BigPy: um Sistema WEB para captura, tratamento e visualização de dados de defesa do consumidor utilizando Python

Weder Alves Ribas[1], **Edgar Noda**[1], **Daniela Marques**[1]

[1]Campus Hortolândia – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) 13183-250 – Hortolândia – SP – Brasil

me@wederribas.com, edgar.noda@ifsp.edu.br

marquesdaniela@ifsp.edu.br

Abstract. *Since the diffusion of the Internet for the public and commercial purposes, it has become a powerful tool for the development of a huge variety of applications. The amount of data generated by its usage and access is huge, offering the opportunity for the development of applications that allow the analysis of the data, providing relevant and useful information. In this context, one of its applications is the analysis of consumer protection data. This paper presents the development of a Web application, called BigPy, for data capture, processing and analysis, using as example the open data from the public portal Consumidor.gov.br.*

Resumo. *Desde a liberação de uso público e comercial da Internet, esta se transformou em uma grande ferramenta para a publicação e construção dos mais diversos tipos de aplicações. A quantidade de dados gerada pelos acessos à rede é gigantesca, propiciando diversas oportunidades para o desenvolvimento de aplicações que analisem os dados e forneçam informações relevantes neles contidas. Nesse contexto, uma de suas possíveis aplicações é a análise de dados de defesa do consumidor. Este trabalho expõe o desenvolvimento de uma aplicação Web, chamada BigPy, para captura, tratamento e análise de dados de defesa do consumidor, utilizando como exemplo os dados abertos do portal público Consumidor.gov.br.*

1. Introdução e Motivação

Desde sua abertura ao uso comercial e público, a rede mundial de computadores, *Internet*, constitui-se como a maior e mais completa ferramenta na busca de informações. Nela, é possível encontrar informações sobre os mais variados assuntos, a um custo extremamente baixo, em sua maioria de acesso gratuito. Além disso, segundo dados do relatório da *International Telecommunications Union* (ITU, 2014), aproximadamente 3 bilhões de pessoas possuem acesso à *Internet* em todo o mundo. Esses dados consolidam a rede como uma ferramenta valiosa para vários propósitos, comerciais ou não.

No cenário brasileiro, a presença da *Internet* não é diferente. Cada vez mais indivíduos, de todas as faixas etárias, acessam a rede para os mais variados fins. Para verificar este cenário, o Comitê Gestor da Internet no Brasil (CGI.br) realizou a pesquisa de Uso de Tecnologias de Comunicação e Informação, indicando que a

população brasileira ativa na *Internet* em 2013, corresponde a 51% da população (aproximadamente 85,9 milhões de pessoas) [CGI.br]. Percebe-se aqui, o grande potencial de alcance existente nas aplicações *Web*, tanto nacional quanto internacionalmente.

Uma das aplicações que se pode encontrar na *Internet* é o acesso a diversos portais para defesa do consumidor, que buscam defender os interesses e direitos dos consumidores frente às empresas. Atualmente existem portais tanto privados como públicos, sendo um exemplo de portal público o Consumidor.gov.br. Criado pela Secretaria Nacional do Consumidor do Ministério da Justiça, como meio alternativo de solução para conflitos entre consumidores e empresas por meio da *Internet*. Este portal tem como objetivo possibilitar ao Governo a visualização do estado dos serviços prestados pela empresas, mediante os dados coletados na intermediação de disputas. Além disso, os dados coletados no portal são disponibilizados para o público geral, possibilitando a construção de ferramentas úteis aos consumidores.

Além do valor da informação contida no portal, outra característica que o torna interessante é a disponibilização dos dados coletados, uma iniciativa própria do portal Consumidor.gov.br, permitindo à sociedade o acesso a diversos tipos de informações neles contidas. Essa disponibilização de dados ocorre mensalmente, com a liberação de arquivos em formato CSV (*Comma Separated Values*), que podem ser acessados em aplicações como o Microsoft Excel ou importados para bancos de dados (relacionais ou não relacionais). Foi também desenvolvido um dicionário de dados pelo portal, contendo as definições e representações dos elementos contidos nos arquivos liberados. Um dos pricipais motivos da escolha do portal Consumidor.gov.br para este trabalho está relacionado à qualidade dos dados, reduzindo a quantidade de iterações para realizar a limpeza dos dados (necessidade de processo *data warehouse*).

Na Figura 1 está um recorte da opção do portal que permite o *download* dos dados:

Figura 1. Disponibilização dos dados no portal. Fonte: Consumidor.gov.br 2018.

No portal Consumidor.gov.br, o usuário pode realizar algumas consultas a indicadores gerais por empresa ou infográficos que demonstram o número de reclamações por região (conforme mostrado na Figura 2). Porém, as opções de

visualização são poucas, limitando-se ao número de reclamações por região e o percentual de reclamações atendidas por cada empresa.

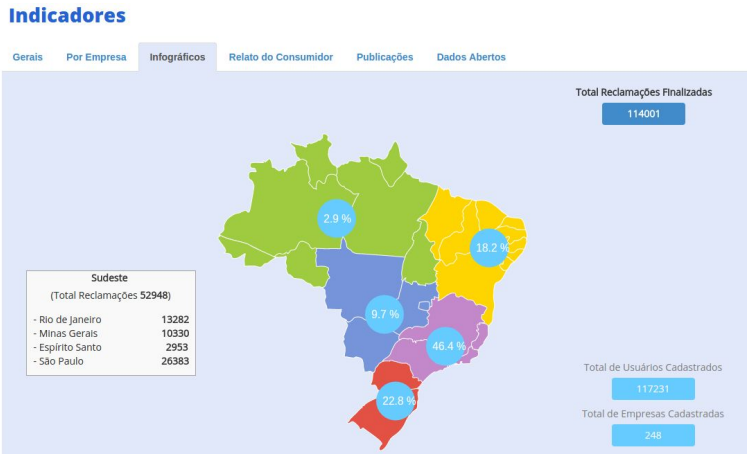

Figura 2. Infográfico de visualização no portal. Fonte: Consumidor.gov.br 2018.

Dada a relevância dos dados contidos no portal e o potencial de constituir-se em um artefato de utilidade pública, considerando o potencial de informação disponível e que ainda não conta com ferramentas adequadas para sua visualização, este trabalho objetiva o desenvolvimento de uma aplicação em plataforma *Web*, chamada de BigPy (acrônimo para "*Big Data with Python*"), que possibilite a captura, tratamento e análise de dados de defesa do consumidor, utilizando como prova de conceito os dados abertos do portal Consumidor.gov.br.

A ferramenta terá uma arquitetura flexível para armazenar e analisar dados de diversas fontes, não exigindo alterações em sua arquitetura para a inclusão de novas fontes de dados. A aplicação desenvolvida, visa possibilitar aos seus usuários extrair informações relativas à prestação de serviços por parte das empresas, permitindo uma tomada de decisão mais efetiva por parte do consumidor quando relativo à necessidade de consumir produtos e serviços de uma determinada empresa.

Pretende-se também possibilitar o acesso à informações de forma estruturada aos usuários sem que os mesmos necessitem de conhecimentos profundos em informática ou programação de computadores, ampliando assim a cognição dos usuários frente aos dados disponibilizados e o potencial número de usuários beneficiados pelas informações contidas no portal.

2. Análise de Sistemas Similares

A fim de melhor compreender o universo em que este tipo de aplicação está inserido e até certo ponto, validar a aplicação proposta bem como sua aplicação real para os usuários, foram pesquisados sistemas com propostas similares que ofereçam a análise de dados para reclamações de consumidores.

Neste levantamento, foram encontrados diversos *websites* contendo reclamações de consumidores sobre empresas, sendo considerada uma aplicação com relevância significativa para este trabalho.

A aplicação considerada foi o site Reclame Aqui [Reclame Aqui 2018]. Neste site, é possível incluir novas reclamações de consumidores contra empresas, visualizar reclamações registradas por outros usuários, visualizar dados gerais sobre o desempenho de uma empresa no tratamento das reclamações dos usuários e, também, comparar empresas entre si.

Embora apresente similaridades com o BigPy, o Reclame Aqui é focado na intermediação da reclamação do consumidor, e seu tratamento posterior pela empresa, que também pode acessar a plataforma para responder aos seus consumidores. O Reclame Aqui também apresenta uma análise do desempenho das empresas na solução de reclamações dos consumidores. A Figura 3 apresenta um recorte da tela com o resumo dos dados de reclamações.

Figura 3 . Portal Reclame Aqui

A análise das soluções similares e o atual nível de informação disponibilizado no site consumidor.gov,br demonstra que a aplicação BigPy possui relevância para sua implementação, permitindo desenvolver variadas análises e visualizações dos dados de reclamações dos consumidores, com gráficos dinâmicos, além de permitir a adição de dados de diversas outras fontes, por adotar um modelo de mapeamento dos dados de forma não-relacional.

3. Referencial Teórico

Nesta seção, apresenta-se o referencial teórico utilizado na concepção e desenvolvimento deste trabalho, contendo conceitos fundamentais da arquitetura e metodologia utilizadas.

3.1 Desenvolvimento *Web* e a adoção de *Frameworks*

O desenvolvimento de aplicações para a *Web* constitui-se como um importante ramo da Informática, possibilitando a publicação de conteúdo dinâmico, com alta disponibilidade e acesso universal aos indivíduos conectados à rede. Com relação ao desenvolvimento *Web*, Deshpande [Deshpande 2003], sugere que a Engenharia *Web* é a aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para o desenvolvimento, operação e manutenção de aplicações baseadas na *Web*. Nesse contexto, o desenvolvimento *Web* apresenta algumas características que o difere do desenvolvimento de aplicações *desktop* estáticas. Segundo Pressman (2011, p. 37) os

sistemas *Web* apresentam grande sensibilidade ao conteúdo, definindo sua qualidade, e uma carga não previsível, podendo haver grandes variações de acesso simultâneamente. Essas características diferem o desenvolvimento *Web* da criação de *softwares* estáticos (por exemplo, aplicações *desktop)*.

Dada sua dinamicidade, a *Web* possui uma grande gama de ferramentas para a construção de aplicações. Sua estrutura básica, é composta pelo HTML (*Hyper Text Markup Language*) para a construção das páginas e sua estruturação. Também utilizando o CSS (*Cascade Style Sheets*) para a definição de estilos e apresentação das páginas. Entretanto, o desenvolvimento de aplicações atraentes e com diversas funcionalidades, exige o uso de outras ferramentas, sejam elas de comunicação com os servidores, quanto a geração de páginas dinâmicas. Nesse contexto, a adoção de um *framework* de desenvolvimento é imprescindível para o sucesso de uma aplicação na *Web 2.0*, definida por [VOSSEN 2010] como a junção de novas técnicas de desenvolvimento, a adoção de novas tecnologias e o novo padrão de utilização da rede por seus usuários. Como definição de *framework* temos:

> "Um framework de desenvolvimento é uma "base" de onde se pode desenvolver algo maior ou mais específico. É uma coleção de códigos fonte, classes, funções, técnicas e metodologias que facilitam o desenvolvimento de novos softwares." (MINETTO, 2007)

Em concordância com a ideia proposta por Minetto (2007), a adoção de um *framework* permite uma construção da aplicação de maneira ágil e com um portfólio de ferramentas que possibilitam a construção de melhores produtos.

3.2 Análise e Visualização de Dados

Para que uma informação se torne atraente ao seu receptor, esta precisa ser apresentada de forma a possibilitar rápido entendimento e compreensão desta informação. Segundo Oliveira 2003 a visualização da informação utiliza-se da capacidade do sistema visual humano na identificação de padrões. Nesse contexto, disponibilizar gráficos, infográficos e tabelas com resumo dos dados, possibilita aos usuários desta aplicação ampliar rapidamente seus conhecimentos com relação às empresas presentes nesses dados.

3.3 Desenvolvimento Orientado a Testes (TDD)

O conceito básico do Desenvolvimento Orientado a Testes (TDD), estabelece que inicialmente seja escrito um caso de teste antes da escrita de qualquer código fonte. Este caso de teste irá, primariamente, falhar em sua execução, uma vez que nenhum código foi implementado (Figura 4). Em sequência, o desenvolvedor deverá compor o código que passe no caso de teste proposto, forçando assim o desenvolvedor a pensar no problema a ser resolvido com o código, com o desenvolvimento da mínima quantidade de código necessária para que o teste tenha sucesso [SALE 2014].

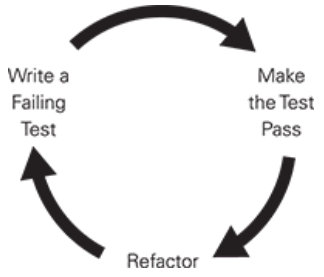

Figura 4 . Ciclo do Desenvolvimento Orientado a Testes (TDD) [Sale 2014]

3.4 Design Responsivo

O desenvolvimento de aplicações com Design Responsivo consiste na elaboração de aplicações que se adaptem ao ambiente onde os usuários visualizam a aplicação, independentemente do tipo de dispositivo utilizado pelo usuário. Esta adaptação deve acontecer de forma sutil, mantendo o *design* geral da aplicação em qualquer dispositivo [LAGRONE 2016].

3.5 Arquitetura de Micro-serviços

A arquitetura de micro-serviços (*microservices architecture*) é definida por Newman 2015 como um conjunto de pequenos serviços autônomos que atuam em conjunto, onde cada serviço é responsável por uma parte específica da aplicação.

A adoção desse modelo de arquitetura provê diversas vantagens ao processo de desenvolvimento da aplicação. Segundo Newman 2015, os principais benefícios adicionados são:

- Heterogeneidade Tecnológica: uma vez que o sistema não está fortemente acoplado, é possível adotar diferentes tecnologias para cada serviço, selecionando aquelas que forem mais adequadas para cada micro-aplicação;
- Resiliência: possíveis falhas que ocorrerem não são propagadas a todo o sistema, sendo possível mapear o problema mais facilmente;
- Escalabilidade: cada micro-serviço pode ser escalado separadamente, dependendo da demanda em cada um. Não é necessário prover o mesmo tipo de infraestrutura para todos os serviços;
- Facilidade de implementação: cada serviço pode ser implementado em produção separadamente, sendo independentes entre si, reduzindo também o risco de grandes impactos em sistemas produtivos em caso de falha.

3.6 Modelo de Desenvolvimento Incremental

De acordo com Pressman 2016, o modelo de desenvolvimento de *software* incremental resulta da junção entre o processo de desenvolvimento linear e paralelo, onde cada fase linear resulta na entrega de um incremento do software (Figura 5).

Ainda segundo Pressman 2016, este modelo de desenvolvimento é ideal para equipes com número de desenvolvedores reduzido, focando principalmente no desenvolvimento do núcleo do produto (parte principal com maior utilização pelos usuários).

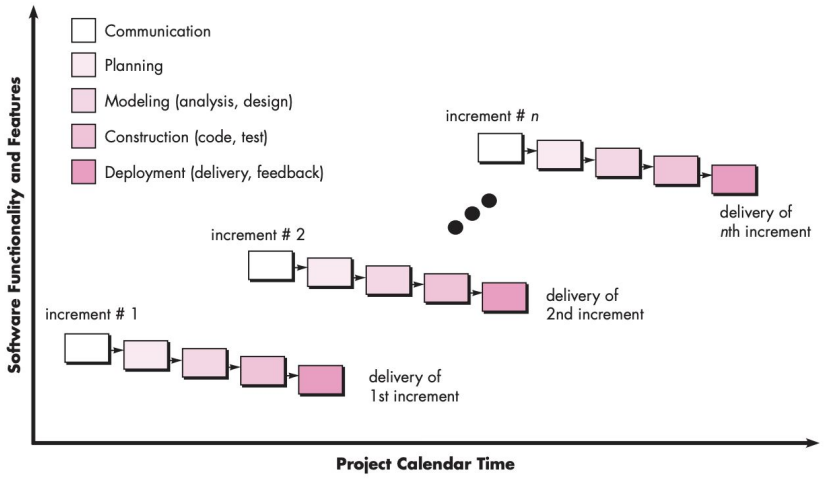

Figura 5 . Diagrama do modelo de desenvolvimento incremental [PRESSMAN 2016]

3.7 GraphQL *Query Language*

A especificação do GraphQL foi desenvolvida pelo Facebook Inc. em 2012 e liberada para uso do público geral em 2015 [Code 2018].

GraphQL é uma linguagem de consulta (*query language*) que possibilita desenvolver APIs (*Application Programming Interfaces*) que sirvam dados de acordo com a necessidade da aplicação cliente, sem realizar *over fetching* (quando uma API retorna mais dados do que será realmente utilizado pelo cliente), reduzindo a sobrecarga na transmissão de dados pela rede, aprimorando, especialmente, a experiência dos usuários de dispositivos móveis [GraphQL 2018]. A Figura 6 apresenta um exemplo de query e resposta em GraphQL.

```
{
  user(id: 4802170) {
    id
    name
    isViewerFriend
    profilePicture(size: 50)  {
      uri
      width
      height
    }
    friendConnection(first: 5) {
      totalCount
      friends {
        id
        name
      }
    }
  }
}
```

```
{
  "data": {
    "user": {
      "id": "4802170",
      "name": "Lee Byron",
      "isViewerFriend": true,
      "profilePicture": {
        "uri": "cdn://pic/4802170/50",
        "width": 50,
        "height": 50
      },
      "friendConnection": {
        "totalCount": 13,
        "friends": [
          {
            "id": "305249",
            "name": "Stephen Schwink"
          },
          {
            "id": "3108935",
            "name": "Nathaniel Roman"
```

Figura 6 . Exemplo de query (esquerda) e resposta (direita) em GraphQL [GraphQL 2018]

3.8 Web Scraping / Web Crawling

Segundo Mitchell 2018, a coleta automática de dados da Internet, utilizando sistemas automatizados que interagem com APIs ou aplicações *Web* (simulando a ação de um usuário comum) é definida como *Web Scraping* (ou *Web Crawling*, quando se tratando de sistemas que interagem com diversas páginas de um *website*).

Web crawlers são utilizados, praticamente, desde o início da operação da Internet [MITCHELL 2018], uma vez que permitem a coleta de dados que não são estruturados, não possuem API pública acessível ou, em último caso, dados que estão protegidos da publicação massiva.

4. Metodologia

O desenvolvimento do projeto iniciou-se pela análise de requisitos da aplicação, para entender com possíveis usuários quais seriam as funcionalidades mais importantes a serem implementadas. Foi desenvolvido um formulário on-line (Figura 7), distribuído para uma amostra de 20 (vinte) potenciais usuários, tendo recebido resposta de 12 (doze) usuários - 60% da amostra total.

No formulário, os potenciais usuários foram questionados quanto a possíveis funcionalidades da aplicação, preferências de interfaces, se já haviam acessado aplicações parecidas e, por fim, se era possível prover informações adicionais caso o usuário tivesse alguma ideia adicional.

Nos resultados apurados do formulário, 7 usuários afirmaram já ter acessado aplicações semelhantes anteriormente, correspondendo a 58% dos usuários respondentes. Os usuários também foram perguntados se há interesse em utilizar este tipo de aplicação. Foi apresentada uma escala de 0 a 10, sendo 0, "não tenho interessante", e 10 "tenho muito interesse". Considerando os 12 usuários respondentes, a média de interesse registrada foi de 8.75, demonstrando um alto interesse dos usuários neste tipo de aplicação.

Para a definição do *layout* da aplicação, visando definir a forma que o usuário realizaria a pesquisa dos dados, 8 (oito) usuários, que correspondem a 66% dos usuários que responderam ao questionário, escolheram como melhor forma de pesquisa a utilização do nome da empresa em uma barra de busca, apresentando infográficos como resultado da pesquisa por empresa.

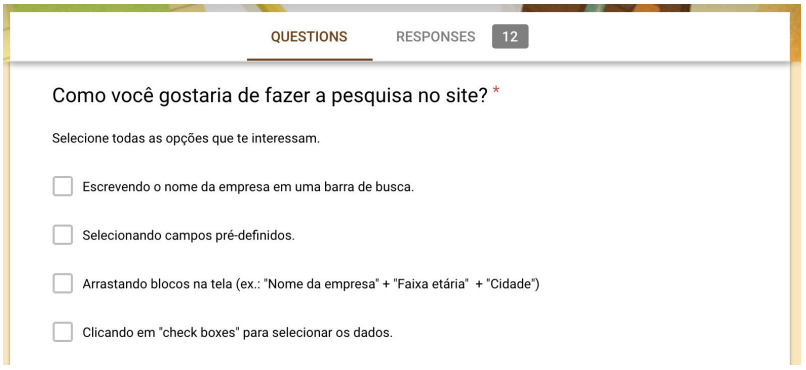

Figura 7 . Exemplos de questões utilizadas no formulário de requisitos

4.1 Desenvolvimento do BigPy

Uma vez definidos os requisitos da aplicação, iniciou-se o processo de desenvolvimento da aplicação, seguindo o modelo de desenvolvimento incremental. O desenvolvimento iniciou-se pela definição da arquitetura da aplicação, seguido pela definição das tecnologias que seriam utilizadas.

Na fase de desenvolvimento do código fonte, iniciou-se o processo pelos testes unitários, seguindo as diretivas do desenvolvimento orientado a testes. Todos os testes unitários era executados a cada nova funcionalidade desenvolvida, visando que todos os testes fossem bem sucedidos ou, quando necessário, era realizado o refatoramento do código.

A ideia principal do BigPy é se estabelecer como uma aplicação de análise e visualização de dados flexível, que permita a integração de diferentes fontes de dados de reclamações de consumidores, que não seja dependente de sua estrutura dos dados.

Nesse contexto, a aplicação foi desenvolvida de forma que houvesse o mínimo acoplamento entre as diferentes partes da aplicação (banco de dados, aplicação no servidor e aplicação cliente). Este modelo de aplicação dará suporte à entrada de dados de fontes diferentes do que a utilizada no desenvolvimento inicial deste projeto. Cada micro-serviço foi desenvolvido de forma a permitir que alterações fossem realizadas em cada parte da aplicação sem afetar as demais.

Outra vantagem encontrada no modelo de arquitetura proposto, é a flexibilidade para construir agregações de dados com a API GraphQL, utilizando uma API singular que será consumida pela aplicação *front-end*, reduzindo a quantidade de código para ser mantido.

4.2 Arquitetura

O desenvolvimento deste trabalho foi baseado na arquitetura de Micro-serviços (*Microservices Architecture*), onde cada serviço que compõe a aplicação foi desenvolvido separadamente e funciona independentemente dos demais [NEWMAN 2015]. Com esta perspectiva, a arquitetura final do BigPy pode ser observada na Figura 8, com as explicações detalhadas de cada módulo da aplicação sendo realizadas nos itens seguintes.

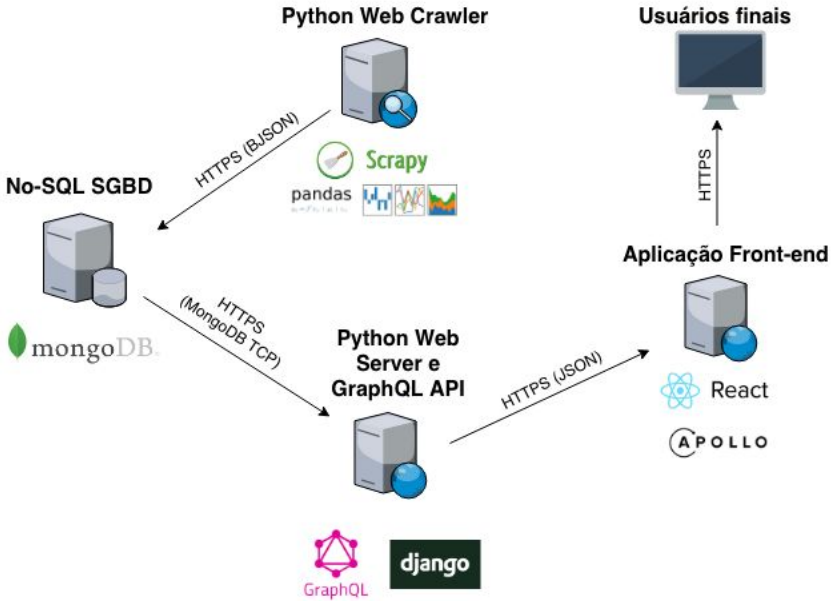

Figura 8. Arquitetura da aplicação

4.3 Python Web Crawler (*scraping*)

Para a captura dos dados do portal Consumidor.gov.br foram desenvolvidos dois *Web Crawlers*, utilizando a linguagem Python [Python 2018], em sua versão 3.6, com o *framework* Scrapy [Scrapy 2018], em sua versão 1.5.

A escolha da linguagem Python se deu devido à sua grande versatilidade para o desenvolvimento de diversos tipos de aplicações (desde simples *scripts* até aplicações *web* completas), o que permitiu utilizar a mesma linguagem em vários módulos da arquitetura proposta neste trabalho. Além disso, a linguagem possui diversas bibliotecas para o tratamento, manipulação e análise de dados [MCKINNEY 2017].

O *framework* Scrapy foi escolhido por ser especializado na construção de *crawlers* preparados para serem escaláveis, possui uma ampla comunidade que mantém o *framework*, e possibilita construir *web crawlers* modulares, que são capazes de percorrer páginas HTML que não possuam uma estrutura bem definida [KOUZIS-LOUKAS 2016].

Para realizar o tratamento, estruturação e formatação dos dados durante a captura nos *web crawlers*, foi utilizada a biblioteca Pandas [Pandas 2018]. Esta biblioteca é a principal ferramenta em Python para manipulação de grandes grandes conjuntos de dados, possuindo diversas estruturas de dados personalizadas, que possibilitam o tratamento dos dados de forma ágil [MCKINNEY 2017].

O primeiro *web crawler* é responsável por capturar os dados gerais referentes às reclamações dos consumidores. O segundo *web crawler* é responsável por obter as reclamações reais dos consumidores, com os textos originais reportados, bem como as respostas dadas pelas empresas. A Figura 9 apresenta um exemplo dos dados obtidos pelo *web crawler* de captura dos dados gerais.

```
{
  "_id":"5be71786437a86218e2f8427",
  "region":"N",
  "state":"PA",
  "city":"Belém",
  "gender":"F",
  "age_range":"41-50",
  "conclusion_date":"2018-10-23T00:00:00.000Z",
  "days_to_reply":10,
  "company_name":"Banco Santander",
  "market_segment":"Bancos, Financeiras e Administradoras de Cartão",
  "problem_reported":"Negativação indevida - desconhece motivo e/ou fornecedor",
  "company_replied":true,
  "customer_rating":4
}
```


Figura 9. Dados gerais da reclamação

Os dados gerais da reclamação compõe-se primariamente de registros sobre características específicas (metadados) da reclamação recebida, como, por exemplo:

- Região, estado e cidade onde a reclamação se originou;
- Faixa etária do consumidor que registrou a reclamação;
- Data de conclusão do registro;
- Intervalo de dias para que a empresa respondesse à reclamação;
- Nome da empresa;
- Segmento de mercado da empresa;
- Categoria do problema reportado;
- Indicação se houve ou não resposta da empresa;
- Nota dada pelo consumidor à resposta recebida da empresa.

A Figura 10, por sua vez, apresenta um exemplo dos dados obtidos pelo *web crawler* de captura dos dados da reclamação real do consumidor.


```
{
  "_id":"5b9d7803437a8657ebe56465",
  "company_name":"SKY",
  "user_report":"Sou cliente de vocês ha muito tempo, porém nos últimos
  "company_response":"Em atenção à reclamação registrada sob o protocolo
  "user_feedback":"<não há comentários do consumidor>",
  "user_rating":5,
  "date":1536807600000,
  "city":"Suzano",
  "state":"SP"
}
```


Figura 10. Reclamação do consumidor e resposta da empresa relacionada

Os dados da reclamação possuem a seguinte estrutura:

- Nome da empresa envolvida;
- Texto original da reclamação reportada pelo consumidor;
- Resposta oferecida pela empresa;
- *Feedback* do consumidor sobre a resposta prestada pela empresa;
- Avaliação do consumidor (nota de 1 a 5);
- Data da inclusão da reclamação;
- Cidade e estado onde a reclamação foi originada.

Quando os *web crawlers* são executados é enviada uma requisição HTTP para a página do Consumidor.gov.br, simulando exatamente as ações que um usuário comum executaria, de forma que os dados possam ser obtidos de forma automatizada. Uma vez que a resposta é obtida pelo *web crawler*, os dados são processados e formatados pela biblioteca Pandas. Nesta etapa, os dados são tratados, removendo-se entradas inválidas (nulas), convertendo caracteres especiais para o formato padrão UTF-8 [W3Schools 2018] e ajustando o formato dos dados para o formato esperado (conforme demonstrado nas Figuras 9 e 10). Finalmente, os documentos gerados são enviados para o banco de dados não-relacional MongoDB, no formato BJSON (formato específico do MongoDB, bastante similar ao formato JSON original).

4.4 Banco de dados (*No-SQL* SBGD)

Dada a natureza do dados coletados pelos *Web crawlers*, que não apresenta estrutura definida para todos os registros coletados, e a necessidade de escalabilidade da aplicação [BRADSHAW 2018], visando reduzir o tempo necessário para gerar os gráficos de análise de dados, este projeto utiliza o MongoDB como Sistema Gerenciador de Banco de Dados (SGBD).

A Figura 11 apresenta um recorte do esquema MongoDB utilizado neste trabalho:

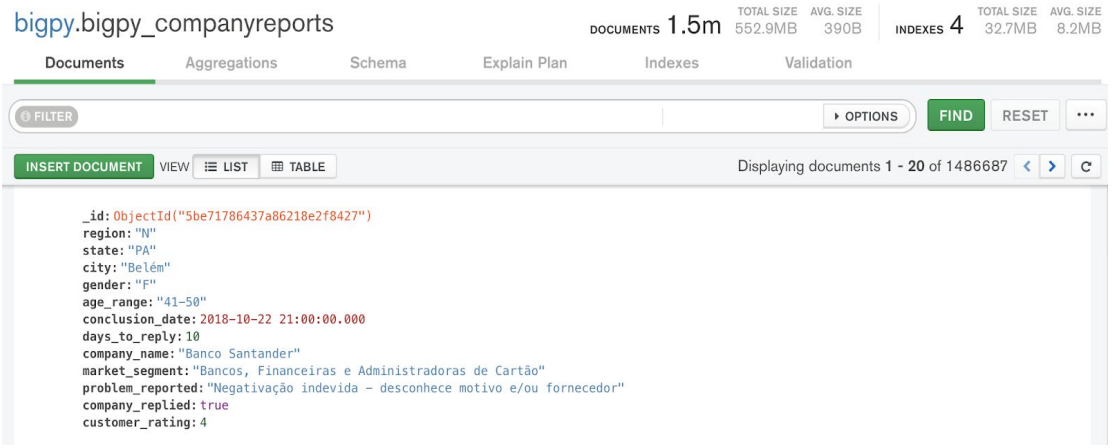

Figura 11. Visualização do banco de dados

Foram coletados pelos *Web crawlers*, aproximadamente, 1.5 milhão de documentos (reclamações de consumidores) do portal Consumidor.gov.br, resultando em 552.9 megabytes de espaço utilizado em armazenamento.

O banco de dados foi hospedado em um servidor *Cloud*, específico para MongoDB, sendo que a conexão ao banco é realizada por meio de requisições TCP/IP, com protocolo próprio do SGBD.

4.5 Python Web Server e API GraphQL

O *web server*, responsável por servir a API GraphQL foi implementado utilizando a linguagem Python [Python 2018], em sua versão 3.6, com o framework Django [Django 2018], em sua versão 2.1.1 e a biblioteca de implementação GraphQL em Python, chamada Graphene [Graphene 2018], em sua versão 2.1.3.

Assim como abordado no item 4.3, a escolha da linguagem Python para implementação da API se deu devido a flexibilidade dessa linguagem para o desenvolvimento de diferentes tipos de aplicações. Somando-se à isto, visando manter a concisão entre os micro-serviços da arquitetura do projeto, a adoção de uma mesma linguagem em diferentes partes da aplicação facilita o processo de desenvolvimento e testes.

A escolha do *framework* Django se deu por uma série de fatores, especialmente pela sua boa integração com o sistema de banco de dados escolhido para o projeto (MongoDB). Além disso, outros fatores foram considerados, estando em consonância com as vantagens do *framework* Django, conforme descritas por [DAUZON 2016]:

- O *framework* é totalmente customizável, sendo possível adicionar módulos externos e/ou realizar a sobrescrita de módulos existentes;
- Utilizando Django é possível se beneficiar do grande ecossistema de bibliotecas da linguagem Python;
- O *framework* respeita o princípio DRY (*Don't Repeat Yourself*), o que significa manter o código simples, evitando ter trechos similares de código repetidos em diferentes partes da aplicação;
- Django provê boa eficiência na execução de testes unitários;
- O *framework* possui uma vasta comunidade, facilitando na resolução de possíveis erros e também provendo as melhores práticas de desenvolvimento com o *framework*.

A biblioteca Graphene [Graphene 2018] foi escolhida por ser a implementação oficial de GraphQL para Python, tendo boa integração com o *framework* Django.

Conforme apresentado na Figura 8, a API GraphQL serve como intermediador entre o banco de dados e a aplicação *front-end*. Neste serviço foram implementados todos os processamentos, agrupamentos e análise de dados requeridas pela aplicação *front-end* para a exibição das informaçõe nos gráficos para os usuários. Dada a flexibilidade proposta por este projeto, para que possam ser integradas diferentes bases de dados, a API possui um único *endpoint* (URL da API que responde a requisições HTTP) que expõe todas as possíveis *queries* GraphQL que podem ser executadas pelo *web server*. A Figura 12 apresenta o esquema GraphQL da aplicação:

```
companiesNames: [String]
totalCompanyReports(companyName: String): Int
totalCompanyReplies(companyName: String): Int
overallCompanyRating(companyName: String): Float
reportsPerDate(companyName: String): [CompanyReportsPerDate]
companiesReportsCount: [CompanyReportsCount]
companyReportsPerRegion(companyName: String): [CompanyReportsPerRegion]
companyReportsPerAge(companyName: String): [CompanyReportsPerAgeRange]
companyRatingPerYear(companyName: String): [CompanyRatingPerYear]
companyReportsPerGender(companyName: String): [CompanyReportsPerGender]
```

Figura 12. Esquema GraphQL implementado pelo web-server

4.6 Aplicação Front-end

O desenvolvimento da aplicação *front-end* para este projeto, teve como foco a construção de uma aplicação que fosse adaptável aos diversos tipos de visualização de dados que seriam implementadas. Para isto, optou-se pelo desenvolvimento de uma aplicação componentizada, que permitisse integrar novos componentes sem a necessidade de reescrever trechos já existentes do código.

Para alcançar este objetivo, além de utilizar as ferramentas padrão para desenvolvimento de aplicações *web* (HTML, CSS e Javascript), adotou-se a biblioteca React JS, em sua versão 16.3 [React 2018]. A escolha dessa biblioteca se deu, principalmente, pelo seu desenvolvimento orientado a componentes, permitindo o reaproveitamento de código e modularização da aplicação. Os componentes criados puderam ser reaproveitados em diferentes partes da aplicação, onde cada um gerencia o seu próprio estado (conjunto de dados relativos ao componente, em tempo de execução) de forma independente.

A Figura 13 apresenta o trecho de código de um dos componentes criados. Este componente, por exemplo, permite a renderização de todos os tipos de gráficos suportados pela biblioteca Chart.JS, não sendo necessário re-escrever a implementação do componente para cada gráfico implementado.

```
class ChartWrapper extends Component {
  componentDidMount() {
    const newChart = new Chart(this.myCanvas, {
      type: this.props.type,
      options: this.props.options,
      data: this.props.data
    })

    this.chart = newChart
  }

  componentWillUnmount() {
    this.chart.destroy()
  }

  render() {
    return (
      <canvas
        ref={canvas => (this.myCanvas = canvas)}
        width={this.props.width}
        height={this.props.height}
      />
    )
  }
}
```

Figura 13. Implementação de um componente em ReactJS

A biblioteca Chart.JS [Charts.JS 2018] foi utilizada para exibir os gráficos dos dados analisados. Essa biblioteca foi escolhida pela sua fácil integração com o React JS, além de ser adaptável para ser visualizada em dispositivos móveis.

Finalmente, para integrar a aplicação *front-end* com a API GraphQL, utilizou-se a biblioteca Apollo [Apollo 2018] para consumir os dados disponibilizados pela API. Esta biblioteca possibilita escrever *queries* na especificação requerida pelo GraphQL, abstraindo a lógica de conexão com a API. A Figura 14 apresenta um exemplo de *query* GraphQL utilizada na aplicação *front-end*:

```graphql
query UserReportFeedQuery($companyName: String!, $cursor: String) {
  allCustomerReports(after: $cursor, first: 5, companyName: $companyName) {
    edges {
      node {
        Id
        companyName
        city
        state
        userReport
        userRating
        userFeedback
        companyResponse
        date
      }
    }
  }
}
```

Figura 14. Query GraphQL implementada pela biblioteca Apollo

4.7 Execução do Desenvolvimento Orientado a Testes (TDD)

Para assegurar a integridade da aplicação e minimizar a ocorrência de falhas durante o processo de desenvolvimento e, principalmente, na utilização da aplicação pelos seus usuários, este projeto utilizou a metodologia TDD. Os casos de testes executados foram desenvolvidos de acordo com as especificações de requisitos esperados da aplicação.

Os testes unitários para o código Javascript foram implementados utilizando a biblioteca de testes Jest [Jest 2018] e React Testing Library [React Testing Library 2018]. Para o código Python, foi utilizada a biblioteca pytest [pytest 2018] e também a ferramenta de testes que já é integrada ao *framework* Django.

A Figura 15 demonstra a saída da execução de testes executada com a biblioteca Jest. Cada componente da aplicação é testado, gerando o resumo final do total de suítes de testes executadas com sucesso ou com falha.

```
 PASS  src/components/App/App.test.js (9.162s)
 PASS  src/components/UserReportFeed/UserReportFeed.test.js (9.325s)
 PASS  src/components/Search/Search.test.js
 PASS  src/components/Navigation/Navigation.test.js
 PASS  src/components/SidebarList/SidebarList.test.js
 PASS  src/components/ErrorMessage/ErrorMessage.test.js
 PASS  src/utils/__tests__/utils.test.js
 PASS  src/components/Insights/Insights.test.js
 PASS  src/components/About/About.test.js
 PASS  src/components/Top10/Top10.test.js
 PASS  src/components/BouncingLoader/BouncingLoader.test.js

Test Suites: 11 passed, 11 total
Tests:       11 passed, 11 total
Snapshots:   10 passed, 10 total
Time:        13.797s
```

Figura 15. Exemplo da execução dos testes unitários

5.0 Resultados

A implementação do BigPy foi realizada com a integração de todos os micro-serviços e os testes unitários, testes de integração e testes de interação da interface. Todos os testes

foram aprovados pelas ferramentas de testes automatizados e as interfaces geradas se comportam conforme definido nos requisitos.

A seguir serão apresentadas as interfaces gráficas geradas pela aplicação. Inicialmente, o usuário é apresentado à tela de pesquisa, onde poderá informar o nome da empresa que se deseja obter os dados (Figura 16).

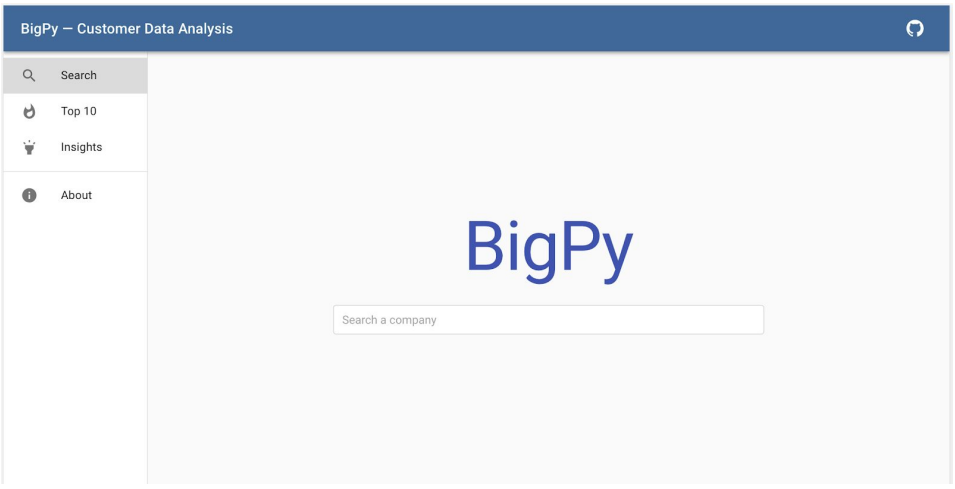

Figura 16. Tela inicial da aplicação com barra de pesquisa de empresas

Após pesquisar pelo nome da empresa, é disponibilizada ao usuário a interface principal do *dashboard* contendo a análise dos dados da empresa (Figura 17). Inicialmente, foram implementados gráficos para visualização da evolução do número de reclamações por ano, evolução na nota dada pelo consumidor ao atendimento da empresa, reclamações por gênero, reclamações por região do país e reclamações por faixa de idade do consumidor.

Figura 17. Dashboard principal com a visualização dos dados

O usuário também poderá visualizar as reclamações postadas pelos consumidores, com a resposta da empresa, *feedback* do consumidor e sua nota final (Figura 18). São disponibilizadas dez reclamações por vez, podendo o usuário carregar mais reclamações no final da página.

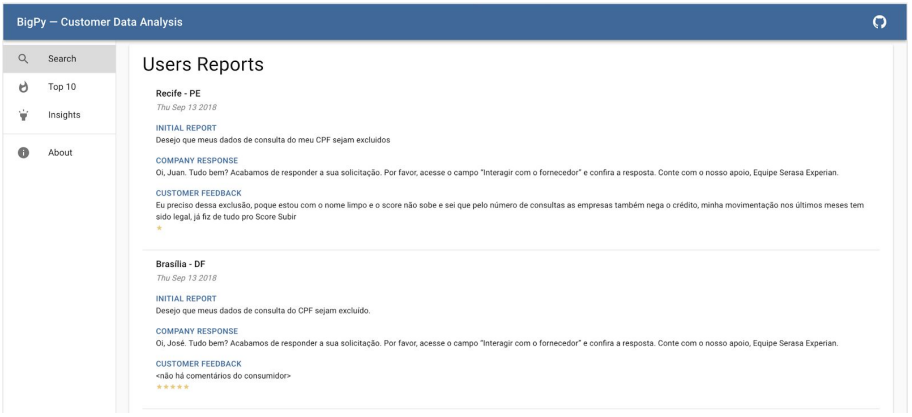

Figura 18. Exemplo da visualização do feed de reclamações dos consumidores

Visando adaptar a aplicação para ser utilizada em diferentes tipos de dispositivos (*laptops, desktops*, dispositivos móveis, etc.), foram utilizadas técnicas de desenvolvimento *web* responsivo, permitindo às diferentes páginas da aplicação se adaptarem aos diferentes tamanhos de telas utilizadas pelos usuários (Figura 19).

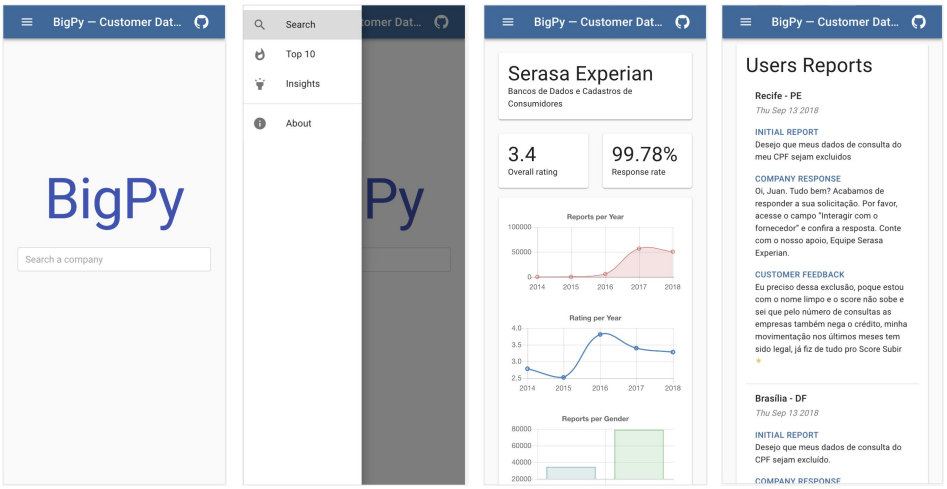

Figura 19. Exemplos de telas adaptadas para dispositivos móveis

Após finalizar o fluxo principal de funcionalidades da aplicação, esta foi apresentada a 7 dos 12 usuários que responderam ao formulário de levantamento de requisitos, correspondendo a 58% do universo de usuários respondentes. Foi demonstrado aos usuários o funcionamento da interface, bem como as visualizações de dados geradas pela aplicação. Todos os usuários aprovaram as funcionalidades de acordo com os requisitos levantados.

6.0 Conclusão e trabalhos futuros

A aplicação desenvolvida foi finalizada com todos os testes implementados. A adoção do desenvolvimento orientado a testes (TDD) possibilitou o desenvolvimento ágil da aplicação, com a minimização da ocorrência de erros durante a implementação das interfaces. Foi possível implementar novas funcionalidades sem impactar funcionalidades implementadas anteriormente, realizando testes de regressão para validar a integridade da aplicação.

A adoção da arquitetura orientada a micro-serviços possibilitou desenvolver cada componente da arquitetura de forma simples, facilitando a integração das partes da arquitetura, com baixo acoplamento entre as partes, provendo assim boa manutenibilidade da aplicação.

Adicionalmente à arquitetura de micro-serviços adotada, a utilização de tecnologias que provêem flexibilidade na captura e análise de dados incentivou a idealização de trabalhos futuros, como por exemplo, a inclusão de novas bases de dados relacionadas à defesa do consumidor. Nesse âmbito, foi feito um estudo preliminar para adicionar dados do Sistema Nacional de Informações de Defesa do Consumidor (Sindec) [Portal Brasileiro de Dados Abertos 2018]. Percebeu-se que, para a integração de mais uma base de dados na aplicação existente, será necessário apenas implementar novas *queries* de agregação em GraphQL, gerando novas visualizações de dados que se façam necessárias. Dessa forma, a aplicação desenvolvida demonstra ter a flexibilidade desejada para analisar diferentes fontes de dados.

Através da arquitetura implementada, percebeu-se também que a aplicação demonstra ser adaptável para sua utilização na coleta, processamento e visualização e outros tipos de dados, não limitados apenas a dados de defesa do consumidor. Pretende-se, em trabalhos futuros, agregar novas fontes de dados em diferentes instâncias da aplicação para validar esta hipótese.

Também como trabalho futuro, pretende-se realizar a implementação de novas funcionalidades na aplicação *front-end*, incluindo: página com lista de empresas com maior número de reclamações, página com *insights* sobre as base de dados capturadas e o *deploy* da aplicação em servidores em nuvem, provendo acesso à aplicação aos usuários finais.

Finalizando este trabalho, espera-se que a aplicação desenvolvida traga aos seus usuários maior entendimento sobre a qualidade dos serviços prestados pelas empresas aos seus consumidores, tornando visível e compreensível à estes dados públicos disponíveis na Internet.

7.0 Referências

Apollo (2018). Apollo GraphQL. `https://www.apollographql.com/`. [Online; acessado em 25 de Setembro de 2018].

BRADSHAW, Shanno; CHODOROW, Kristina. MongoDB: The Definitive Guide: Early Edition. 2019. ed. USA: O'Reilly Media, Inc., 2018. 425 p.

BROWN, Ryan. Django vs Flask vs Pyramid:: Choosing a Python Web Framework. `https://www.airpair.com/python/posts/django-flask-pyramid` [Online; acessado em 18 de Junho 2015].

Chart.js (2018). Chart.js. `https://www.chartjs.org/`. [Online; acessado em 25 de Setembro de 2018].

CGI.br, Comitê Gestor da Internet no Brasil . TIC domicílios e empresas 2013: Pesquisa sobre o uso das tecnologias da informação e comunicação no Brasil. Barbosa, A. F. (ed.). São Paulo: Cetic.br. 2014. 662 p.

Code (2018). Facebook Code. https://code.fb.com/core-data/graphql-a-data-query-language/ [Online; acessado em 12 de Setembro de 2018].

DAUZON, Samuel; BENDORAITIS, Aidas; RAVINDRAN, Arun. Django: Web Development with Python. 1. ed. Birmingham, UK: Packt Publishing, 2016. 717 p.

DESHPANDE, Yogesh et al. Web Engineering. Journal of Web Engineering, [S.l.], 27 nov. 2018. 1, p. 3-17. https://dl.acm.org/citation.cfm?id=2011098 [Online; acessado 18 Junho 2015].

Django (2018). Django Project. https://www.djangoproject.com/. [Online; acessado em 13 de Setembro de [2018].

GIFT, Noah; JONES, Jeremy M.. Python: for Unix and Linux System Administration. Sebastopol: O'reilly, 2008. 433 p.

Graphene (2018). Graphene Python. https://graphene-python.org/. [Online; acessado em 13 de Setembro de [2018].

GraphQL (2018). Facebook Code. https://graphql.org/ [Online; acessado em 12 de Setembro de 2018].

Google (2018). Google. https://www.google.com.br. [Online; acessado em 30 de Setembro de 2018].

Jest (2018). Jest. https://jestjs.io/. [Online; acessado em 25 de Setembro de [2018].

KOUZIS-LOUKAS, Dimitrios. Learning Scrapy. 1. ed. United Kingdom: Packt Publishing, 2016. 243 p.

LAGRONE, Benjamin; FRAIN, Ben; FIRDAUS, Thoriq. HTML5 and CSS3: Building Responsive Websites. 1. ed. Birmingham: Packt Publishing, 2016. 728 p.

MCKINNEY, Wes. Python for Data Analysis. 2. ed. Sebastopol: O'Reilly Media, 2017. 511 p.

MINETTO, Elton. Frameworks para Desenvolvimento em PHP. 1. Ed.São Paulo: Novatec Editora, 2007. p. 17-

MITCHELL, Ryan. Web Scraping with Python. 2. ed. USA: O'Reilly Media, Inc., 2018. 231 p.

NEWMAN, Sam. Building Microservices. 1. ed. USA: O'Reilly Media, Inc., 2015. 251 p.

NGINX (2018). NGINX. https://www.nginx.com/. [Online; acessado em 13 de Setembro de [2018].

OLIVEIRA, M. C. F. e Levkowitz, Haim. (2003). From visual data exploration to visual data mining: a survey. IEEE Transactions on Visualization and Computer Graphics, 9(3):378–394.

Pandas (2018). Python Data Analysis Library. https://pandas.pydata.org/. [Online; acessado em 13 de Setembro de [2018].

Portal Brasileiro de Dados Abertos (2018). Cadastro Nacional de Reclamações Fundamentadas (PROCONS - Sindec). http://dados.gov.br/dataset/cadastro-nacional-de-reclamacoes-fundamentadas-procons-sindec1 [Online; acessado em 30 de Outubro de 2018].

PRESSMAN, Roger S.. Engenharia de Software: Uma abordagem profissional. 7. ed. Porto Alegre: McGraw-Hill, 2011. p. 771.

pytest (2018). pytest. https://docs.pytest.org. [Online; acessado em 13 de Setembro de 2018].

Python (2018). Python. https://www.python.org/. [Online; acessado em 13 de Setembro de 2018].

React JS (2018). React JS. https://www.reactjs.org/. [Online; acessado em 25 de Setembro de 2018].

React Testing Library (2018). Jest. http://npm.im/react-testing-library. [Online; acessado em 25 de Setembro de [2018].

Reclame Aqui (2018). Reclame Aqui. https://reclameaqui.com.br. [Online; acessado em 23 de Agosto 2018].

SALE, David. Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing. 1. ed. United Kingdom: John Wiley & Sons, 2014. 211 p.

VOSSEN, Gottfried; HAGEMANN, Stephan. Unleashing Web 2.0: from concepts to creativity. USA: Elsevier, Inc., 2010. 345 p.

W3Schools (2018). W3Schools. https://www.w3schools.com/charsets/ref_html_utf8.asp [Online; acessado em 17 de Setembro de 2018].