda estava trabalhando com csv, porém, depois de ter escolhido trocar para o MongoDB, precisei apenas pegar os dados novos e inserir na base, simples assim =D. O que me motivou a trocar a base de dados de csv para o Mongo foi a complexidade que seria sincronizar os arquivos csvs com os containers docker, tendo que gerenciar volumes de dentro e fora do container. Já com o MongoDB, todo o processo de requisição e envio de dados acontece de formar isolada.

App.py

Depois de ter deixado tudo redondinho para uma única página, decidi reproduzir os mesmos passos para as demais páginas. Tentei modularizar ao máximo as funções para que não existisse repetição de código, já que a estrutura é bem parecida de uma página para outra. Depois de fazer as três páginas rodarem com sucesso, fiz um script principal(/app.py) que chama os outros scrapers. Com isso, consegui centralizar os três scripts em um único lugar e, posteriormente, quando formos rodar o projeto, precisamos executar apenas o app.py.

Bug – Aprendendo com os erros

Vários bugs ocorreram durante o processo, vou elencar apenas os principais para não tomar muito tempo.

- **Batata Quente**

  Precisei passar e retornar o webdriver como parâmetro para todas as funções que utilizam o driver. Caso não seja retornado, na próxima vez que ele for utilizado, você toma um erro de Session ID na cabeça pra ficar esperto. Tentei usar o driver como variável global(na minha cabeça daria certo, mas não foi bem isso que aconteceu). Então, a solução foi sempre passar o driver como argumento na chamada de uma função e retorná-lo no fim.

- **A instalação do pandas no docker**

  Depois de todas as tentativas, percebi que o pandas é a biblioteca mais difícil de se colocar em um container, e poucos são os guerreiros que conseguem instalar essa lindeza lá. Eu, iniciando há dois dias no Docker, obviamente não iria conseguir. Porém, tive a brilhante ideia de baixar uma imagem pronta no Docker Hub que já possuía o pandas(não sei como essa pessoa conseguiu essa proeza) e instalar as outras bibliotecas que eu precisava, como o pymongo e o selenium, e funcionou. Esta imagem pode ser vista no Dockerfile disponibilizado no github. Demorei um pouco para aprender a mexer no Docker, confesso que, no início, foi um pouco difícil, mas depois que peguei o jeito, acabei me apaixonando pela ferramenta. Desde o início, queria que o projeto fosse algo simples de transportar da minha máquina para produção. Já estava nos meus planos estudar Docker, mas foi só depois da necessidade de produzir este projeto que eu coloquei a mão na massa e aprendi na tentativa e erro.

Estratégia de deploy

Escolhi utilizar o Docker por ser simples e prático de fazer um deploy, além de deixar o ambiente em produção extremamente isolado de outras funções do Sistema Operacional. Com apenas alguns comandos, é possível transportar o código que roda na minha máquina, com todas as dependências, para produção. Tudo isso graças a criação das imagens docker. A minha estratégia de deploy para este projeto seria criar uma máquina virtual na nuvem, rodar minhas imagens docker dos containers necessários e colocar um Job Scheduler para ficar rodando os containers de tempo em tempo, tudo isso de forma automática.

Escalando a solução

Atualmente, o sistema utiliza um container para fazer o scraping sequencial das 3 páginas, porém, podemos escalar essa solução fazendo um grid de containers, ou seja, cada página de noticia teria um container único e os processos rodariam em paralelo, otimizando tempo de execução dos scripts. Além disso, outras páginas do ramo podem ser incorporadas, fazendo com que o sistema obtenha dados de mais fontes e, consequentemente, mais informações para serem analisadas e insights para serem gerados. Caso o projeto chegue a um ponto de lidar com uma grande quantidade de dados, poderíamos pensar numa estratégia utilizando o ecossistema Hadoop.

Minha experiência

O envolvimento neste projeto, acrescentou muito no meu conhecimento em python, pandas e selenium, assim como me fez explorar novos territórios, como Docker e mongoDB, por exemplo. Tive alguns momentos de desespero e frustração, mas sei que tudo isso faz parte de qualquer desafio (O próprio nome já diz), e graças a Deus e ao meu desejo de ver esse projeto concluído, continuei seguindo em frente. Aprendi muito com cada bug, erro e frustração. Confesso que foram os dias mais intensos da minha vida, e valeu a

pena cada segundo. A quantidade de conhecimento que obtive nesse processo não tem como mensurar. Só tenho a agradecer a empresa Oncase pela oportunidade de participar deste processo. E que venham mais desafios =D.