UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Project Architecture

Big Data Management

Project Group:
Andrea Armani
Ricardo Holthausen
Jesús Huete
Dimitrios Tsesmelis

Professor(s):
Oscar Romero
Sergi Nadal

Project Period:
Winter Semester 2020

Date of Submission:
June 18th, 2020

Edifici B6 del Campus Nord, C/Jordi Girona, 1-3, 08034 Barcelona

Table of Contents

Table of Contents 2

Brief product description 3

Data sources 3

- Football Statistics 3
- Twitter API 4
- Facebook API 4
- Behavioural Test 5

Data Modeling and Storing 5

Data Exploitation 6

Functional Architecture 7

Tool Selection 8

- Landing Zone 8
- Transformations 8
- Knowledge graph 9
- Formatted Zone 9

Use Cases and Data Flows 9

- Data Ingestion 10
  - SoccerWiki Scraping 10
  - Facebook data scraping 10
- Data Processing/Analysis 10
  - Twitter Data Processing 11
  - CPRD Behavioural Test Data Processing 11

References 11

Appendix 2 13

Brief product description

FreeKick is a scouting tool for football teams which, besides the traditional statistical information of the player's performance, retrieves psychological information from a set of tests we provide within the application, as well as social media from the players (Facebook, Twitter, etc.). The goal of our product is to provide a complete profile of the players that will be used by the football teams to enhance their scouting experience.

Data sources

To provide such information, we need to collect data from several sources, namely:

- Demographic and statistical information from a set of webpages from which we will perform scrapping
- Social media profile activity where we intend to apply NLP algorithms to analyze the emotions from their posts and mentions (anger, anxiety, stress, happiness, etc.)
- Psychological tests result that we intend to provide within the application

Football Statistics

Our product distinguishes the football players in two main categories, namely the **professional** one and the **new talents**. Regarding the first category, the basic player's information is extracted from https://en.soccerwiki.org/wiki.php. Soccer Wiki is an open source soccer orientated wiki made for the fans, by the fans. It is a collaborative database and anyone can create and edit data. This community driven database contains information on players, clubs, stadiums, managers, referees, leagues and other data related to the world of soccer. According to its robots.txt file (https://en.soccerwiki.org/robots.txt) the database can be freely scraped. Our approach is to scrap the content of the database using **Scrapy**. From this source we are extracting the following data for each player:

- Personal information: name, club, age, birthdate, nation, height, weight, overall rating
- Position data: preferred position, preferred foot
- Player's attributes (e.g. Aerial Ability, Aggression, Strength, Corners, Pace)
- Similar players
- Rating history

The available information includes **104,353 players**.
This source is used to initialize our database with the basic information for the players. This means that this task is executed only once (initialization phase). Periodically (once a year), we are conducting an updating phase that is looking for changes (e.g. a player

moved to another team) in the source database and updates the information in our system. The information for the new talents will be similar to the previously collected one and it will be collected directly from the user interface. The volume of this data is expected to be from **hundreds to thousands** of players.

Twitter API

The Twitter streaming API is used to download twitter messages in real-time. It is useful for obtaining a high volume of tweets, or for creating a live feed using a site stream or user stream [1]. Besides the streaming API, the REST API module allows developers to retrieve and generate information with traditional GET and POST requests. Such requests are limited per day (100,000 requests per day). For this project and the data that we are interested in, Twitter4j API service will be used to retrieve user timeline tweets related to the football players.

Data would be received in a pull fashion. Such tweets information is received in a JSON format, with a defined structure. Depending on the type of tweet (timeline, mention, retweet, reply, etc…) structure of the JSON will change. Examples of attributes within the tweet JSON received are:

- Creation Date
- ID
- Text
- User Info (Nested JSON)
- Extended Tweet Info (Nested JSON)
- Entities Hashtags (Nested JSON)

In this case, we are interested in two types of incoming tweets: tweets posted by the football players themselves and tweets where these players are mentioned. Depending on the popularity of the player, the volume of tweets being received may vary. Thus, it is important to have them stored in a way where it will be easier to query the relevant information. Options for such storage systems, due to the format of the data, are distributed file systems (HDFS) or a NoSQL database (MongoDB). Because the data will be retrieved with the same structure every time, we can assume the schema is constant. Still, there must be checks in place to make sure that those attributes which we consider relevant are not retrieved with null values.

Facebook API

Another data source that will be used is Facebook Graph API data [2]. This API is the primary way for apps to read and write data from and to this social network. For our final product, at the moment of the sign up, the user will be asked for giving permissions to our app in order to be able to retrieve this information. Besides this, we will regularly (v.g.:

once per day) request this information, in order to update the information that we have. This information will arrive as a JSON object with a variable number of attributes (depending on the data that the user has introduced in Facebook). The pace of arrival depends, thus, in the pace at which users register to our app. Heterogeneity and expected volume will depend on the user activity on Facebook and the number of users, respectively.
There is a limit of 200 requests per user and hour. Our expectations are not to exceed this limit, as we do not need to make an intensive usage of these requests (v.g.: once per day, in order to update our information about the users).

Behavioural Test

The fourth type of data our app is going to ingest are behavioural tests. One of such tests is called CPRD and is specifically designed to analyze athletes. It measures how their performance is affected from 5 different psychological traits:

- **Stress Control**: it describes how the athlete adapts his behaviour to a different demand in training or competition and how he/she react to stressful situations;
- **Performance under evaluation**: how the athlete reacts during an evaluation and how he/she reacts to comments after a bad performance;
- **Motivation**: how an athlete keeps himself motivated and how much important is the sport for him;
- **Mental ability**: this factor includes mental abilities that help the sport performance;
- **Team Cohesion**: how the athlete is integrated into his team in terms of interpersonal relationships and satisfaction gained by working with the team.

The scale of the volume regarding personality tests is linear with the number of users.

Data Modeling and Storing

To understand how we're going to model and store our data sources, we first need to define the queries that are going to run on top of them. Below there is a sample of how we are going to use the existing data. Precisely, the main focus of our product is on presenting useful insights of the football players to the teams.

<u>Most frequent queries:</u>

1. Show the players by:
    a. Gender = male/female
    b. Location (Continent/Country/League/Team)
    c. Age ranges (e.g. [18-20])
    d. Player's position
2. Sort by:
    a. Behavioral Attribute1 (e.g. anxiety is low)

   b. Behavioral Attribute2

Depending on the data source, the model of the data being stored will defer. Such raw data will not be pre-processed in any way during the extraction phase. Thus, it will have at first a very simple model that allows the file storing system to benefit from data partitioning. Every source will be stored and modeled independently, as the transformations expected are different between each other. After applying the transformations and creating the on-demand views, the data models will change greatly to adequate an optimal storing strategy and mechanism that will benefit the system to perform well with:

- Sequential reads
- Indexing and pre-fetching
- Different fragmentation and partitioning techniques

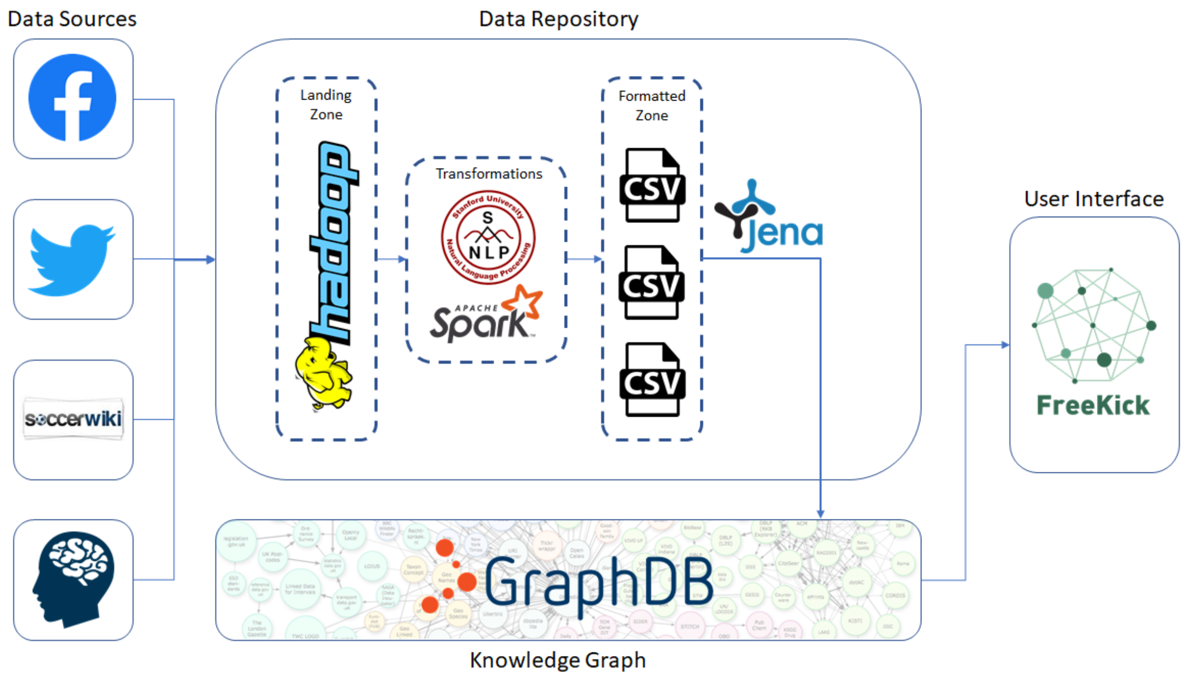
Data Exploitation

Different data exploitation operations will be performed on our data. Such operations will allow us to do aggregations, sentiment analysis, image analysis, etc.

1. Tweets → NLP/Sentiment Analysis
2. Facebook posts → NLP/Sentiment Analysis
3. Twitter and Facebook photos → Google Cloud Vision analysis/classification
4. Football API scraping → Data pre-processing
5. Psychological test → Data pre-processing

After these data pre-processing and exploitation techniques are applied, such transformed data needs to be stored again within our Knowledge Graph to be queried by the application. With this approach, at the end we will have implemented a complete big data flow processing.

Functional Architecture



In the above diagram it is depicted the functional architecture of our product. On the left hand side, there are the data sources of our system. In most of the cases, the inputs are in JSON format. This data is directly stored inside our Landing Zone without being processed. They are part of the Batch layer of our application as the collection of such data is done periodically. For instance, let's take as an example the collection of the players' demographics and statistics which is coming from scrapping. In this case, this operation should be performed at least once per year as the information of the player may change from one year to another, taking into account that the duration of the contract that a player has with a team varies from one to a couple of years. Moreover, this operation is done in batch mode as we will massively update the information for all the players rather than entry. By storing the data in raw files, it will allow us to integrate it later with new sources of data without needing to modify any schema, as well as generating new on-demand views as requested by the users.

The data that we collect is coming from heterogeneous sources and hence a pre-processing step is needed. That is why, right after the storage of the input files from the source system, we need to apply some transformations and prepare the data in such a way that it will be ready to be imported to the Knowledge Graph later. The main transformations that we apply in this step are done with Spark and by applying Natural Language Processing (NLP) techniques from Stanford University to generate Sentiment Analysis [3]. In the next step, the processed data are stored in the Formatted Zone in csv files. Within the Metadata Management System we will link the different pre-processed files to generate a global schema of our data. The tool will utilize these mappings to generate the on-demand views which will be consumed by the user interface within the Transformed Zone.

Tool Selection

As pictured within the architecture diagram, different tools are being used within each of the phases of the data processing flow.

Landing Zone

To store raw data we evaluated two alternatives, **MongoDB** and **HDFS**. Since our resources are formatted in a JSON fashion, both solutions could be implemented, however we finally selected the latter. The reason that led our decision is that we are going to ingest these data into Spark to perform a pre-processing step, so we never query these data, hence we don’t benefit from indexing.
Furthermore, Hadoop can store much more formats beside JSON and CSV. Even though right now we are ingesting JSON, it is reasonable to think that during the lifecycle of the software there will be a point in time in which we will decide to store new data to refine the model and if these new data are not JSON or CSV we would end up introducing an initial step to format them just to be able to store them in the Landing Zone [4].

Transformations

Regarding the step of the transformations, we mainly considered two alternative tools, namely **Apache Spark** and **Hadoop MapReduce**. After analyzing the data that we are collecting and the transformations that we need to make, we conclude that the most suitable tool in our case is Spark.
The main difference between these tools is the way they are processing the data. On the one hand, Spark processes data as much as possible in-memory, while MapReduce tasks are frequently reading and writing data to disk. Apache Spark should be definitely prefered in cases that the data that need to be processed have a high possibility of fitting in memory, otherwise they will be flushed to disk and reading/writing cost will be introduced. As a result, in cases that a huge dataset needs to be processed, MapReduce should be used as it has designed to mainly interact with the disk.
As we have already described, in our case the volume of data is not huge and hence it can fit in memory. For example, the initial number of players that we are scraping is around 100.000. Using Apache Spark to do the transformations on this dataset will be more efficient than doing them with Hadoop MapReduce.

Knowledge graph

Another important issue for our proof of concept was selecting the proper technology for modeling and describing raw data, as well as for querying it. In the beginning, we were considering using ODIN as a Metadata Management System. However, the fact that the queries we want to execute are standard, persuade us to build a Knowledge Graph that will make the data integration process easier.

The main reason that made us decide to use Knowledge Graph to integrate our data in a single storage engine was the existence of different data sources and the heterogeneity of the data. Specifically, the fact that the main concept of the data source is the **football player** implies that at some point in our architecture, we have to integrate the data in a single point, where the same player from the different sources will be considered as the same one. In addition, Knowledge Graphs improves the quality of our data, as we are able to link them with other ontologies and add semantics that can be later exploited to produce more meaningful results. For instance, as we have already described, we are using the players' data from social media and dbpedia includes the concept of Facebook and Twitter. Hence, we can link our data with these classes and directly use their properties without the need of redefining them.
Regarding the selected tools, we use Protégé to create our Tbox, Apache Jena to create the triples that represent our Abox and GraphDB as a database engine to import and query the data.

Formatted Zone

After the transformations are done for each corresponding data source, the data needs to be stored back to a repository for it to be utilized later by our application. For that reason, we'll generate CSVs with the transformed data, which will once more be processed to generate our ABOXs for the Knowledge Graph. Now the question would be which graph database repository to use?
Such comparison and decision will be reviewed more in detail on the Semantic Data Management project.

Use Cases and Data Flows

This specific project has 4 different sources of data, as mentioned earlier. It's important to understand the use cases and flow of each of those sources to give a complete picture of the big data ecosystem around the product.

Data Ingestion

Regarding the data ingestion, we will depict the process followed to collect data related to the football players that we include in our product. **The produced dataset that we use for our PoC can be found in this link.**

SoccerWiki Scraping

In our product we are interested in differentiating the players according to the league and the country that they are currently playing. That is why, the main program of our scrapper receives as an input the link of the league that we want to scrap (e.g. Spanish leagues). The next step to be done is to list all the teams that belong to this league and call another parser that scrap all the players of each team. The parser of the player is the part of the spider that generates the data and outputs them to a JSON file. The collected information and the transformations that are done in this step are the following:

1. Get the player's demographics ("Full Name", "Club", "Age", "Nation" etc...)
2. Get the current player's rating
3. Get the attributes of the player and flatten (transformed from list of attributes to different columns)
4. Get the list of positions and extract only the first one
5. Get his prefered foot

Facebook data scraping

Data from Facebook allows our product to enrich the personality information, as well as the possibility to obtain additional information such as team preferences for the players. Besides, information obtained from facebook posts by using NLP tools is important in order to assess the motivation of the players. In order to do this, we need to ingest our system with data from Facebook, which is obtained in the following manner:

1. The user has to give permissions to our Facebook API app
2. Get the user's demographics
3. Get the user's favorite team/s and athlete/s
4. Get personality information from the user (based on his/her likes)
5. Get sentiment analysis from player's posts

A visual representation of both processes is available in Appendix 2.

Data Processing/Analysis

For the data processing and analysis scenarios, we will depict the process followed by Twitter API and the behavioural test.

Twitter Data Processing

Previous explanation within the report showed the different capabilities that Twitter4J API [5] has available for the developers to retrieve data from the application. In this specific case, we're interested in retrieving the tweets and mentions from the different football players being scouted. Thus, we don't require to do it in a streaming approach, as it doesn't make sense to wait for the players to send tweets. We will therefore retrieve the player's timeline and mentions utilizing the *getUserTimeline*. This will retrieve a JSON document with an array of the tweets within the timeline.
Using Apache Spark, we created a JAVA project which retrieves the text from the tweets and mentions from the player and applies the NLP Sentiment Analysis algorithm developed by Stanford University [3]. This algorithm will define a number grading for the text from 0 to 4 (Very Negative, Negative, Neutral, Positive or Very Positive respectively). An end-to-end process diagram can be seen in Appendix 2. An example of the expected result after processing all of the tweets for one specific football player would be as following:

```
dani37pacheco:2
```

Where the first value is the twitter username and the second would be the average sentiment for the user's timeline. In this example, a value of 2 would mean "Neutral" sentiment. These transformations will be stored on a new CSV to be processed for generating the ABOXs.

CPRD Behavioural Test Data Processing

In the raw version of the behavioural tests, each test contains all the questions along with their answers.
As mentioned before, each test aims to measure 5 different characteristics of the mentality of a player, where every characteristic is an aggregation of some of the answers inside the test. Therefore, a Spark job is required to produce a new version of the data that contains the result of the test aggregated by the measures we considered.

References

[1] Twitter Developer Documentation. Retrieved the 2nd of June of 2020 from: https://developer.twitter.com/en/docs

[2] Facebook Graph API documentation. Retrieved the 2nd of June of 2020 from: https://developers.facebook.com/docs/graph-api/

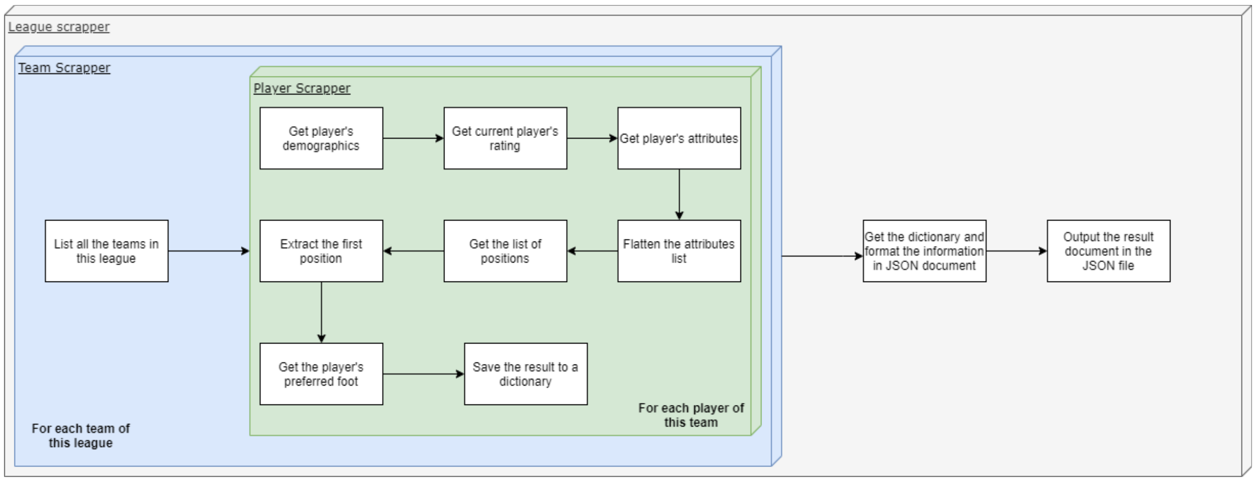
[3] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60. [pdf] [bib]

[4] Aptude. Retrieved the 2nd of June 2020 from: https://aptude.com/blog/entry/hadoop-vs-mongodb-which-platform-is-better-for-handling-big-data/

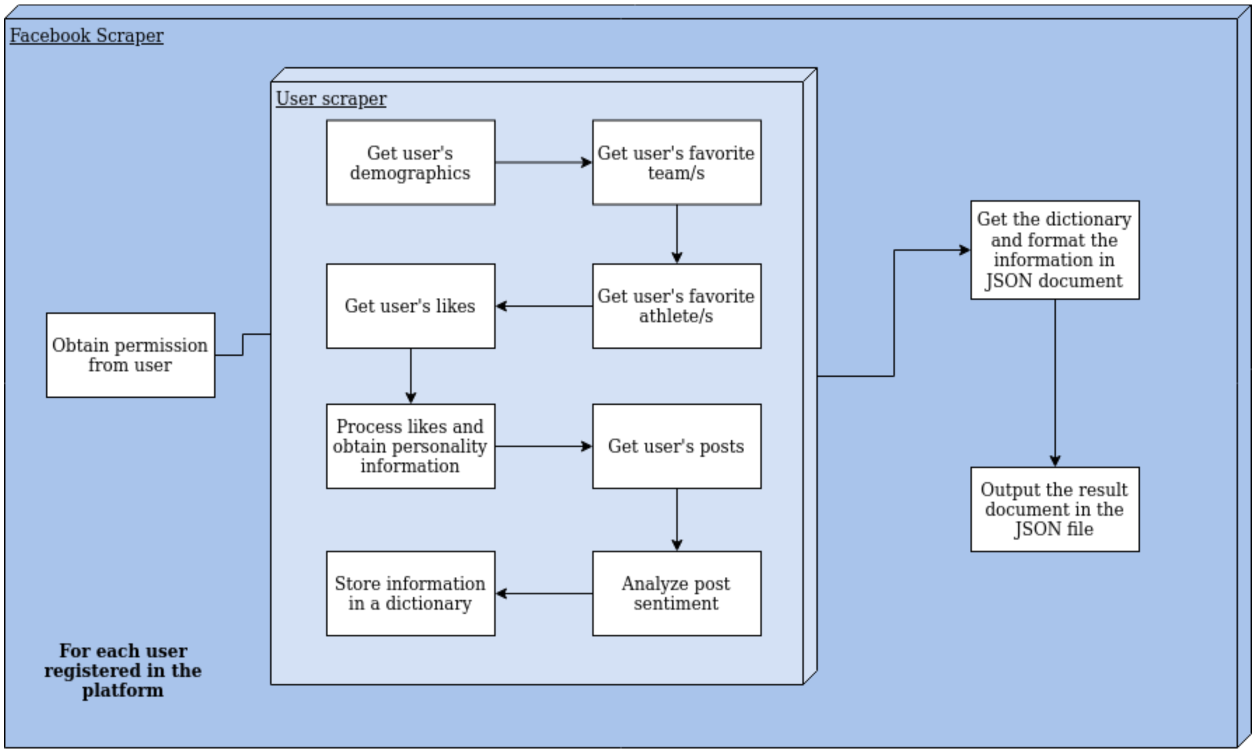
[5] Twitter4j Git repository. Retrieved the 3rd of June of 2020 from: https://github.com/Twitter4J/Twitter4J

Appendix 2

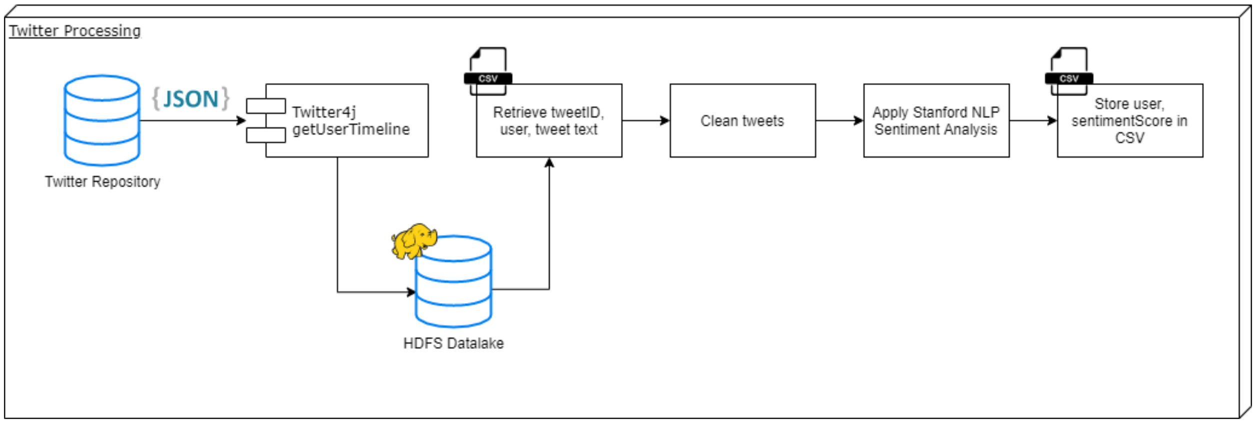
Scraping data flow diagram


League scrapper
Team Scrapper
Player Scrapper
Get player's demographics
Get current player's rating
Get player's attributes
List all the teams in this league
Extract the first position
Get the list of positions
Flatten the attributes list
Get the player's preferred foot
Save the result to a dictionary
For each player of this team
For each team of this league
Get the dictionary and format the information in JSON document
Output the result document in the JSON file

Scraping Facebook data flow diagram

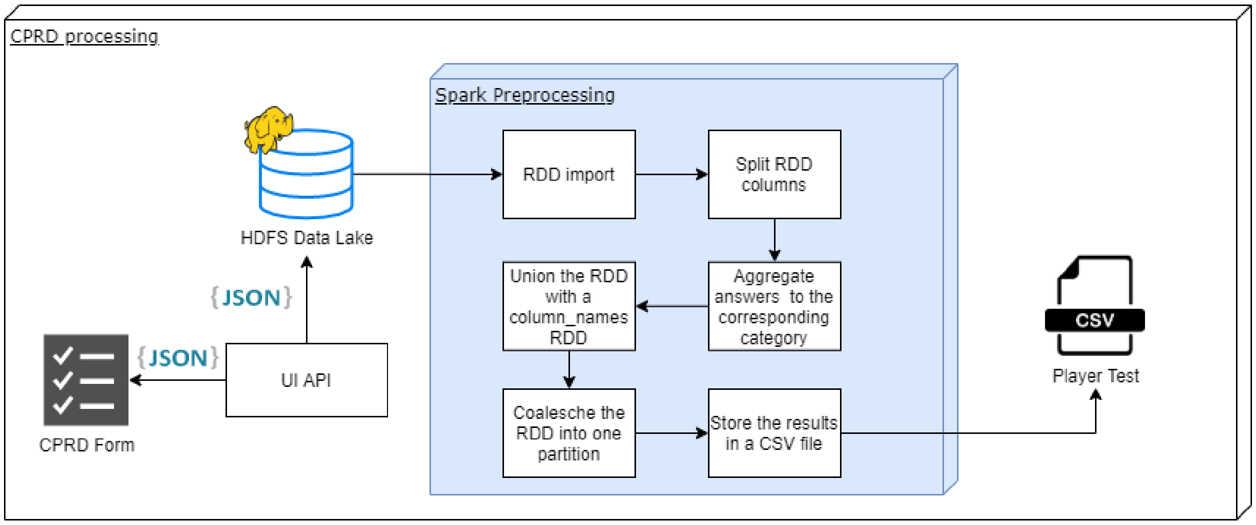

Facebook Scraper
User scraper
Get user's demographics
Get user's favorite team/s
Get user's likes
Get user's favorite athlete/s
Obtain permission from user
Process likes and obtain personality information
Get user's posts
Store information in a dictionary
Analyze post sentiment
For each user registered in the platform
Get the dictionary and format the information in JSON document
Output the result document in the JSON file

Twitter data processing diagram



Twitter Processing
Twitter Repository
{JSON}
Twitter4j getUserTimeline
CSV
Retrieve tweetID, user, tweet text
Clean tweets
Apply Stanford NLP Sentiment Analysis
CSV
Store user, sentimentScore in CSV
HDFS Datalake

CPRD Test data processing diagram



CPRD processing
Spark Preprocessing
HDFS Data Lake
RDD import
Split RDD columns
Aggregate answers to the corresponding category
Union the RDD with a column_names RDD
Coalesche the RDD into one partition
Store the results in a CSV file
CSV
Player Test
{JSON}
{JSON}
UI API
CPRD Form