**Teste de Software (CE-229), Projeto de Sistemas de Banco de Dados (CE-240) e Tecnologias da Informação (CE-245)**

**Sprint #02**

**TS#01 – Coleta de Dados (via Redes Sociais/API)**

1. **Introdução**

Este documento descreverá sobre o desenvolvimento do código-fonte para realizar a captura de posts no Facebook. Vale ressaltar que que esta rede social, diferentemente do Twitter, não permite mais que sua API recolha posts de acordo com uma palavra-chave selecionada.

A solução encontrada pelo Time Scrum 1 (TS1), por sugestão do General Product Owner, Rafael Shigemura, foi a de selecionar algumas páginas de empresas informativas que pudessem cobrir os mais variados tipos de crise, de acordo com a tabela COBRADE. Assim, foram escolhidas dez páginas, a princípio, para que pudessem ser coletados dados sobre elas e armazenados no Banco de Dados Cassandra.

* 1. **User Story selecionada para este relatório**

Conforme mencionado na seção anterior, a User Story que será descrita neste documento será uma alternativa para coleta de dados feitas através da rede social Facebook, que não pode ser realizada com sucesso na primeira Sprint.

|  |  |  |
| --- | --- | --- |
| ID | User Story | TM Alocados |
| TS01-US01 | **COMO** segmento de coleta de dados,  **EU** **DESEJO** comunicar com a API de redes sociais - Facebook,  **PARA** obter informações relacionadas a emergências e grandes eventos. | Carlos Dória (Dev)  Fellipe Rey (Dev)  Fernando Trindade (Tester) |

1. **Desenvolvimento da aplicação**

Inicialmente a aplicação foi desenvolvida na linguagem Python pelo membro Carlos Dórios. Entretanto, devido a algumas dificuldades em realizar a conexão de banco de dados utilizando o Python, o time em conjunto decidiu que deixaria a linguagem de desenvolvimento opcional para o time.

Assim, a coleta de dados, anteriormente feita em Python, passou para o Java. O motivo da escolha foi o conhecimento e familiaridade de alguns membros com a linguagem, podendo assim aumentar a produtividade e diminuir o número de impedimentos durante o desenvolvimento.

**2.1 Páginas Selecionadas**

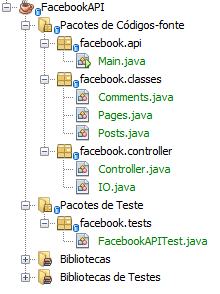
Foram selecionadas algumas páginas do Facebook para que os dados pudessem ser coletados e analisados. A escolha destas páginas se deu ao fato de transmitirem notícias com veracidade, sendo algumas delas os canais de comunicação oficiais de suas áreas.

A princípio, foram selecionados 10 (dez) websites que para ajudar na coleta de dados para prever e acompanhar uma crise. Este número pode aumentar, provendo assim a escalabilidade do sistema e garantindo que o código executará em tempo aceitável.

As páginas selecionadas foram: [Climatempo](https://www.facebook.com/Climatempo.Meteorologia/), [Desastres – Brasil](https://www.facebook.com/DesastresBrasil), [OVALE](https://www.facebook.com/jornalovale/), [Folha de S.Paulo](https://www.facebook.com/folhadesp), [Câmara Municipal de São José dos Campos](https://www.facebook.com/camarasjc), [G1 – O Portal de Notícias da Globo](https://www.facebook.com/g1), [UOL Notícias](https://www.facebook.com/UOLNoticias), [Estadão](https://www.facebook.com/estadao), [Jornal da Band](https://www.facebook.com/jornaldaband) e [CBN](https://www.facebook.com/radiocbn). É possível acessar cada uma destas páginas clicando no Hiperlink associado a cada uma delas.

**2.2 Estrutura do Código-fonte**

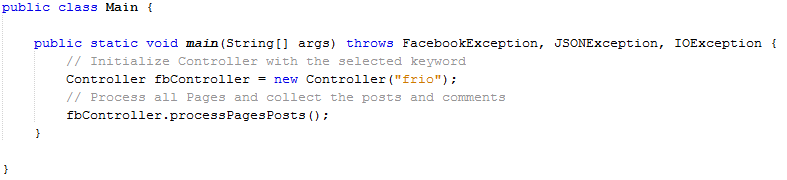
O código fonte está distribuído entre três pacotes, sendo eles: f*acebook.api*, f*acebook.classes* e f*acebook.controller*. Também há um quarto pacote, o *facebook.tests* que se dedica a executar os testes unitários. A imagem abaixo ilustra como os pacotes estão estruturados no projeto da Integrated Development Environment (IDE) Netbeans.

**

*Figura 1 – Estrutura dos pacotes e classes no projeto Netbeans*

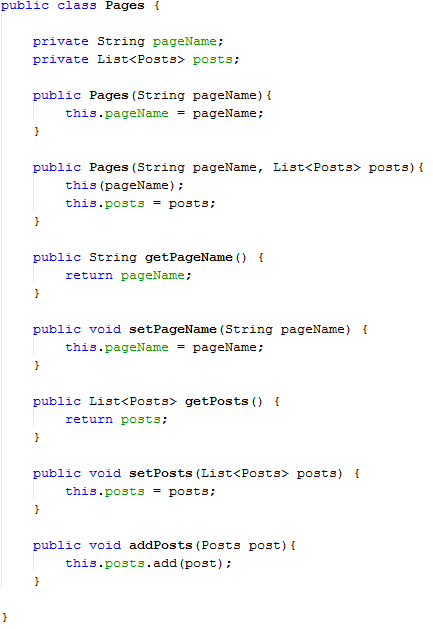
Abaixo, serão descritos rapidamente as classes associadas a cada um destes pacotes e suas funcionalidades.

**facebook.api**: Este pacote contém a classe Main, que é responsável por instanciar o controlador e executar o método de captura de posts de acordo com a palavra-chave selecionada.

**

*Figura 2 - Classe Main.java*

**facebook.classes**: O Facebook segue uma estrutura que pode ser assim descrita: Para cada página, existem um número *n* (0 ≤ n ≤ ∞) de publicações (posts). Cada post contém um número *m* (0 ≤ m ≤ ∞) de comentários. Então, para que as estruturas de nossas classes tenham coerência com o Facebook, foram também criadas três classes principais: *Pages*, *Posts* e *Comments*, referentes às Páginas, que contém as publicações que por sua vez podem conter comentários.



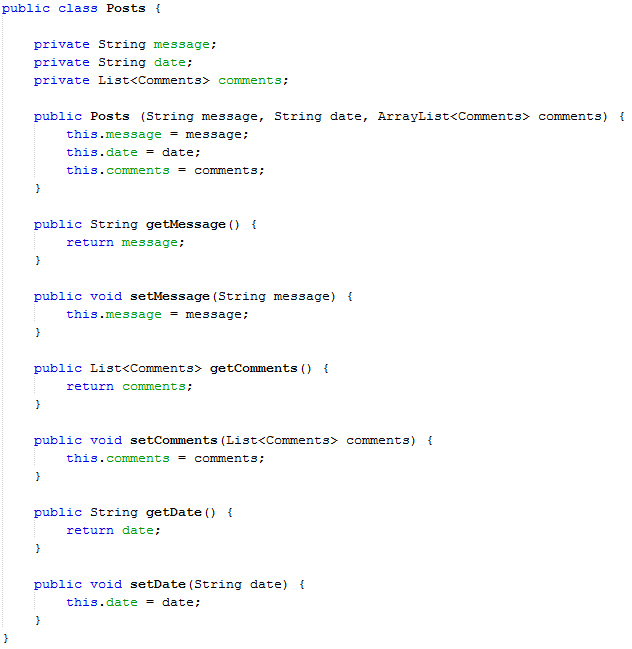
*Figura 3 – Estrutura da classe Pages*

A classe *Pages*, exibida acima e conforme já explicada no parágrafo anterior, tem um nome e uma lista de Posts associado a ela. Também contém dois construtores e um conjunto de setters e getters para cada uma das variáveis declaradas para esta classe.

Para a lista de Posts, foi criada uma nova classe para representar todas as possíveis publicações que uma página pode ter. É possível, para estas publicações, pegar o texto de cada uma delas, bem como a data em que foi publicada e os comentários associados.

Uma das limitações encontradas pela API do Facebook é que, conforme esta está em um período de transformação e a tendência é que boa parte das coletas de dados sejam descontinuadas, algumas informações que poderiam ser de grande ajuda para o time de processamento de dados não puderam ser coletadas.

Destacamos principalmente para a parte de posts que, caso alguma foto ou vídeo sejam publicados sem a ausência de um texto, não é possível realizar a captura desta mídia. Assim, não é possível capturar vídeos e fotos e, se estes forem relacionados a alguma crise em andamento, esta informação não tem como ser coletada.

**

*Figura 4 – Estrutura da classe Posts*

Outro problema também notado foi em questão ao referenciamento geográfico da notícia. Todas as páginas coletadas se referem a portais de notícias, então mesmo que pudesse ser coletado o local de origem da publicação, não necessariamente ele refletiria no local exato em que a crise está ocorrendo.

Outro fator importante a ser considerado diz respeito a abreviações e alcunhas para lugar. Por exemplo, a cidade de São Paulo pode ser descrita de diversas maneiras: São Paulo, S. Paulo, SP, Capital Paulista, Terra da Garoa, etc. É impossível realizar uma rotina que determine todos os possíveis nomes para todas as cidades, estados e regiões brasileiras, tendo em vista que o projeto BD-ITAC deverá ser geral.

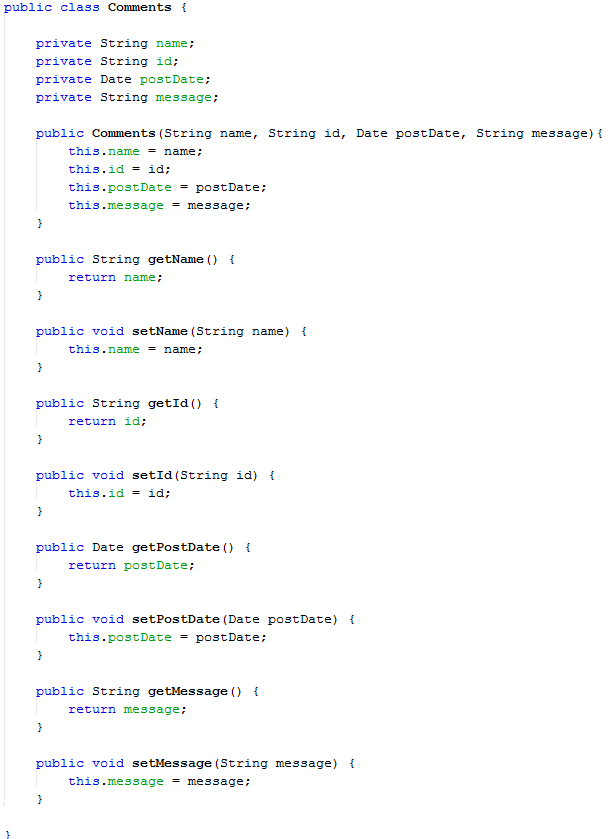
Já a classe *Comments* é responsável por armazenar cada comentário feito em cada uma das publicações das 10 páginas analisadas. É possível obter algumas informações sobre os comentários, tais como o Nome e ID da pessoa que realizou o comentário, a data de postagem e o texto da mensagem publicado pelo usuário.

Também é necessário ressaltar que os comentários, assim como as publicações, também sofrem dos mesmos problemas. Não é possível realizar um referenciamento geográfico para determinar de qual cidade/região o usuário que está realizando o comentário é, bem como não é possível obter os arquivos em formato de mídia (fotos e vídeos) caso ele poste algum.

Do ponto de vista do projeto, esta perda é enorme, uma vez que o usuário pode comentar em uma publicação enviando fotos ou vídeos de acontecimentos relacionados à crise, que podem ser processadas pelo time 5.

Também não é possível coletar o número de *likes* (nem para as publicações quanto para os comentários). Embora o número de *likes* não altere a quantidade de informações obtidas, ele pode ser um grande fato qualitativo: Presume-se que os comentários que possuem um maior número de *likes* são aqueles que possuem uma maior aceitação do público ou que os usuários concordem com o texto publicado, sugerindo que este possua talvez um maior grau de importância que outros comentários.

Caso o número de *likes* pudesse ser obtido, ele poderia auxiliar o time que realizará o processamento de dados a realizar uma ordenação baseada no número de *likes*, assim os comentários mais relevantes poderiam ser processados primeiro, o que acarretaria em uma diminuição do tempo total de processamento dos dados no Spark.

**

*Figura 5 – Estrutura da Classe Comments*

**facebook.controller**: Neste pacote, encontramos duas classes, que são as responsáveis por executar a lógica desta aplicação, sendo elas as classes Controller.java e IO.Java. Abaixo descreveremos melhor sobre cada uma destas classes e sua importância dentro desta aplicação.

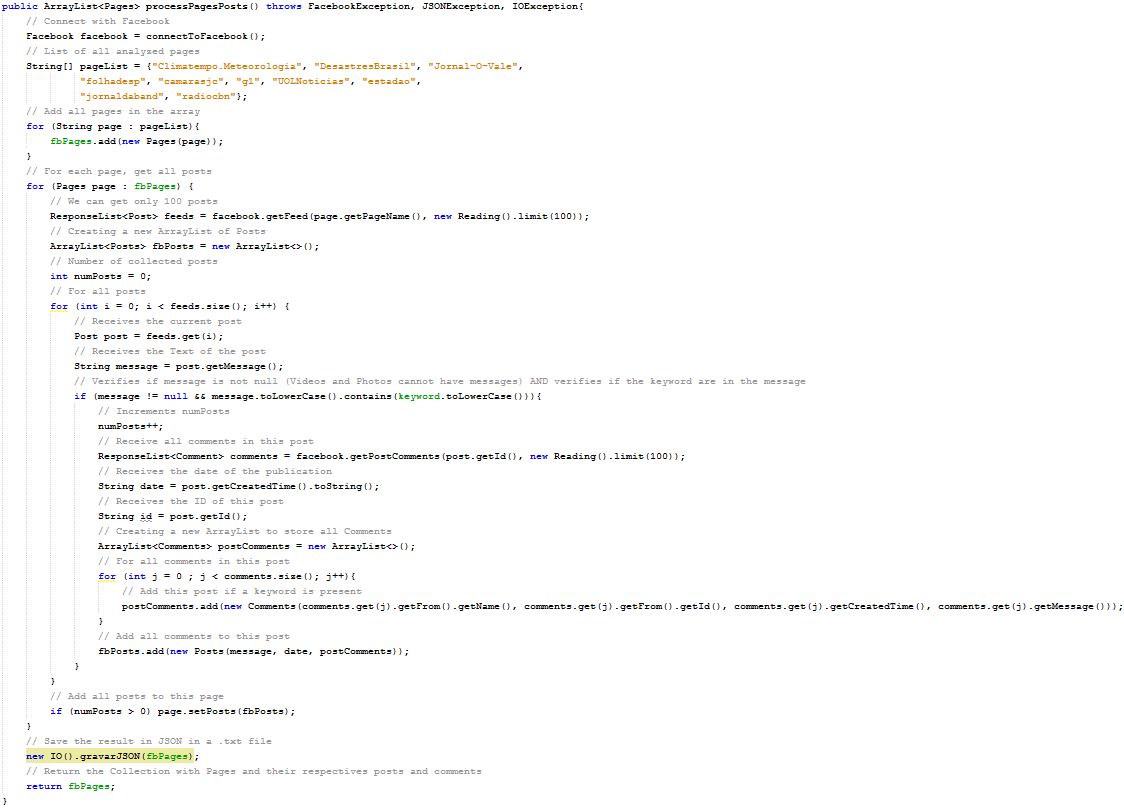


Figura 6 – Estrutura da Classe Controller

Nesta classe, temos dois métodos importantes, sendo um deles responsável por realizar a conexão com o Facebook através do App ID, App Secret e do Token de acesso, permitindo assim que as coletas de dados sejam feitas. O outro método é onde reside a lógica do sistema em si. É nele em que o processamento é feito: A partir de cada uma das páginas selecionadas, ele obtém as últimas 100 publicações da página e verifica se, para cada publicação, existe no texto a palavra-chave que estamos buscando.

Existe, além das limitações já explicadas, outra dificuldade em realizar a coleta de dados no Facebook. Se não definirmos um número de dados a serem coletados, por padrão o Facebook nos dá somente as últimas 25 publicações da página.

Este número pode ser estendido, entretanto, há um limite, que é o de poder obter apenas as últimas 100 últimas publicações da página. Isto pode ser um empecilho na hora de realizar uma previsão de crise, porque não há como constatar se esta crise já ocorreu em determinada época em anos anteriores.

Após realizar a coleta dos dados, eles serão salvos em uma lista de Páginas que por sua vez chamará um método para transformar estes dados para o formato JSON e posteriormente salvar em um arquivo texto.

A transformação em JSON segue a ordem natural de como os dados estão inseridos: Temos um vetor de páginas, onde cada posição deste vetor contém o nome da página junto com um vetor de todas as publicações obtidas. Cada posição do vetor de publicações possui o texto da publicação, data da postagem e outro vetor, este que por sua vez armazena todos os comentários relacionados à publicação. O vetor de comentários possui, em cada posição, as informações de nome e ID referente ao usuário que fez a publicação, o texto com o horário em que foi publicado.

O arquivo é salvo provisoriamente em um arquivo texto contendo em seu conteúdo os dados em formato JSON. Assim que o Time 6, responsável pelo Banco de Dados do projeto, liberar o Cassandra, será possível salvar este resultado diretamente no banco de dados não relacional.

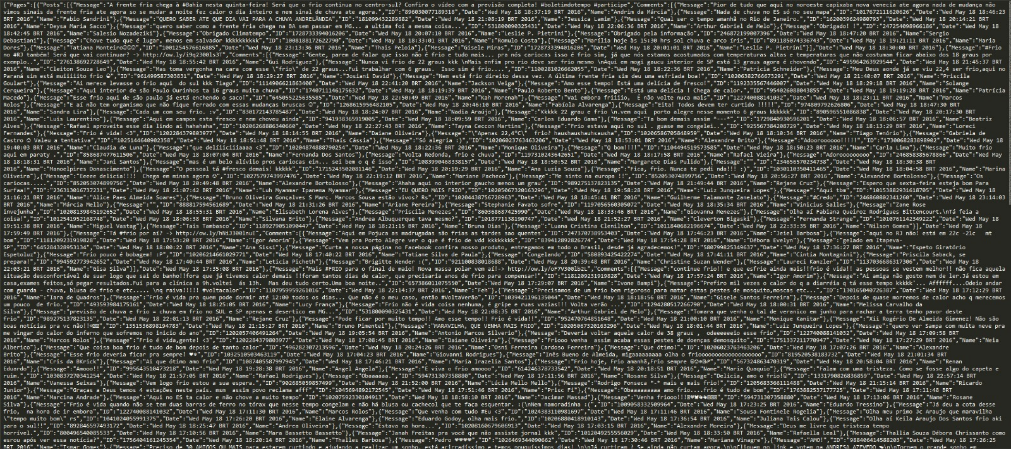


Figura 7 – Saída em JSON da busca pela palavra-chave “frio”

**

*Figura 8 – Estrutura da Classe IO*

**2.3 Testes de Software Realizados**

Para esta aplicação, foram realizados testes de baixa branca e testes de caixa preta. Abaixo será descrito um pouco sobre o pacote de testes e sua execução.

**Pacote “facebook.tests”**: Neste pacote está a classe FacebookTesteAPI. Ela é responsável por armazenar e posteriormente executar os testes unitários desenvolvidos. Como nosso time é responsável apenas pela coleta de dados e o processamento feito é apenas na filtragem dos dados, não foi possível criar vários casos de teste para a aplicação, entretanto, os dois casos criados foram suficientes para desenvolver e refatorar o código de forma que qualquer possível falha de programação fosse detectado e posteriormente, corrigido.

O primeiro teste realizado foi a busca pela palavra-chave “chuva” nas páginas pré-selecionadas do Facebook. Através do primeiro teste pudemos provar que os resultados coletados pela aplicação não eram nulos e que as publicações continham a palavra-chave chuva.

O segundo teste, que é parecido com o primeiro, se concentra em provar que, para uma palavra-chave não existente em nosso dicionário (no caso, a palavra utilizada foi “qpeiabe”, nenhuma publicação deveria ser encontrada. Portanto, seguindo a lógica do teste unitário, também conseguimos provar que para palavras-chaves inexistentes, não haveria nenhuma publicação.

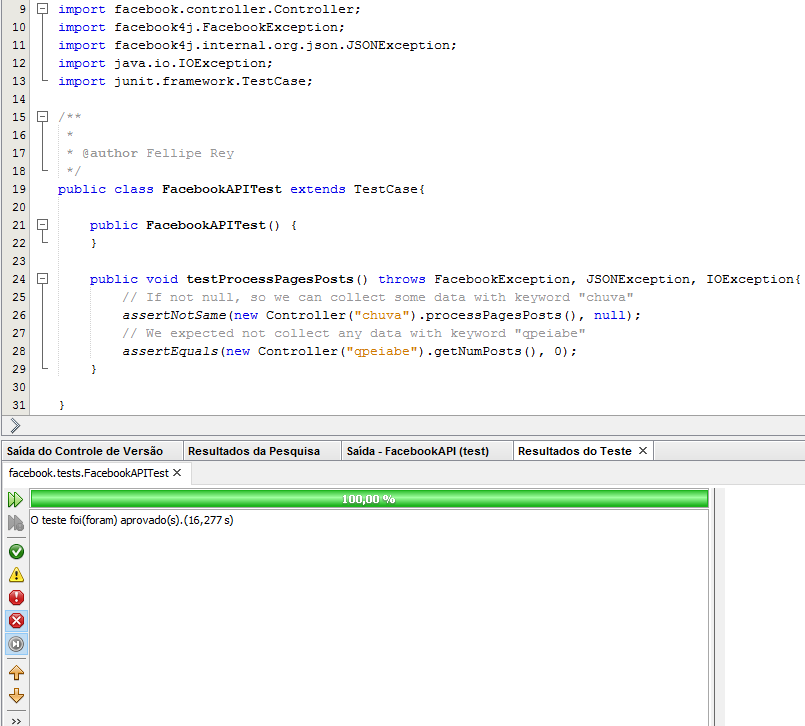


Figura 8 – Estrutura da classe FacebookAPITest e sua execução

Por não achar necessário replicar dados neste documento, decidimos referenciar que os testes de caixa preta que foram desenhados para que o ATDD fosse realizado está disponível no repositório do Time Scrum 1 e também no Github.

1. **Referências**

**Implementando Testes Unitários em Java – Parte 1**. Disponível em: <http://www.devmedia.com.br/junit-implementando-testes-unitarios-em-java-parte-i/1432>. Acessado em: 19/05/2016

**Implementando Testes Unitários em Java – Parte 2**. Disponível em: < http://www.devmedia.com.br/junit-implementando-testes-unitarios-em-java-parte-ii/1549>. Acessado em: 19/05/2016

**Facebook4J**: A most easily usable Facebook API wrapper in Java. Disponível em: <http://www.facebook4j.org>. Acessado em: 19/05/2016