Inbenta Developer Challenge

Developer:

- Name: José María Sánchez Salas
- Email: josemaria.sanchezsalas@gmail.com
- GitHub: https://github.com/jmssalas
- GitHub of the challenge: https://github.com/jmssalas/inbenta-developer-challenge

Date: April 20, 2020

Table of contents

1. Introduction

This document contains the explanation of my solution for the Inbenta Developer Challenge. The content of this document is structured in the following sections:

1. Introduction: It is this section and contains the introduction of this document.
2. YodaBot: This section contains the solution of the YodaBot challenge.
3. Text Analyzer: This section contains the solution of the Text Analyzer challenge.

All files used to do this challenge is stored in the following GitHub repository: https://github.com/jmssalas/inbenta-developer-challenge

2. YodaBot

First at all, this challenge is for a full-stack developer. However, I'm a backend developer and I just developed the backend part of this challenge. I told Ángela (selec.angela@gmail.com) that if there was a problem with that and she told me that there was not, so in this section I'm going to explain my backend solution.

The solution of this challenge is in the yodabot.zip file attached with this document and it contains:

- **The code of the server side.** It is in the server/ folder. The documentation of the API and environment variables is in the README.md file inside of this folder.
- **The Postman files to test the API.** These files are in the postman/ foder and they are the collection and the environment file. The collection contains two folders:
    - Laravel API: To test my API.
    - Inbenta API: To test the Inbenta API.
    - Poke API: To test the Poke API.

The server is deployed in AWS and the public server's URL is:
http://inbentachallenge-env.eba-xmutg9av.eu-central-1.elasticbeanstalk.com/

Due to the SWAPI project is no longer maintained, Verónica (vbarea@inbenta.com) told me that I could use the PokeAPI to obtain a list of 5 pokemons instead of Star Wars characters and a list of 5 Pokemon locations instead of the Star Wars films. So, I used the PokeAPI to do the points b) and c) of the 5th part of this challenge.

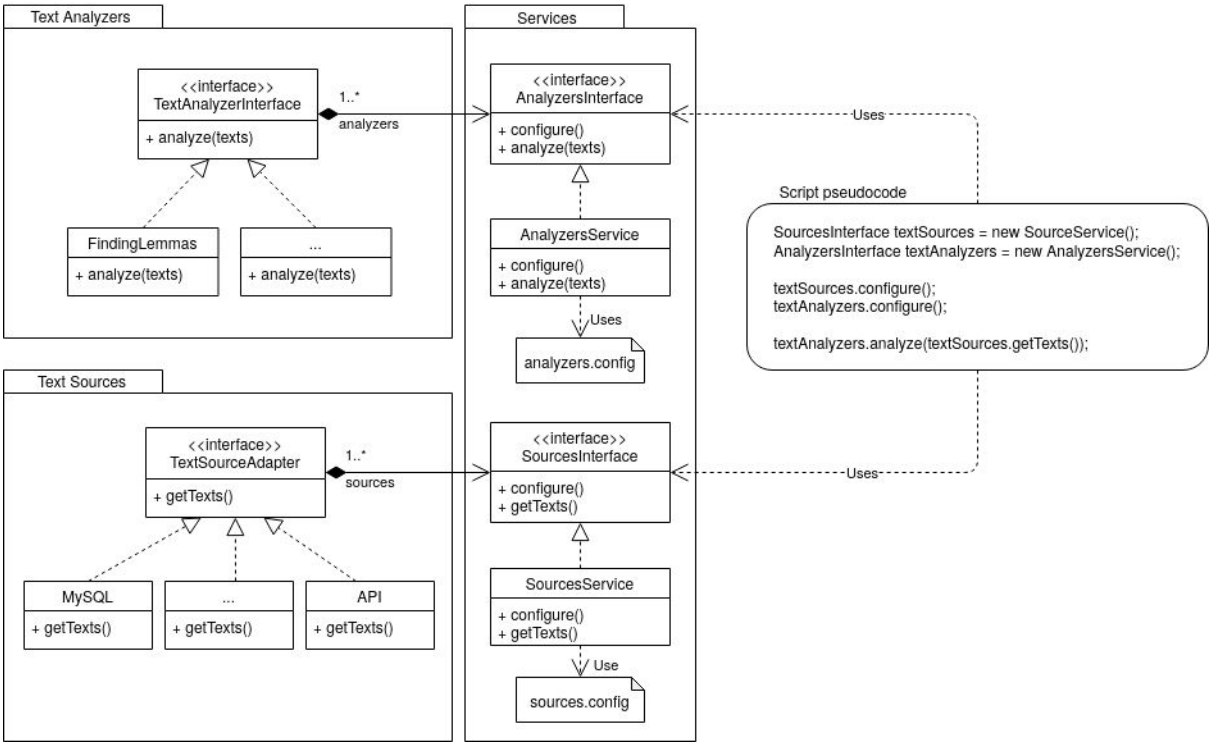
3. Text Analyzer

In this section, I'm going to explain my solution of the Text Analyzer challenge. To do so, this section is divided in two subsections:

- Analysis and design, where I explain the analysis of the challenge and the technical design of my solution.
- Development planning, where I explain the development planning of my solution.

3.1. Analysis and design

The following diagram shows the UML diagram of my solution:



Next, I'm going to explain the meaning of each element in the diagram and what are the relationships between them:

- **Text Analyzers package.** This package contains the interface TextAnalyzerInterface which represents a text analyzer and contains the method analyze(texts) which encapsulates analysis of the texts given as parameter. Each text analyzer has to implement this method according to its algorithm.
- **Text Sources package.** This package contains the interface TextSourceAdapter which represents a text source and contains the method getTexts() which encapsulates the obtaining texts. Each text source has to implement this method according to its text source.
- **Services package.** This package contains two interfaces:

    - AnalyzersInterface which defines the methods to configure what text analyzers will be used to analyze the texts and to execute the analysis of all analyzers.
    - SourcesInterface which defines the methods to configure what text sources will be used to get the texts and to get the texts from all sources.

  The implementations of these interfaces (AnalyzersService and SourcesService) are responsible of their correct configuration and execution (using their own configuration file).
- **Script pseudocode.** This is the pseudocode of the main function of the script, where it specifies which implementations are used for each service interface and then configure and use them.

The **configuration of texts sources and which analysis will be executed** are managed by the configuration files of each service implementation. These configuration files are going to be developed according to the needs of each service. The **class dependencies** are going to be resolved by these configuration files as well.

The following **software patterns** have been used:

- **Adapter Pattern.** This pattern has been used with the text sources, because each text source will have its own mechanism to get the texts and it is necessary to unify them.
- **Strategy Pattern.** This pattern has been used with the text analyzers and the services, because using this pattern the implementation of each text analyzer and each service can be different and the main script will not change.
    - Note: this pattern has been used with the text analyzers because in this solution each algorithm will be developed. If the idea is to use an external tools that have the algorithm already developed, the Adapter Pattern will be the best approach.

With this solution, it is easy to **include both new analysis algorithms and new text sources**, it is only necessary include them in the configuration files and modify the services to use them. The **complexity** of this solution lies in the development of the configuration files and the services to understand these configuration files to manage both text sources and text analyzers. I think the complexity of this solution is medium, considering that the complexity of each text source and analyzer is out of this complexity and is independent.

3.2. Development planning

To break the development process I would divide the responsibilities in three parts with its estimation:

- Text Analyzers:
    - Definition and implementation of AnalyzersService (with its configuration file) (3 days).
    - Implementation of each text analyzers (1 day each).

- Text Sources:

    - Definition and implementation of SourcesService (with its configuration file) (3 days).
    - Implementation of each text source (1 day each).
- Main script:
    - Implementation of the main script (2 hours).
    - Testing with several configurations (6 hours).

One developer would be the responsible of the Text Analyzers part and the other one would be of the Text Sources part. When one of them would have finished, he/she would start with the main script. Once the development would be finished, the main script would be tested by both developers. Supposing one text source and analyzer, the solution would take 5 days: 4 days to develop sources and analyzers and 1 day to develop and test the main script.