Europe IT Jobs Website Final Report

Author: Juan Alonso Roman Garzon
Course: BSc (Hons) in Computing
Email: 10381163@mydbs.ie
ID: 10381163
Supervisor: Clive Gargan

Abstract

In this project we will implement a system which will be able to collect job descriptions from job sites allocated on internet, filters those which are IT and make them available to the user through a website.

In order to get this, we will need to implement three subsystems and integrate them all together:

- Crawler. This will collect the information from Internet.
- AI. Through machine learning techniques, we will label jobs in two categories: IT jobs, and No IT jobs.
- Search Engine. This will allow to the user access to a website, to search for jobs.

Acknowledgments

I would like to thank my brother, Jose for his support and helps in this project, as well as my family. Also, I would like to thank to all those people who shares their knowledge and contributes to open sources projects only because they love IT world as much as I do, and of course, a good challenge. I cannot forget to thanks to all those people who makes questions in forums about some technical issue. I have never understood their patient, but they open the path for all who comes after them.

Contents

Chapter 1: Introduction

Objectives

As everyone knows, the IT sector is growing up in an incredible pace. Companies more than ever has realized how important is the technology to achieve their goals. IT is no a department anymore, it is now a part of every department. It is the glue and the tools that everyone needs to carry out their tasks in an increasing competitive world. For this reason, the number of jobs opportunities in the IT sector is increasing exponentially. Right now, we can see the lack of professionals with experience in the market. This is doing that salaries are increasing and market from different countries are pushing to appeal more and more IT professionals.

Even considering the Brexit, Europe is a big market that allows people to move around and who speak English as a common language. However, there is not a web specialized in the IT market for Europeans. There are many webs for jobs, in each country, some cross borders but are not easy to use for this propose, and almost no one specialized in the IT sector.

Eurojobs is one example of a generic jobs' opportunities across Europe.

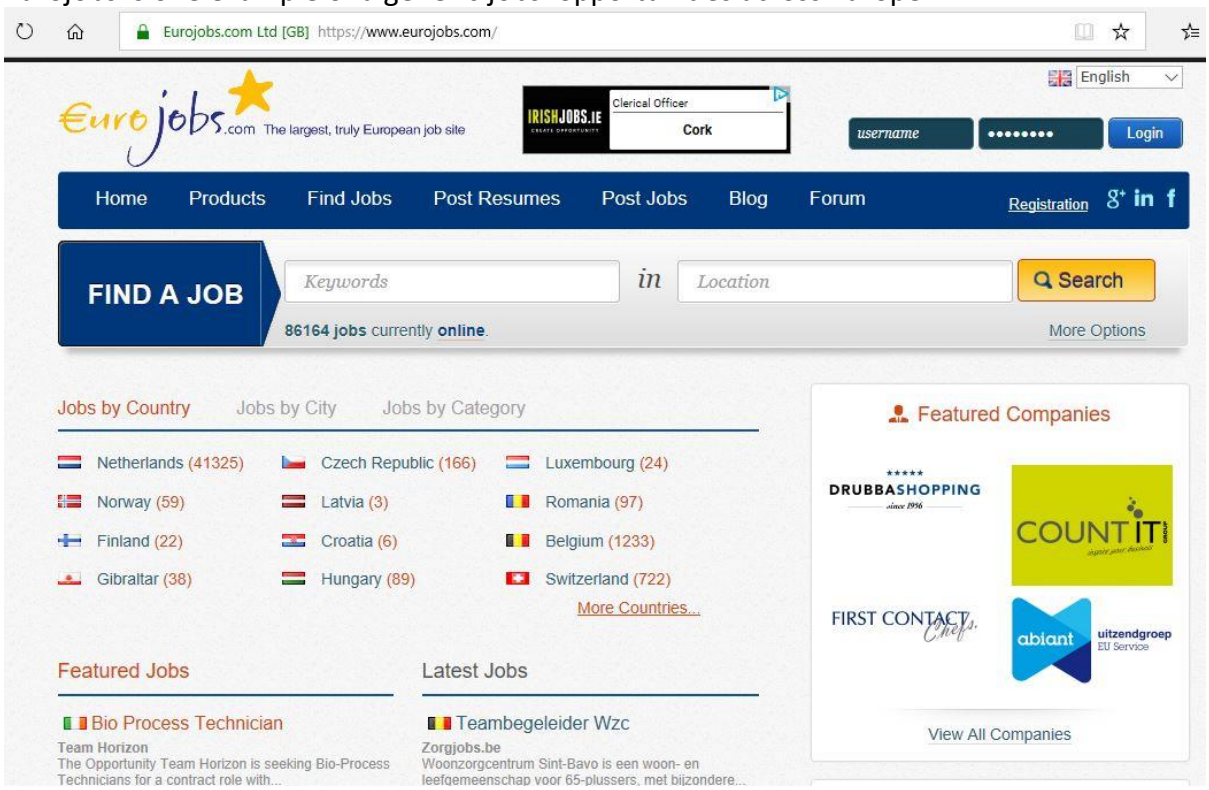


Figure 1: EuroJobs website

Another example is Europe Language Jobs, a website specialized in jobs for people who speak more than one language. This is an example of website specialized in one area: people who speak more than one language and wants to work as translators. It is a good solution for a niche.

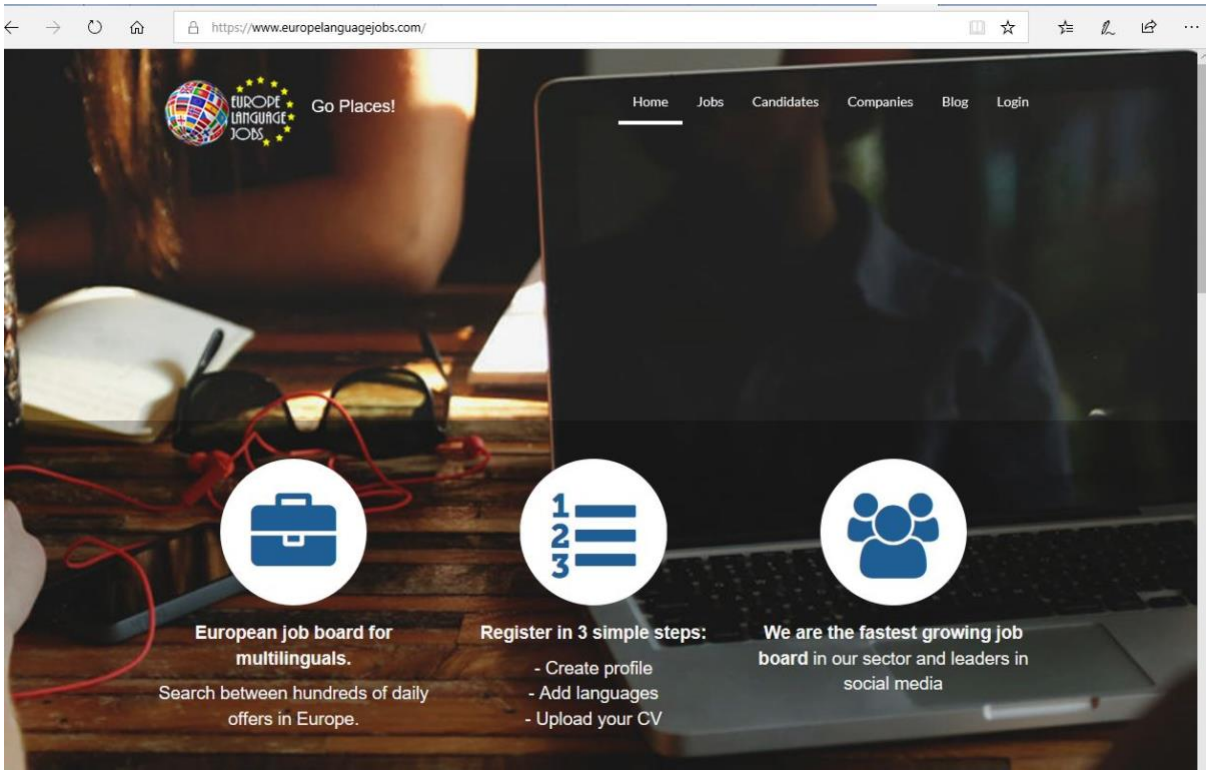


Figure 2: European language Jobs website

Another solution that many websites specialized in search of jobs is to add a section for international positions in other European countries, like jobs.ie does.

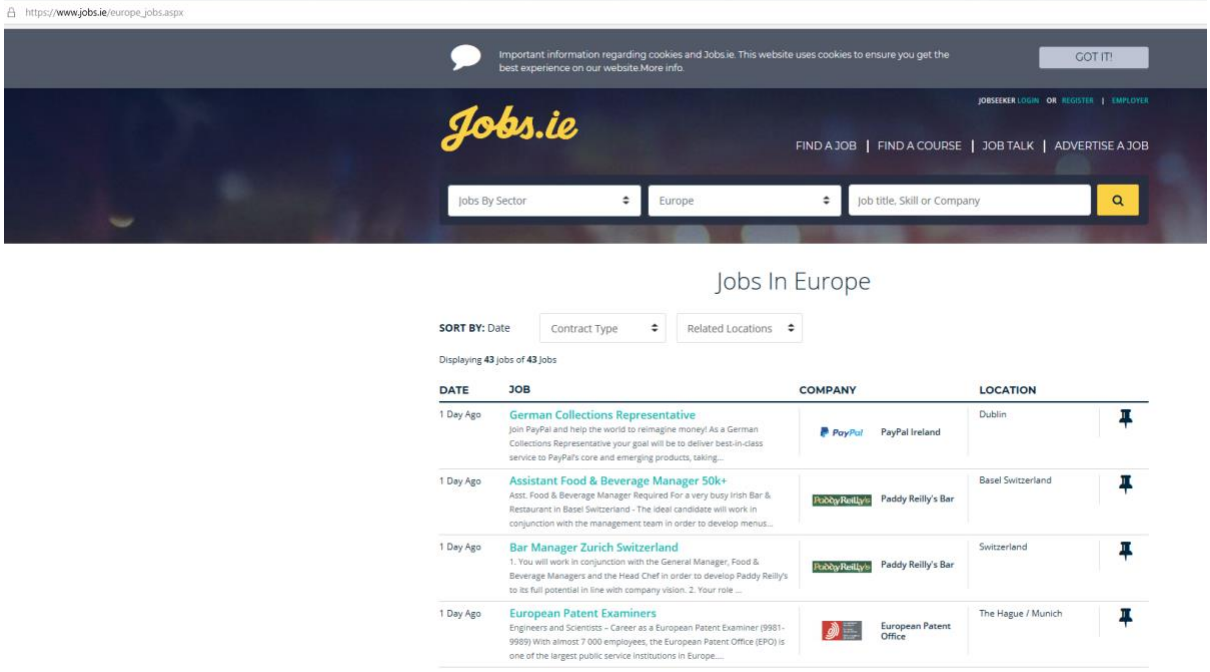


Figure 3: Section for European Jobs in Jobs.ie

This project will focus in creating a web site which will allow to anyone find IT jobs in Europe, all of them in English. We will focus in creating a search engine easy to use that will list a series of jobs given a search entry.

It will use a crawler to get jobs from websites across Europe and select only those that are relevant for the IT sector.

Approach

As we are not going to have employers or recruiters who are going to create a new position in our web, we are going to search on internet for any IT job position in English. For that, we are going to need to collect a bunch of websites which provide this information in their countries. So, for instance, we are going to go to Irishjobs.ie and collect every job opportunity they have issued in their portal. Then, we are going to filter those jobs that are in the IT sector and in English. After we collect and filter those jobs positions, we are interested in, we will provide a web portal in which our users will be search those in which they are suitable for. As we are collecting a lot of information about IT jobs across Europe, we are going to be able to provide what countries are more interesting for IT workers, what software languages are getting more popular in each country, and so on. So, another service we can provide to our users are statistics about their jobs, or another jobs in IT.

This project will have three parts:

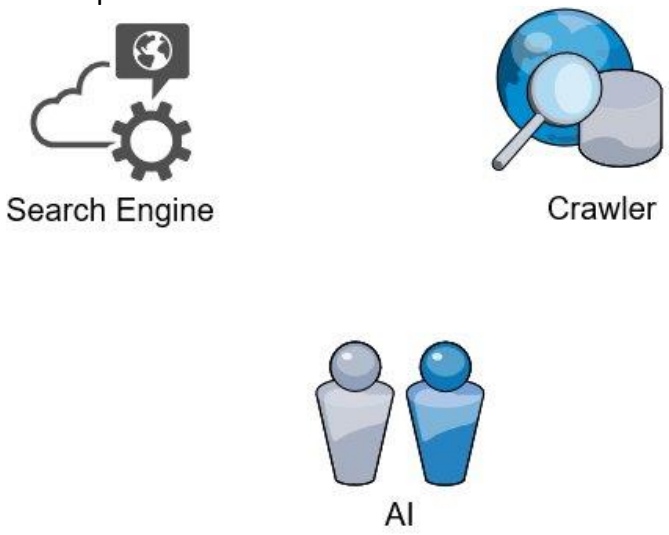


Figure 4: Approach

Crawler

We will implement a highly scalable system which will allow us to increase the number of websites that our crawler is going to use as source for our jobs. This system will be fault tolerance. As we are going to access to websites that can be down, or it is possible there is any issue on the network, we need to provide to our systems with mechanism to be fault tolerance, so, it will be able to keep working after a fail occurs. The crawler will consist in three components:

- **Administration Web**: This component will allow us to add new domains. Every job site will be a domain in our system. So, we identify a new job site which we want to get jobs' description from, then we add it using the administration web. Also, it will allow us to monitor what work has been accomplished.
- **Master**: This component will check in the database what domain are pending to process and will orchestrate the work to be done.

- **Node**: This component will communicate with the Master, telling it what its capacity is and when a domain has been processed. So, the Master can assign it the next domain.
- **Hadoop**: This will be a Hadoop instance in which our system will save each job description using JSON format.

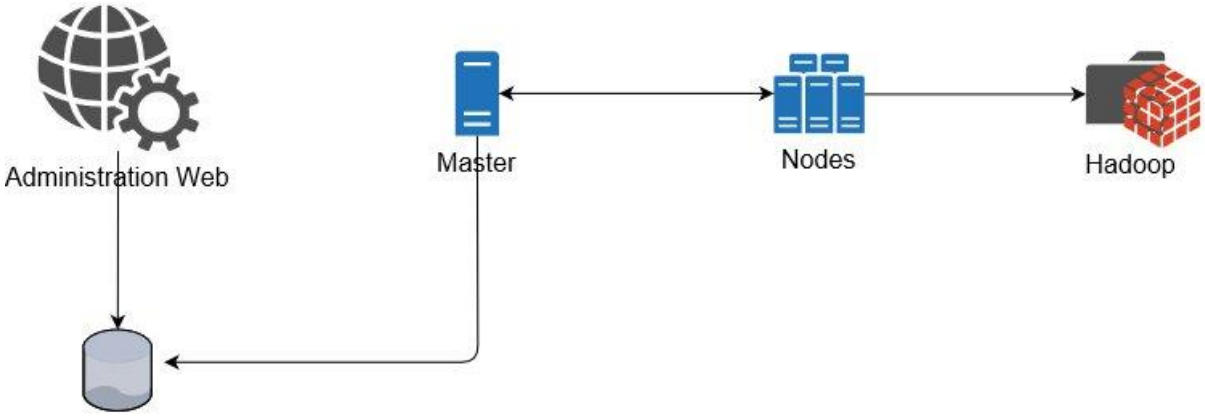


Figure 5: Crawler Approach

The Administration web will be implemented in Flask, which is a light technology that allow us to implement a website. The database will be in PostgreSQL, that is an opensource RDBM very reliable, able to manage concurrency (it will be access by Administration Web and Master). Master and Node will be implemented in Java. Java is the native language for Hadoop, which will make easier the integration, as well as it is a well-known language that is easy to create multi-threading programs able to communicate using the network, and work in parallel.

AI

In order to filter those jobs that are relevant to us, IT Jobs, we are going to implement an AI system that will decide if a job is or is not IT. First, we are going to need to train a model. To accomplish this, we are going to use the ML library of Spark. This library is using a higher-level API built on top of Data frames, instead of using MLlib built on top of RDD. We will use pyspark to create this model. Second, we will create a program in Java which will use that model to filter IT jobs and will send them to Solr.

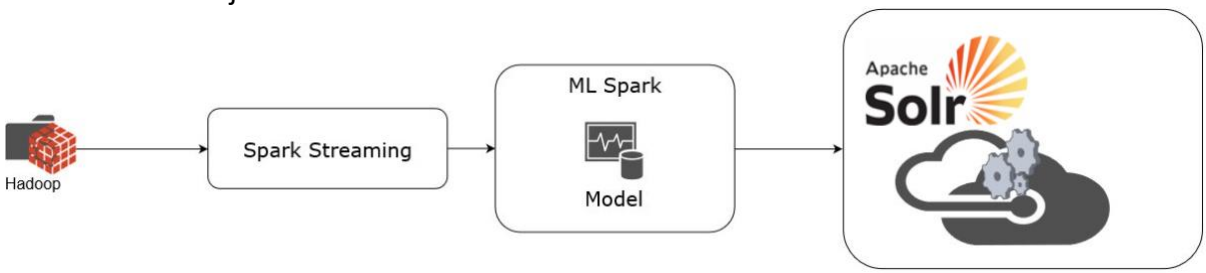


Figure 6: AI Approach

Search Engine

Once we have all the information in a NoSQL database as Solr is, we can query data. We build up a web which will do request to Solr for each user entry. Solr will retrieve this information and we will show it up to the user.

Figure 7: Search Engine Approach

Assumptions

As we are going to access to other websites to collect the information needed for our system, we assume that will not be banned from these websites. We are going to do many requests, as many as jobs they have, in a relative short period of time. They may consider us as a DOS attack, or they just consider that is not worthy to waste their resources in order to provide us with the information we require. In order to reduce the impact in their servers we will reduce the frequency of our requests to one per second and domain. Even, reducing the frequency, there is always a possibility that they decide to restrict the access to their servers, we assume this won't happen.

Chapter 2: Background

Context

This project will allow us to have a better understanding of the IT jobs in Europe. How they are being created, where, what specializations are getting more popular, in which country a job is more demanded.
The decrease of the price in transportation, the long-distance communications, a common currency, and an increase of the knowledge of English, is making that Europeans are moving from one country to another more frequently.
In order to facilitate this, is needed a central portal where people can see what opportunities are in which market. Now, if you are planning to go to a country, you need to get some information about what websites for searching jobs is more popular for your sector. Some of them are not even available in English, what makes the searching more difficult. Right now, a big demand of IT workers is raising in Europe and people are willing, more than ever, to leave their comfort zone and go to work to another country. From anywhere of Europe they will have a portal which will help them to find an IT job that suit their expertise.

Methodologies

In order to put in place this solution, we are going to need to use some technologies like: crawlers, machine learning, Big Data, web technologies.
Crawlers or Spiders are called the software used to search and download content on internet. They are programs that go from link to link parsing the content on a web page to get what they are looking for. They allow us to do so called web mining. More and more information can be found on the web right now, so this kind of software are becoming more and more useful every day. The most famous crawler is the one used by Google, Googlebot. Google uses a crawler to get information of every web and index their content. Using an algorithm, it retrieves that information when a user does a search.
In order to select whether a job is IT or not, we will use machine learning, a technic called Supervise Learning.
There are three types of classifications in Supervised Learning[1].

- Binary Classification: We are going to classify our input data in two categories assigning them one of two labels. Example: an email can be classified as Spam or Not Spam. In this example, “Spam” and “Not Spam” are the two labels.
- Multi-class/Multinomial Classification: We are going to classify our input data in one of many categories (more than two). For example: a fruit can be classified as banana, pineapple, orange, and so on. Each of them is a label, as “banana” or “orange”. A fruit cannot be more than one, it is “banana” or an “orange”.
- Multi-label Classification: We are going to classify our input data in one or more categories. For example: a movie can be classified as a thriller, romantic, comedy, terror, and so on. We have, in this case, several labels: “thriller”, “romantic” as examples. A movie can be classified as “romantic” and “comedy”.

[1] Luu, H. (2018). Beginning Apache Spark 2. Berkeley, CA: Apress.

In supervised learning we infer a model from a training data which we have labelled previously. There are some algorithms that we can use, the most important are:

- Linear Regression
- Logistic Regression
- Naïve Bayes
- Decision Trees
- K-nearest neighbour
- Support vector machine
- Random forest

In our case we will use Logistic Regression, which is great for a binary classification problem, as it is our problem. It consists in giving a series of input variables, find the coefficient that weight each input variable. The logistic regression is named for the function regression, whish has an S-shaped curve. This curve maps any value into a value that goes from 0 to 1, those values are not included. As we can see in the following figure, when we are training the model, the algorithm is finding the coefficients that better fit curve to our training data, dividing the inputs in IT jobs (green) and other kind of jobs(red). This will allow to predict the label of the new input.

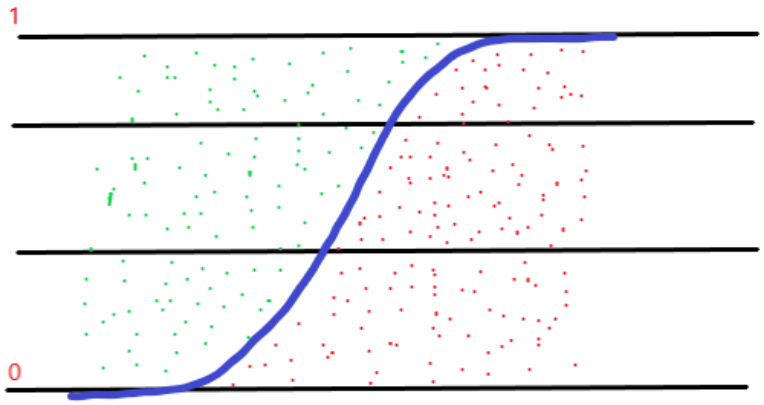


Figure 8: Logistic Regression

Software

PostgreSQL

In our project we will use two ways to save our data. First, we will use a most traditional approach: RDBMS. Our choice has been PostgreSQL. PostgreSQL is one of the most advanced open source relational database. It is a software that has been used for over 30 years, so, it is quite reliable and mature. It is extended used by many organizations in their production environments like: Udacity or Cloudera, and it has over 5% of the Market Share[2]. The main reason to use PostgreSQL is because we don't need a licence, it is reliable and mature. The other choice could be MySQL or MariaDB, also its use is very extended among many companies and it is open source, however it lacks functions as: Intersect, full outer join, partial indexes, so on. PostgreSQL allows many programming languages for stored procedures: Ruby, Perl, Python, JavaScript… and MySQL only SQL.

[2] Idatalabs.com. (2019). *PostgreSQL commands 5.56% market share in Database Management System*. [online] Available at: https://idatalabs.com/tech/products/postgresql [Accessed 10 Jan. 2019].

Solr

Second, we will use a NoSQL database as it is Solr. Well, Solr is more than just a database, it is also a search engine. It has been designed to deal with millions of documents, indexing then automatically. You don't have to worry about what scheme you have to create, or what is the structure of the data and their relations, not even dealing with indexes or normalization. Solr will manage that for you, we just have to focus in add documents and do the query. Solr is written in Java and uses the Lucene Java search library, as it does Elastic Search. Solr is an open source project from Apache Software Foundation. It is scalable, so, we can add new resources in case that our data increase, as it can have multiple servers in a cluster; it is highly reliable and fault tolerance. It is a mature technology used by companies like Netflix, Source Forge, Instagram, CNET and so on.[3]

IntellijIDEA

This will be our IDE to develop our java code. This IDE provide a lot of functionalities that will allow us to speed up the development, checking constantly the syntax, managing imports, even checking the spelling in our comment, among many other functionalities. It also allows us to use Gradle for our builds.

PyCharm

This IDE belongs to IntelliJ, and it is a great tool to develop in python, helping us setting up the environment, imports, and so on.

Gradle

Gradle is an open-source build automation system, like Ant or Maven, or at least following the same concepts. It is a domain-specific language based on Groovy and will allows us to automate the builds and deployments.

Jenkins

Jenkins is an open-source automation server, which will allow us to deploy automatically our code into our test environment. It is a fork of Hudson developed by Sun in 2004[4]. Once we have setup the automated tasks, we won't have to worry anymore about issues on the deployment.

Spark

Spark is a general-purpose distributed data processing engine built for speed, ease of use, and flexibility[5]. The key for being faster than other approach like Hadoop MapReduce is because is built on memory. It does an exhausted use of memory to store the data. It allows to use four languages using its API: Scala (the native language in which Spark is being written on), Java, python, and R[6]. It was created as a research project, due to the bad performance of Hadoop MapReduce for online jobs. API Spark has 4 main libraries: Spark SQL, Spark Streaming, MLlib, GraphX. In our project we will focus in Spark SQL, which will allow us to work with Data Frames, instead of using RDD, which is the lowest datatype in

---

[3] Wiki.apache.org. (2019). *PublicServers - Solr Wiki*. [online] Available at: https://wiki.apache.org/solr/PublicServers [Accessed 10 Jan. 2019].

[4] Heller, M. (2019). *What is Jenkins? The CI server explained*. [online] InfoWorld. Available at: https://www.infoworld.com/article/3239666/devops/what-is-jenkins-the-ci-server-explained.html [Accessed 10 Jan. 2019].

[5] Luu, H. (2019). *Beginning Apache Spark 2: With Resilient Distributed Datasets, Spark SQL, Structured Streaming and Spark Machine Learning library*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/beginning-apache-spark/9781484235799/html/419951_1_En_1_Chapter.xhtml [Accessed 10 Jan. 2019].

[6] Spark.apache.org. (2019). *Overview - Spark 2.2.0 Documentation*. [online] Available at: https://spark.apache.org/docs/2.2.0/ [Accessed 10 Jan. 2019].

Spark. Also, we will use Spark Streaming which using micro batches, allow us to work with a constant flow of data.

Apache web server

This is the most well-known web server. We will use this server for our search website. We will create a website in PHP which will be the interface for the user requests. The user will access to our website and introduce the search. This is a mature technology that has been used during years and has a market share of active sites of 42.04%[7]

CodeIgniter

CodeIgniter is a framework to develop dynamic web sites in PHP. It is a loosely framework, which avoid most of the boilerplate to create a site, and allows you, but not force you, to use the development pattern MVC (Model Visual Controller). We have chosen this framework because it is open source, it has a strong community, and it is very mature after over 12 years. It has been recognized as one of the fastest and lighter PHP frameworks (Lerdorf, n.d.). We will use it to create the website for searching, which will be used by the user to perform their job searchs.

Git

Git is one of the most popular distributed version control system. It is open-source developed by Linus Torvalds[8]. We will generate many files of code in different languages. Although, there is not a team which have to synchronize their work, we are going to work in different machines and have a history of changes. It is always good to have a repository, specially one on the cloud that will do a backup in case that something bad happens. We will use GitHub, as it has a free account for students.

Hadoop

Hadoop is an essential system if we want to handle big data. Our problem accomplishes the 4 v's:

- **V**olume. Our system will have to manage all the jobs coming from all European countries and keep them in a repository for further analysis. That is a huge volume of data.
- **V**elocity. Crawlers are going to work at the same time in many domains, downloading the description of each job. Jobs are being generated across all the countries in a fast pace.
- **V**ariety. The format of each website is different, we are going to need to work with a diversity of formats, and not structured data.
- **V**eracity. Due to the information is on internet and there is no so much control about what people post in those domains, we have to deal with this problem too.

Hadoop is a great ecosystem; however, we will use only HDFS integrated with Apache Spark.

Flask

Flask is s microframework for python to create light-weight websites. In order to manage new domains for our crawler and check the history of the domains and how many pages we have got, we need an interface. Flask will allow us to create that interface.

---

[7] News.netcraft.com. (2019). *May 2018 Web Server Survey | Netcraft*. [online] Available at: https://news.netcraft.com/archives/2018/05/29/may-2018-web-server-survey.html [Accessed 10 Jan. 2019].

[8] Chacon, S. and Straub, B. (n.d.). *Pro Git, Second Edition*. Apress, p.13.

Hardware

We will create a virtual machine with 8GB of RAM, and two CPU's. The operative system will be Linux Debian 9. This will be our test environment. In this virtual machine, we will install all the software that we need and do the deployment of the code implemented for this project.

Chapter 3: Literature Review

As I am working with many technologies, there will be necessary a diverse of literature. For doing the crawler, we have reviewed the book: Web data mining: exploring hyperlinks, contents, and usage data Liu, B.

To start to work with flask, the place is the website: http://flask.pocoo.org/, as you can find a strong community that has done a great job documenting this framework.

For learning about Hadoop, a great book is Professional Hadoop by Kai Sasaki and other.

For Apache Spark 2, a great book would be: Beginning Apache Spark 2

With Resilient Distributed Datasets, Spark SQL, Structured Streaming and Spark Machine Learning library Hien Luu. It gave me a good understanding of this technology.

Text analysis is a specific area of machine learning, so, we have to dive into literature like: Applied text analysis with Python: enabling language-aware data products with machine learning (Benjamin Bengfort, Rebecca Bilbro and Tony Ojeda)

There are some good examples about word2vec and how apply in the book: Machine Learning with Spark - Second Edition (Rajdeep Dua)

To get starting with Solr a good book is: Scaling Apache Solr (Hrishikesh Vijay Karambelkar)

Chapter 4: Crawler

Requirements

[Req:001]

The system will be implemented in Java, so algorithm that parse the index document and the web document must be java classes. They will extend the class Manager (to parse the index document), and the class Worker (to parse the content document).

[Req:002]

The system will provide a web that allow to the user add new domains, specifying the URL of the domain, as well as the name of two classes which contain the algorithm for parse the index document and the content document. If those class are not in the library, they must attach the jar with both classes.

[Req:003]

The system will provide another web that will allow to see the status of the process, how many machines are connected, how many processes are running for each node, how many pages has been processed for each domain, and so on.

[Req:004]

The system is composed by one Server master, which will contain the list of domains that need to be processes. This machine will be also the host of the web server needed for the interaction with the user. The server will not process any domain, for that we will add nodes.

[Req:005]

For adding a Node, we need to install a program, a java program, and a properties file in which we will specify the characteristics of the Node, and the address of the Server.

[Req:006]

The library with all java class for processing domains, will be stored in the Server, which will provide access to each Node.

[Req:007]

When a new node starts, this will be open a communication (TCP) with the Server, to register and inform to the server about what capacity of work it has.

[Req:008]

Every 3 seconds by default, although this can be setup in the properties file of the node, the node will inform to the Server about the status of its capacity.

[Req:009]

We will develop a protocol of communication between Server and Nodes, for each type of message.

[Req:010]

If the Server doesn't respond to a Node, this will log an error. This will continue with its job, until it has finished.

[Req:011]

All the documents downloaded from the web, will be storage in a Distributed System, for a posterior analysis.

[Req:012]

There will be a database hosted by the Server, which will storage domains in a consistent way, also this will provide a concurrent access.

[Req:013]
Each document that has been already downloaded will override the last version of the document into the storage system.

[Req:014]
The system will allocate two server sockets, one for the web and other to listen the communication with each Node.

[Req:015]
If a Node doesn't communicate for a period greater than 30 secs, the Server will consider that the Node is down and will log it as an error.

[Req:016]
Each Node will be able to run more than one domain in parallel, and access more than one web content for each domain. There will be only one thread to process the index document.

[Req:017]
There must be a way to stop the server and nodes in an easy way.

[Req:018]
To add a new domain, we should have to create a manager class to parse the index page of the domain, and a worker class to download the content. Those classes will provide methods easy to extend.

System Design

Overview

The system will have the following architecture:

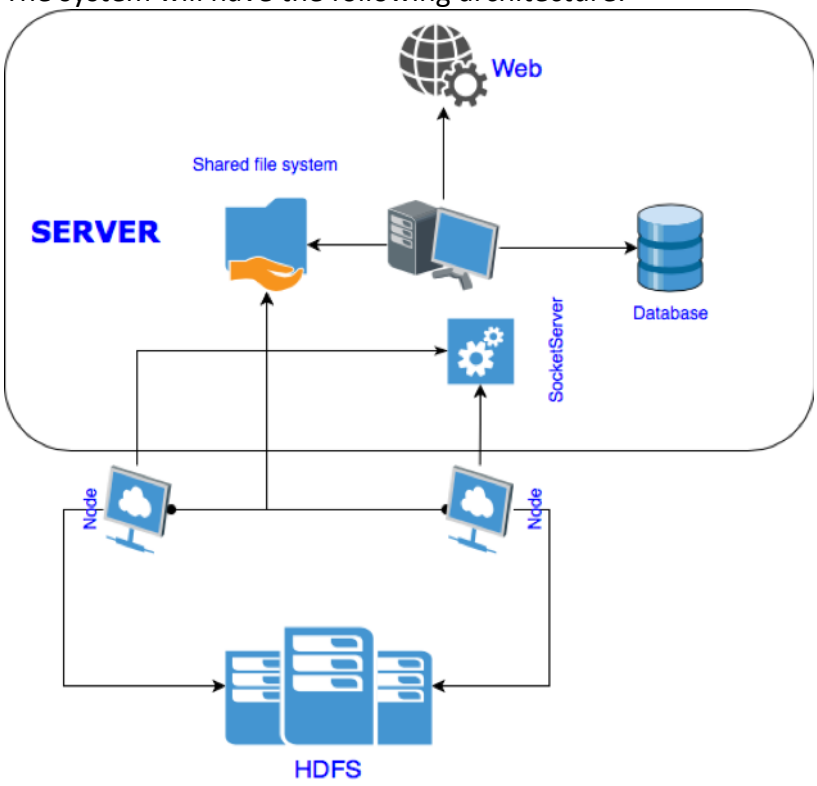


Figure 9: Crawler Overview

Module	Function
Web	The web will provide the GUI to the user. There will be two webs: one to add new domains and their classes, the other to be able to see how is

	going the process, providing information about the nodes, the domains and how many pages have been processed. [Req:002] [Req:003]
Shared File System	This will provide a repository for the classes for each domain. This file system will be a folder called lib, in which the server will upload the jars provided by the user when they enter a new domain, if those classes don't already exist. Every Node should have access to it [Req:006]. Also, we will save the logging in this file system. [Req:010] [Req:015]
Database	Database which will provide support to the web, as well to contain the list of domains and their classes, to parse them.
Socket Server	This will be a process which will listen Node's request.
Node	Each Node of the system, which will process the domain that the server sends to it.
HDFS	Hadoop system in which all nodes will store the content downloaded from each domain.

Communication

Discovery

Every Node has a configuration file **node.config** in which the IP address of the master will be setup. So, when the machine start will load the configuration file and load the IP address. It the IP address is wrong, this error will be recorded into the log, and the node will stop working as it is not able to connect to the master. As well, it will be in the **node.config** file the port in which the master will be listening. As soon, as the node reach the master, this will start the communication to register.

Protocol

The pattern of communication that the system will follow is Client-Server model: Master/Slave. The Server will be listening any request and the client will start every connection. The transmission will be one-to-one, and data can be transmitted in both directions.

Message

The message will be codified in ASCII, so this protocol will be text-oriented. This has some advantages, as it is easy to understand and monitor, flexible to extend and easy to test. We can use a telnet application to connect and send messages to the server. As disadvantage, it is too large.

This protocol will allow different types of messages: control and data transfer. The message will follow always the following structure:

The first part will be the command, which specify the meaning of the message, what is going to be about the parameters. Following the command are the parameters, can be one or more, depend of the command. Every part of the message: commands and parameters will be separated by the character|. Every message will end with a character '\n', new line. This will indicate that the message has finished, and the Node or the Server has the message. The message will be validated, and then processed.

List of valid messages

[INIT|MACHINE NAME |IP]

This is the first message that a Node will send to the Server, to be registered. If the system has already another machine already registered with the same name, will send a message of error to the Node that is trying to be registered, and this one will stop and log the error. In case that the machine doesn't already exist, the server will register the node as an active node in the database and send confirmation message.

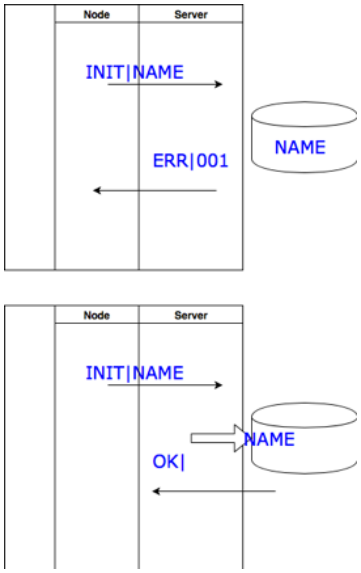


Figure 10: Communication Diagram

[ERR|001]

This message will go from the server to the Node, in case the message that is trying to register is already registered. The node will stop.

[OK|ID]

This message will go from the server to the Node, in case the Node has been registered successfully. The ID is the identifier of the Node in the system, is a unique number. This number will be stored by the Node for future communications, so it can be identified by the server.

[REP|ID|CAP|SPARECAPACITY]

Every 3 seconds, the Node will report to the server its status. The command is REP, followed by three parameters. The first parameter is ID, which is the unique identifier of the Node in the system. Then, the second is the Capacity of the Node, it means, the number of jobs that can carry on. The last parameter is the spare capacity, which is means, that, how many more jobs is capable to take.

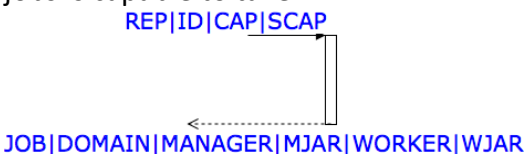


Figure 11: Communication Diagram 2

In case that there is spare capacity and there are more jobs pending, the Server will send a message with the job to do. Then, the Node will send again the Report with the capacity updated, and the server will repeat the process. In case there is no spare capacity (it is cero), or there is no jobs to do, the server will send a ACK-> OK| message, so the Node will end the communication.

[JOB| URLDOMAIN |MANAGER|MJAR|WORKER|WJAR]

In case that there is spare capacity for the Node and there are any job outstanding, the Server will send a message with the job that the Node has to do. The message will have the command: JOB, and the following parameters:

- URLDomain: This is the domain that is going from which we are going to get all the documents.
- Manager: This is the name of the class of the manager, the java class which will get the indexes for all the documents that we are going to download from the domain. The name of the class must has the complete address into the package: com.anansi…nameofclass.
- MJAR: The name of the jar in which this class is. This class will be loaded by the Node from the server.
- Worker: The name of the class of the worker, the java class which will download the document specified by the manager.
- WJAR: The jar which contains the Worker class.

[DEV|NODEID|URLDOMAIN|NUMBEROFPAGES]

This message is sent by the Node, when it has finished a job, to inform of that to the Server. This information will be sent in the periodic 3 seconds report. The message is DEV, followed by the parameters: NodeId, the identified of the Node in the system; URLDomain is the domain that has been processed; NumberOfPages, the number of pages that has been downloaded from that domain.

[STOP|]

This message will be sent to the Server by a telnet application or other. When the server receives this message, it will send this message to the rest of the Nodes. Every Node will stop. After about 30 seconds, the server will stop too.

[OK|]

The server will send this message to finish the communication with a Node, to indicate that there is nothing else to say.

Heartbeat and Node report

As we said, after register, the Node, will send a report with the status every three seconds.

Database Design

data model

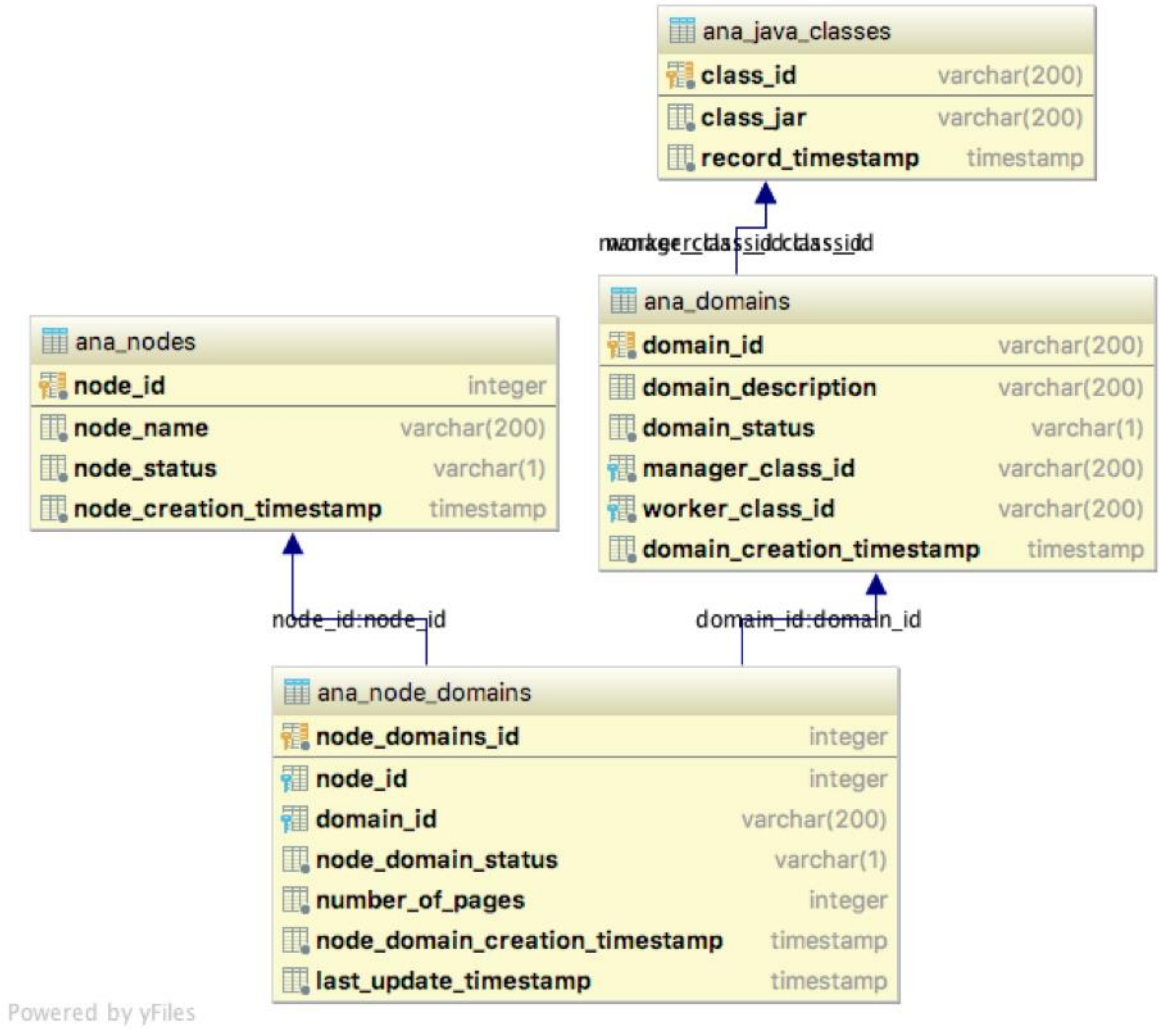


Figure 12: Data Model

The database will consist in one schema: anansidb with 4 tables.

Ana_domains

This table will contain every domain that the system will access to download their documents.

- Domain_id: The primary key, will be a natural primary key, the actual URL to the website.
- Domain_description: This is an optional value, a description about the website.
- Domain_status: This is the status of the node. The status can have two values (we need to create a constraint for this, so we can keep the database integrity): 'A' active, 'D' disables.
- Manager_class_id: The name of the class for the manager which is going to handle this domain. This is a foreign key to the table ana_java_classes.
- Worker_class_id: The name of the class for the worker which is going to handle this domain. This is a foreign key to the table ana_java_classes.

- Domain_creation_timestamp: This is an audit column to record when the domain was created.

Ana_nodes

This table will contain all the nodes that has ever been registered into the Server.

- Node_id: Surrogate key which will be the unique identifier for a node online. Along the life of the application, a node can have many rows in this table. Everytime a node is registered into the Server, a new row will be created.
- Node_name: Name of the node. Only one row with the same node_name and the status A can exists. A constraint in the database to check this is created.
- Node_status: A for an active node, D for a node when is no longer online.
- Node_creation_timestamp: A field which let us know at what time the node was registered into the system.

Ana_node_domains

This table will register every job performed by a node. Every time the server assign a job to a Node, a new row will be created. A node, can handle more than one job at the same time.

- Node_domain_id: Surrogate key, which will be the unique identifier.
- Node_id: The identifier of the Node which is doing the job
- Domain_id: Id of the domain .
- Node_domain_status: This is the status of the job. When the node is in the middle of this task, the status will be P (processing), when it has finished it will be C (completed).
- Number_of_pages: This is the number of pages that has been processed for that domain. This data will not be available until the job has completed the task.
- Node_domain_creation_timestamp: This is the date at what the job was assigned to the node.
- Last_update_timestamp: This is the last time when the row was updated.

Ana_java_classes

This table will contain every new class that has been registered into the system. For every domain, there are two kind of classes (they can be the same or used across many domains), the worker and the manager. In this table, we register them and link them with the jar which contains them.

- class_id: Natural primary key, which is the name of the class.
- Class_jar: Name of the jar which contains the class, so the node can call remotely and load the class.
- Record_timestamp: The time when the class was registered into the system.

Software design

overview

There are three components software in our system: website, server and node. The website will provide the interface to the user, so they can add new domains and check the status of the nodes. Also, will provide the jar classes to every Node. The server will orchestrate nodes; it will provide jobs and will check the status. Node will be the software which will do

the job, which can have more than one instance. Let's have a look about how they are integrated:

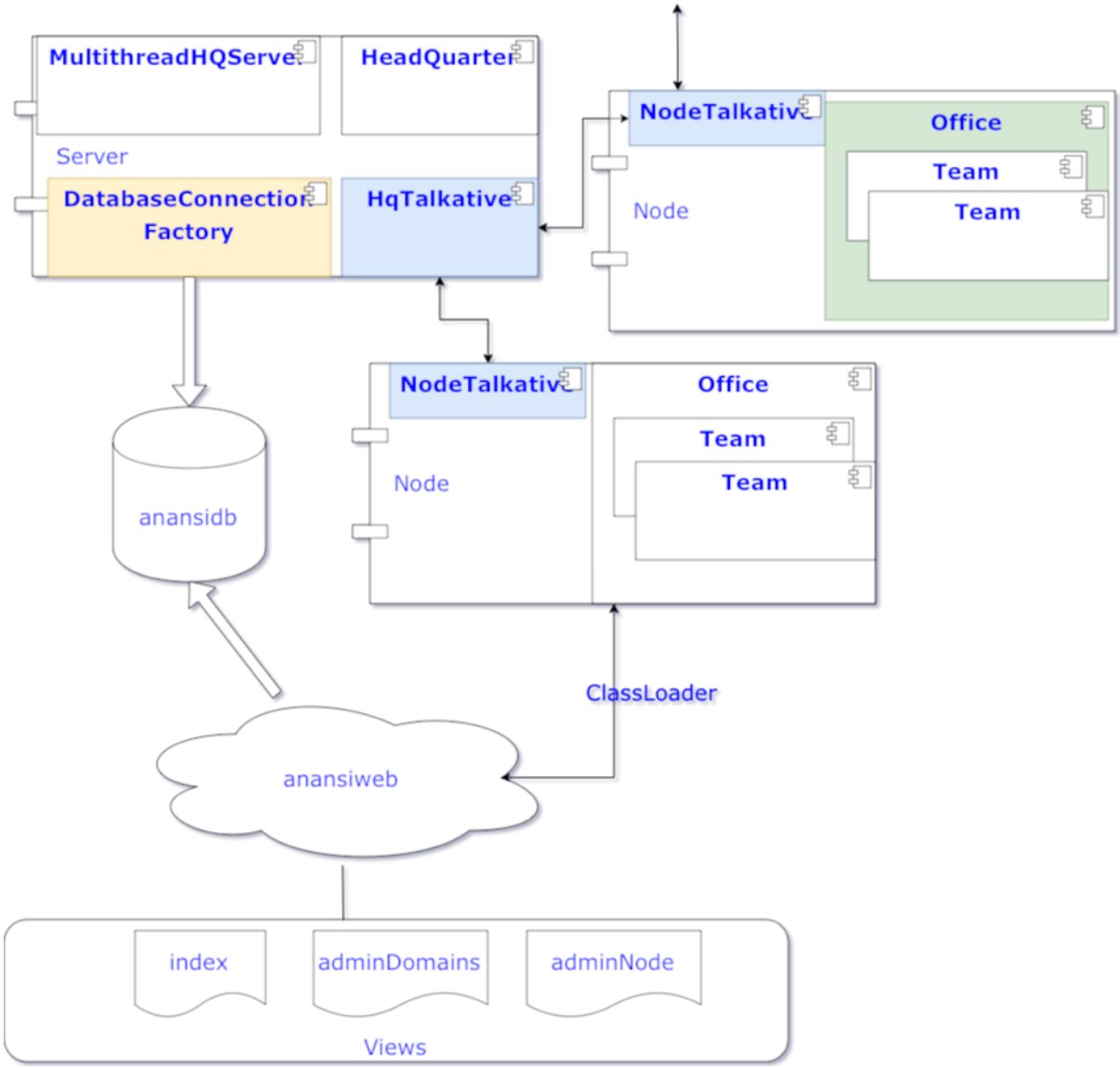


Figure 13: Sowftware Design Overview

Anansiweb

The web will provide the interface to the user, so they can add new domains, classes and check the status of the process. It will have three views:

- **Index**: This is the main web, it is a presentation to the user, and it has a link to the rest of the pages.

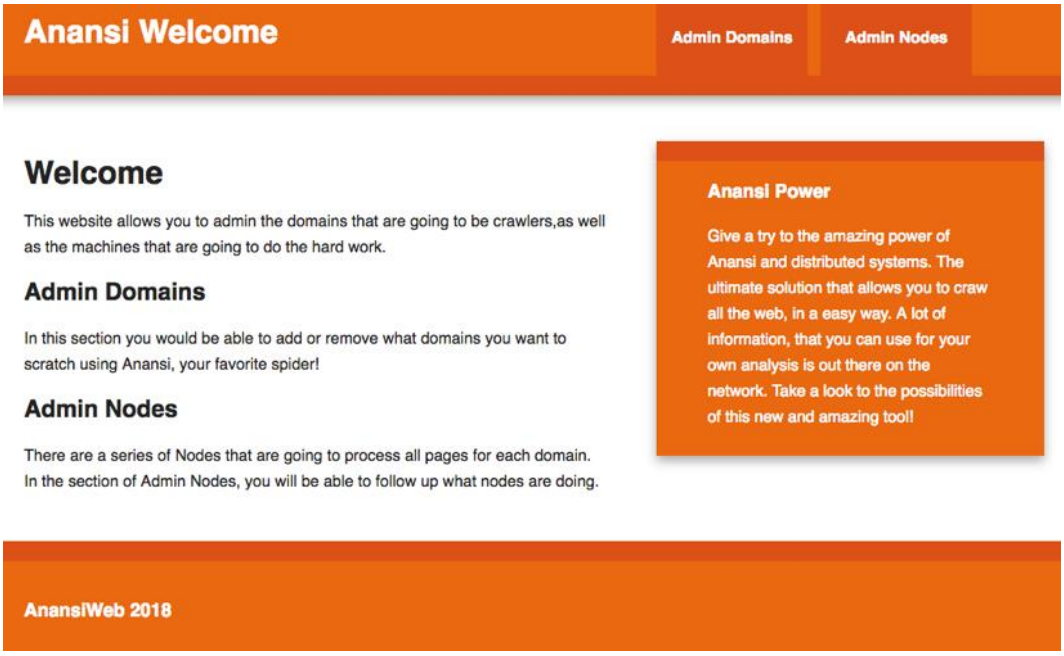


Figure 14: Anansi Web Home Page

-

- **adminDomains**: This is the page in which the user can add new domains and their classes.

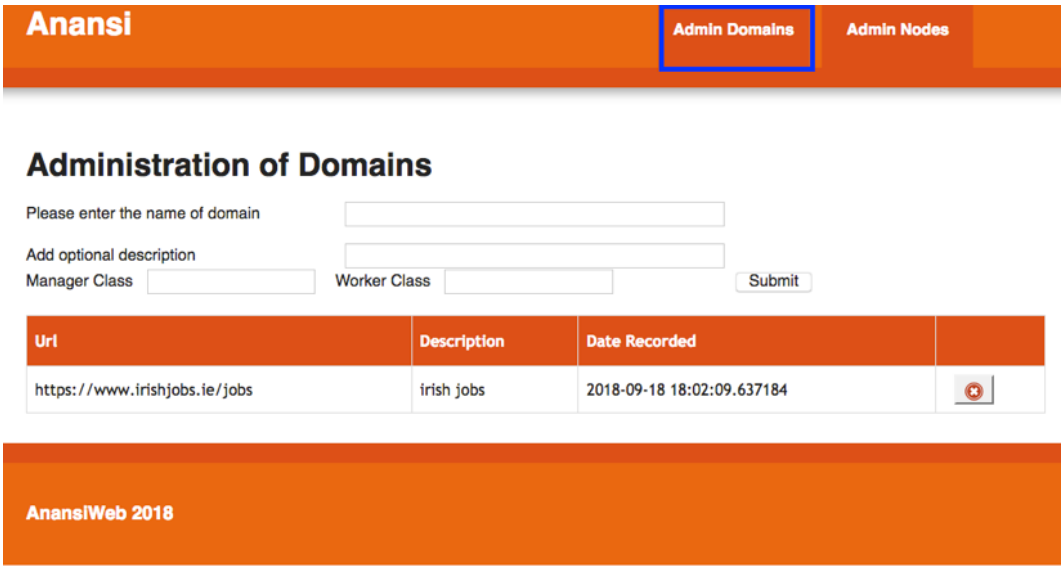


Figure 15: Anansi Web Domains Page

When the user adds classes that not exist into the system, the website will redirect it to a page which will allow them to update those classes. Only the class missing will be required.

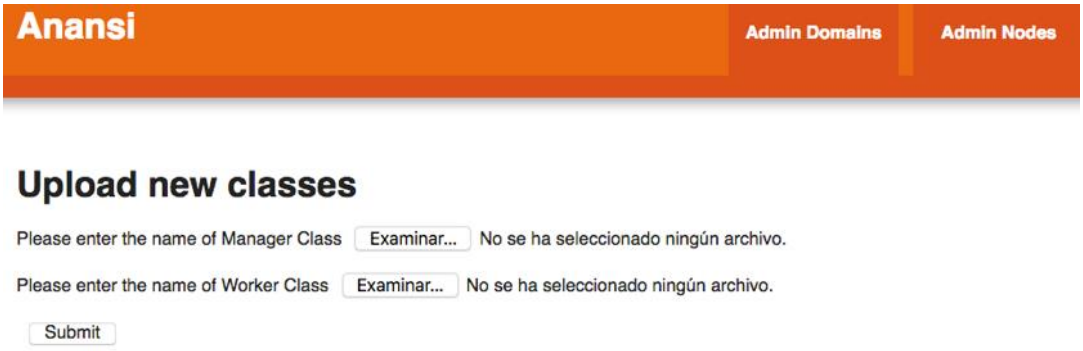


Figure 16: Anansi Web Upload Class

Only jar files are allowed, in case to try to upload another kind of files, the system will give an error and it will inform to the user.

As we can see in the first screenshot of the adminDomain, below the form that allows to the user to introduce new domains, there is a table showing all domains actives. Domains can be removed with the cross that exists in the table. This won't actually remove the domain, but it will change the status to D, which will do it invisible to the user, but will keep the integrity of our database.

- addNodes: This page will show every node active into the system, and what domains are working on.

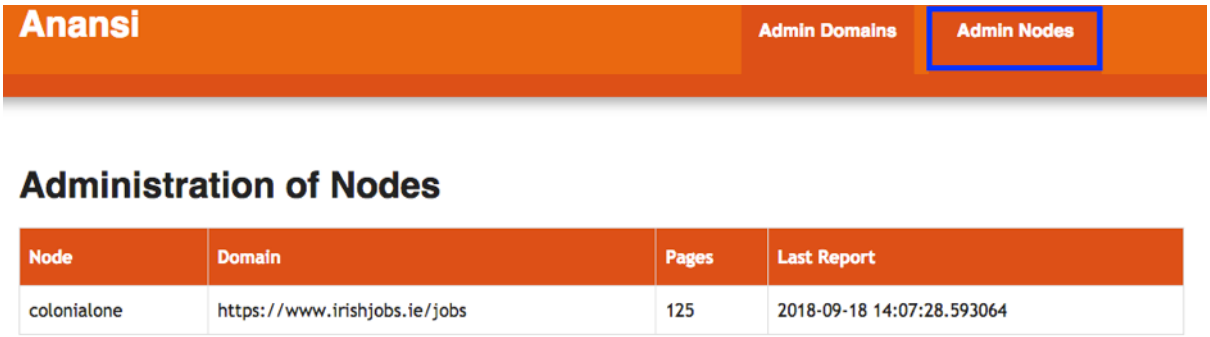


Figure 17: Anansi Web Nodes Page

Server

The server will communicate with the web through the database. This means, that the web will insert new domains, and the Server will register new nodes, and they share this information through the database. The database will provide the concurrent access, that our distributed system needs. This database must have failure recovery process, guarantee the atomicity and the consistency of the information. The server has four components:

- HeadQuarter: This will start up the rest of components, it is the entry point of the application.
- DatabaseConnectionFactory: This class will create a pool of connections that can be reused by any server socket.

- MultithreadHQServer: This is the server socket which will accept the connection from the nodes. For every connection accepted, this will create a new thread, which will process the communication. In this case, the server can provide service in parallel to multiples nodes.
- HQTalkative: This will take care of the communication with the node. This will process the message and connect to the database to get the next jobs, and assign the job to the node. Following the protocol, will communicate with the Node. When the communication has finished, this will close the socket. There could be more than one HQTalkative instance at the same time, so we have to manage the concurrency. These nodes are not sharing objects, all of them will access to the database which will guarantee the concurrency and the integrity

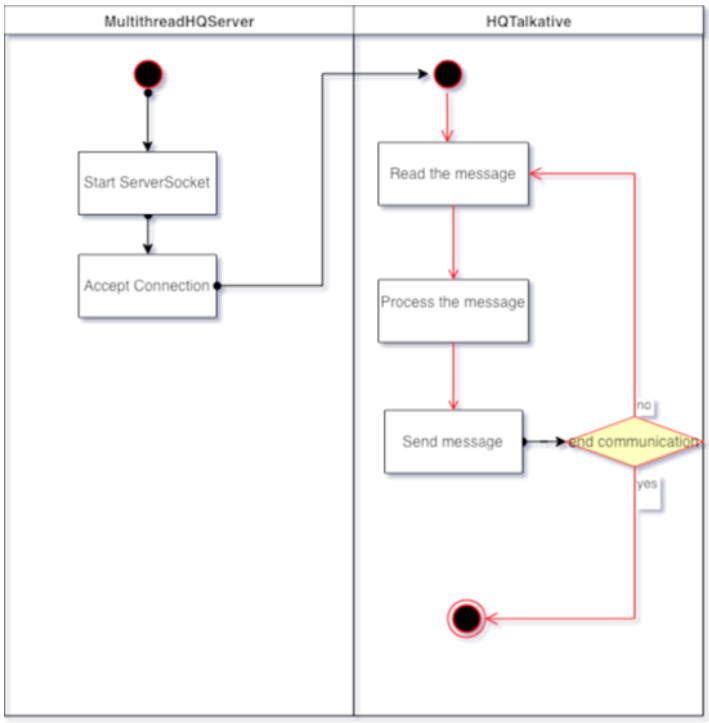


Figure 18: Master Flow Control

Node

The Node is the most complex module. As each node can have multiples threads workings, we call them teams, and each team is going to have from one to n workers downloading pages, and a manager to orchestrate them, the complexity and the concurrence will be a challenge. To afford this, we have tried to mimic an office structure. Each node will be an office, which can have many teams. In each team there will be a manager, which will assign the tasks and workers who will perform them. In order to manage all the work, there will be a scrumboard, in which the manager will set the new tasks. Let's see this concept in a diagram for clarification

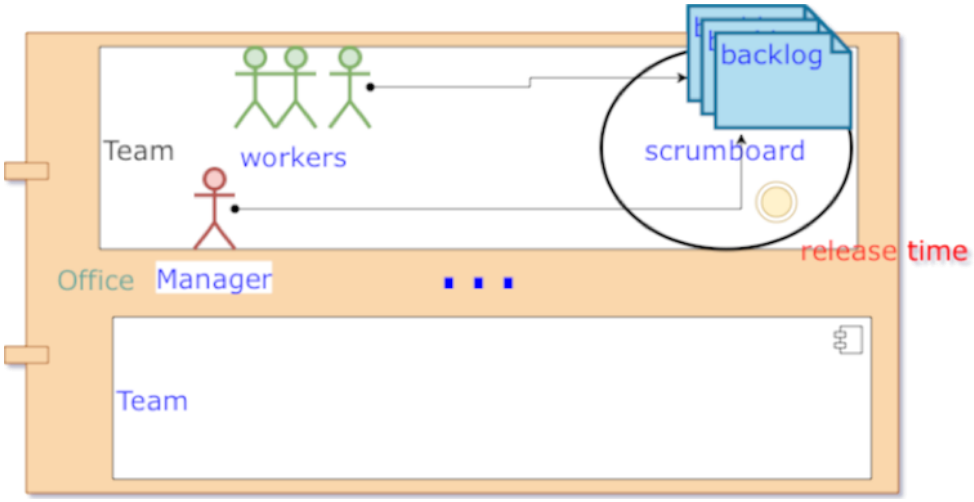


Figure 19: Office component

As we can see, in order to communicate the process, workers and managers, we will have an object called scrumboard. This object will have a queue with every task, pages to be downloaded. The Manager will put new pages into this queue called backlog and workers will take them from the queue when they are available, and they have finished their previous task. Teams will not share their tasks, so they don't need to share the scrumboard object. However, all teams will share two queues, the first queue is the queue where jobs will be added, projectQueue. The second queue is the deliveryQueue, every time a team complete a job, they will put in this queue and the NodeTalkative process will pick up from the queue and communicate to the server.

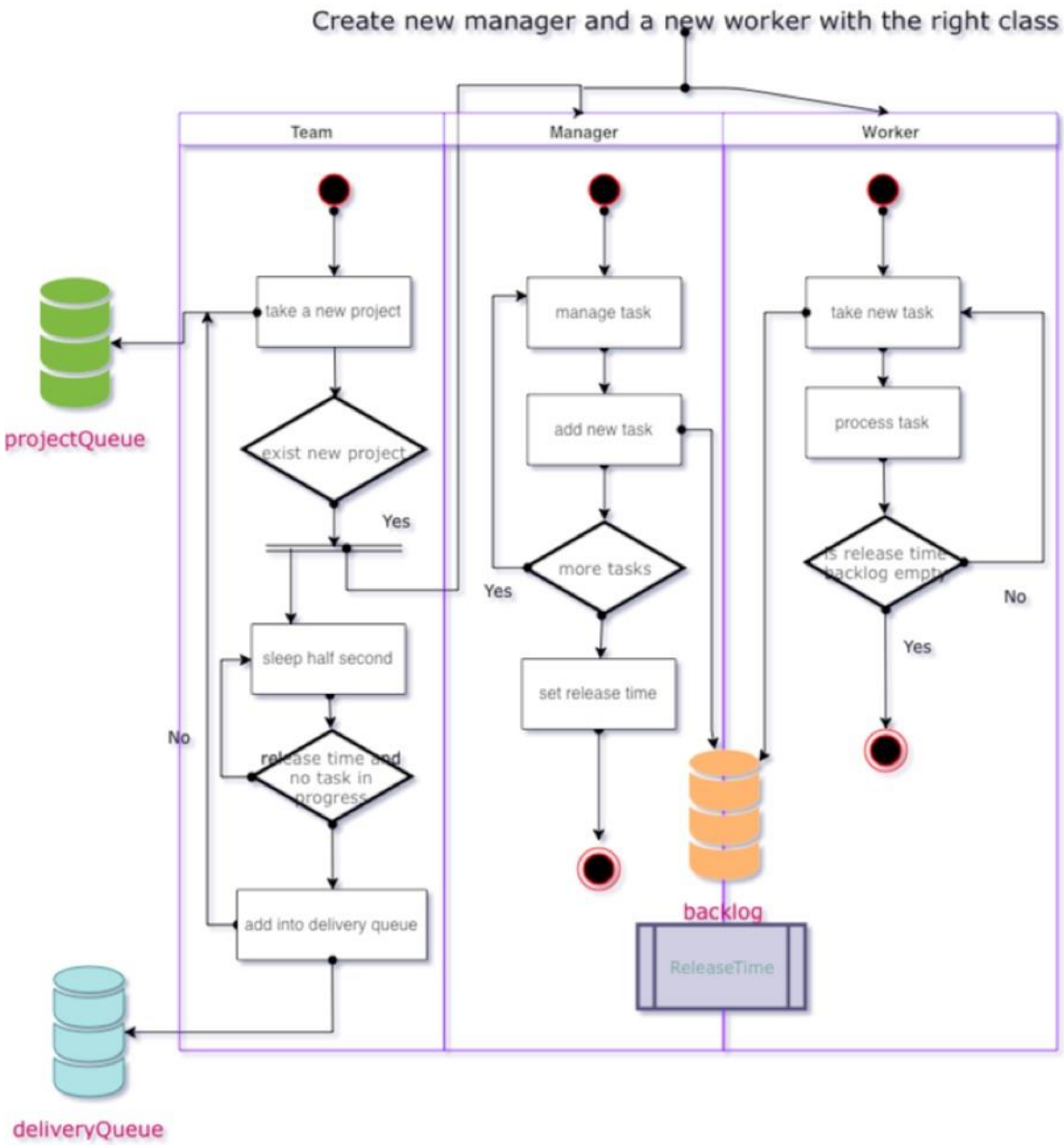


Figure 20: Node Flow Control

For the queue we use a Java object that allow to block the object until it has a value on it, so we don't have to worry about the object wasting time into the processor. ReleaseTime is a shared object, we use Java techniques to synchronize its access. This means, semaphores. Every worker will write a new file for each document in the HDFS system, using the native API that Hadoop provide.

Extension of new classes

As the requirement 3.1.18 [Req:018] specify, we need to provide an easy way to extend the number of domains that our system can process. So, the system can process a new domain, it should extend the following classes:

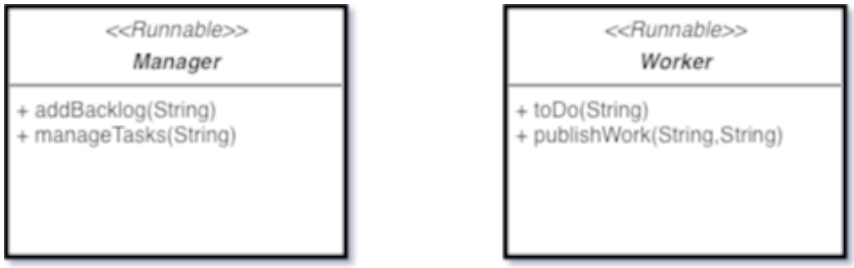


Figure 21: Manager and Worker Classes

The new Manager class must override the method manageTasks, which should call to addBacklog for every new index to a page that it finds.
The new Worker class must override the method toDo, which should call to publishWork for every new page that has been downloaded, so the framework will take care of storing it into the HDFS system.

Chapter 5: AI

Requirements

[Req:019]

The system must be able to detect a new job in the file system.

[Req:020]

The system must be able to load a job from a file formatted in JSON with the following attributes:

- Title: Title of the job. Websites have a title for each job.
- Description: The whole text of the job. Without html tags.
- Link: The URL link which point to the source of job offer.

[Req:021]

The system must be able to distinct and filter jobs that are not IT with an accuracy of over 90%.

[Req:022]

The system must be able to send those jobs that have been identify as IT jobs, to Solr, using a JSON format.

[Req:023]

The system must be fault tolerance, in case that there is any problem with a job, it will process as far as it becomes available again.

[Req:024]

The system must be highly scalable. In case that the volume of work increase, it should be possible increase the throughput increasing the number of resources.

[Req:025]

The system must be able to be integrated with Hadoop and access to its HDFS system.

System design

Overview

The system will have the following architecture.

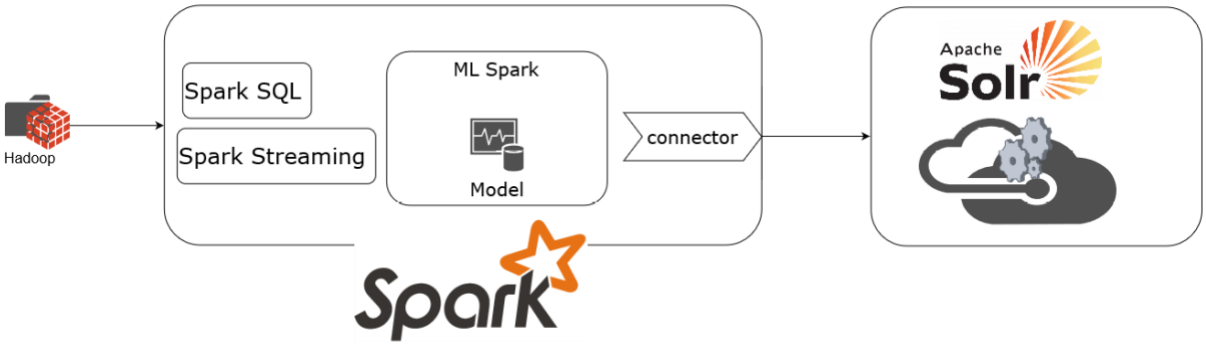


Figure 22: AI System Design Overview

Module	Function
HDFS	Hadoop system in which all nodes will store the content downloaded from each domain. [crawler system]
Spark Streaming	Spark streaming will be able to detect and load a new file in Hadoop System [Req:019] [Req:025].
Ml Spark	It will be able to distinct jobs and filter using Spark SQL capabilities [Req:021]
Spark SQL	Using Spark SQL capabilities will able to load a JSON format file out-of-the-box [Req:020]
Connector	This module will send each processed job to Apache Solr using JSON format [Req:022]
Spark	Spark technology it is fault tolerance [Req:023] and highly scalable [Req:024].
Apache Solr	Apache Solr will be the storage system for IT Jobs filtered by our machine learning. Our system will send new documents using SolrJ connector [Search Engine System].

Training Model Design

Input training data

First thing needed when we are going train a model is to prepare the training data. The training data is a sample of the data that we need to classify, which we have already classify manually, without intervention of Machine Learning. It is a tedious work, but we must be accurate if we want to create a robust model. We save a sample of 3141 jobs downloaded by the crawler in a folder, 1832 have been classified as “IT jobs” and 1309 have been classified as “No IT jobs”. It is good that the sample is well-balanced. Each job is saved in individual files codec in UTF-8 and the format of the information is in JSON. The structure of the JSON file is the following:

```
{
    “title”:”<<title of the job>>”,
    “labelIT”:”<<classification>>”,
    “description”:”<<description>>”
}
```

- title: Title of the job, that has been created by the agency or company which has created the entry.
- labelIT: This value is the one that we have added manually. It can have two values, 0 for those jobs that are not IT, or 1 for those which are IT.
- description: This value is the whole text of the job. It has not been clean up, only formatted in a way that is compatible with JSON attribute value.

Code:

```
# Loading the training data into our data frame
data_frame = spark.read.json("./training")

# Cast label IT as a double type, it must be a double to be label in ml algorithms
data_frame = data_frame
.withColumn("labelITTemp",dfOriginal.data_frame.cast("double"))
.drop("labelIT")
```

```
.withColumnRenamed("labelITTemp","labelIT")
```

Data Transformations

Data is coming from a web, what means that we are going to need to clean it up before we can use it for training. Tags, html characters, punctuation symbols and so on can affect to the result of our model, that is why it is important to do a previous transformation. Another kind of transformation that we need to perform is specific for natural language processing. We cannot use for our training what is called raw text, we need to tokenize and remove the stop words[9]. Tokenize means, to convert a raw text in a vector in which every element of the vector is a word. Basically, use space to know when start and end a word and it split it into a vector. Removing stop words means, taking that vector which, we have converted our raw text into, and remove the words that we use in our grammar to connect verbs, names, adjectives, and so on. For example: the, and, this … are stop words in English. Obviously, each language has its own set of stop words. They are not relevant for the meaning of a text, that is why they should be removed.
Ml library of Spark has the tools that allow us to tokenize the text and remove the stop words for a specific language, included English.

Code:

```
# Regular expression to match html symbols:   &amp; and so on
regexp_html_symb = "&[^;]+;"

# Regular expression to match symbol of punctuation
regexp_punctuation ="\.|,|-|'|\"|@|#|~|\_|\!|\?|\(|\)|[0-9]+|\:|\/|;"

from pyspark.sql.functions import *
# Remove html symbols
data_frame = data_frame
.select(regexp_replace(col("description"),regexp_html_symb,"")
.alias("textClean"), col("labelIT"),col("title"),col("description"))

# Remove puntuation symbols
data_frame = data_frame.drop("description")
data_frame = data_frame
.select(regexp_replace(col("textClean"),regexp_punctuation,"")
.alias("description"), col("labelIT").alias("label"),col("title"))


from pyspark.ml.feature import RegexTokenizer
# Creating tokenizer: we specify where the raw text is: description; and the name
# of the new column with our text tokenizer: textVector.
# The pattern to split words is " " [blank]
# All the token will convert words to lowercase
regexp_tokenizer = RegexTokenizer()\
.setInputCol("description")\
.setOutputCol("textVector")\
.setPattern(" ")\
.setToLowercase(True)

# Apply tokenizer to our data frame
data_frame = regexp_tokenizer.transform(data_frame)

from pyspark.ml.feature import StopWordsRemover
# We create the function which will remove the stop words. The language is English
englishStopWords = StopWordsRemover.loadDefaultStopWords("english")
remover =
StopWordsRemover(inputCol="textVector",outputCol="textVectorClean",stopWords=englis
hStopWords)
```

[9] Beysolow II, T. (2018). *Applied Natural Language Processing with Python*. Berkeley, CA: Apress.

```
# Apply transformation to remove stop words
data_frame = remover.transform(data_frame)

# we are interesting only in the following columns: label, text and title
data_frame =
data_frame.select(col("label"),col("textVectorClean").alias("text"),col("title"))

#  label  |  text   | title
# ----------------------------
```

Features

A vector of text is not adequate for training a model. Machine learning works with vector of number, each element is a feature and the number is the value of that feature. So, in order to train our model, we need to convert in some how the vector of words that we got from the tokenizer into a vector of numbers. To accomplish this, Tomas Mikolov, a Google's engineer develops a model architecture for computing continuous vector representations of words from very large data sets[10]. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words (Word2vec, n.d.) Ml library of Spark has an algorithm that implement this solution for us.

Code:

```
# Feature
from pyspark.ml.feature import Word2Vec
feature = Word2Vec()
```

Learning Algorithm

As we have said in the introduction, we are going to use the Logistic Regression algorithm, which will do a binary classification.

Code:

```
# Classification Algorithm
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="label", featuresCol="features")
```

Pipeline

Pipeline in the glue which joins data frames, transformers and estimators. It allows us to automate the process, so we can use it to iterate over and over on it to adjust the learning algorithm. Improving the model in each step, as we can see in the following diagram.

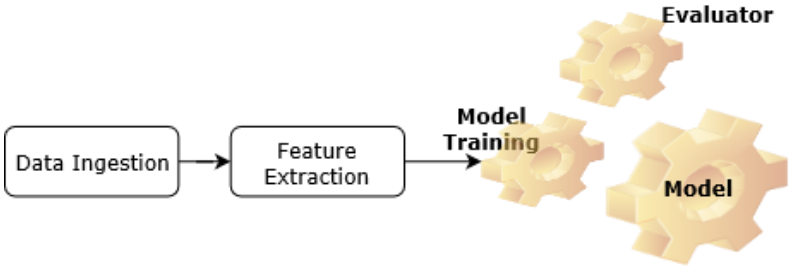


Figure 23: Training Model Pipeline

Code:

```
# Creating the pipeline with the feature and classification algorithms
from pyspark.ml import Pipeline
stages = [feature,lr]
```

[10] Mikolov, T.; Chen, K.; Corrado, G. & Dean, J. (2013), 'Efficient Estimation of Word Representations in Vector Space', CoRR abs/1301.3781 .

```
pipeline = Pipeline().setStages(stages)
```

Evaluation

In order to tuning the model, to improve it, we need to find a way to stablish when a model is better than other. For doing this, we need an evaluator. Our evaluator will be specific for binary classifications and will use the metric of area under ROC (receiver operating characteristic) to score our model. This metric is based on the rate of true positive and false positive. True positive is when the prediction was positive, in our case it is an IT Job, and it actually is an IT Job. A false positive is when the prediction was positive, in our case it is an IT job, it was negative, no an IT job.

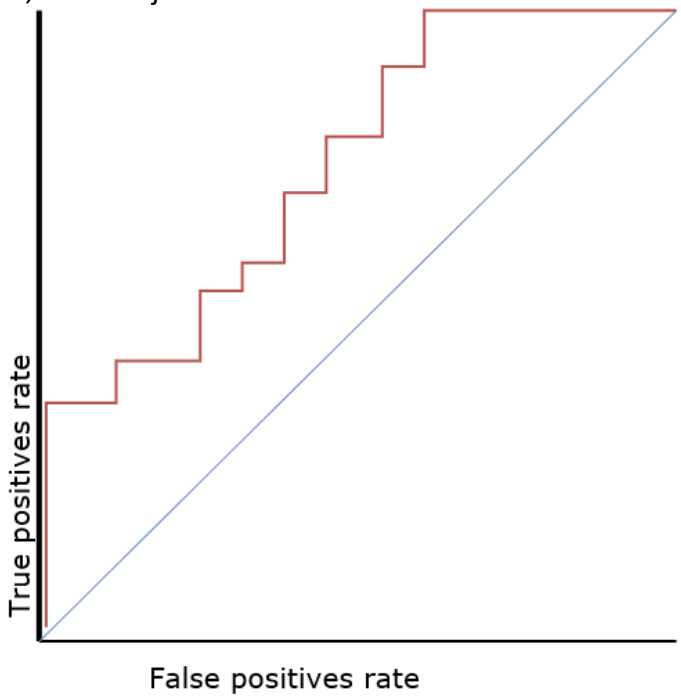


Figure 24: ROC area

The area under the function, in our figure the line in red, it is bigger when the prediction is better. So, we can compare distinct models and know which is better.

Code:

```
# Evaluator. It will work on the prediction column
# we will use the metric "areaUnderROC"
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()\
.setMetricName("areaUnderROC")\
.setRawPredictionCol("prediction")\
.setLabelCol("label")
```

Tuning

One of the most important tasks in Machine Learning is tune the algorithm that we are going to use to get the best result. Each model that we are trying to predict has a different behaviour. The maths that under the hood of the algorithms that use machine learning to predict those models have a series of so-called hyperparameters. Those parameters are not related to the data, they are parameters that adjust the machine learning algorithm to get a better prediction. Our feature algorithm, as well as, our learning algorithm have a series of parameters that can be tuned to get a better fit to our training data. Finding the best values

for those hyperparameters can take a much effort, however, most of the machine learning libraries provide with tools to automatize this task. In spark, inside the library ml there is a module called tuning that helps with this. We are going to use ParamGridBuilder which is used to define the hyperparameter space, we add what are the hyperparameters that we want to tune and the range of values that can take. The class that will do the split of our training data in test and training to perform the tuning will be TrainValidationSplit.
<<Validation for hyper-parameter tuning. Randomly splits the input dataset into train and validation sets and uses evaluation metric on the validation set to select the best model. Similar to CrossValidator, but only splits the set once>>
(Spark.apache.org, 2019).

Code:

```
# Tuning

# Creating the hyperparameter space
from pyspark.ml.tuning import ParamGridBuilder
params = ParamGridBuilder()\
.addGrid(feature.inputCol, ["text"])\
.addGrid(feature.outputCol, ["features"])\
.addGrid(feature.minCount, [3,5,10,15])\
.addGrid(lr.elasticNetParam, [0.0,0.5,1.0])\
.addGrid(lr.regParam, [0.1, 2.0])\
.build()

# Automate tuning to get the best parameters and
# hyperparameters
from pyspark.ml.tuning import TrainValidationSplit
tvs= TrainValidationSplit()\
.setTrainRatio(0.75)\
.setEstimatorParamMaps(params)\
.setEstimator(pipeline)\
.setEvaluator(evaluator)
```

Results

After running the setting up all the functions, we fit the model and get the result.

Code:

```
# Fitting the model
tvsFitted = tvs.fit(train)

## check
evaluator.evaluate(tvsFitted.transform(test))
```

Getting the following result:

```
>>> evaluator.evaluate(tvsFitted.transform(test))
0.9037016735728367
```

Figure 25: Result of training model

So, we have a 90%.[Req:021]
We can save the model to reuse it later in our application.

```
tvsFitted.bestModel.write().save('./modelTrainSplit90')
```

Software Design

First approach

Once we have the model trained, we are ready to develop the application that is going to be run by Spark. We can create our application in Java, Python or Scala. Scala is the native language in which Spark is written, and many developers prefer to use it because of this reason, however, learning Scala is out of the scope of this project. The prefer choice, after training the data model and get familiarized with the API in python of Spark it is to write the application in Python. There is an API developed by lucid works, the same people who has developed Solr to do the integration of Solr with Spark. This API can be used in python. After writing the whole code, we realized that this API doesn't support structured streaming[11]. Structured streaming in the technology that Spark recommended, and is encouraging to developers to use it, as RDD is lower lever data type.

Final approach

We decide don't use the API to integrate Solr and Spark developed by lucid work team and use our own design. In order to connect our application with Solr, we will use the connector created for Java by Apache Solr. The API is called SolrJ.

Class Diagram

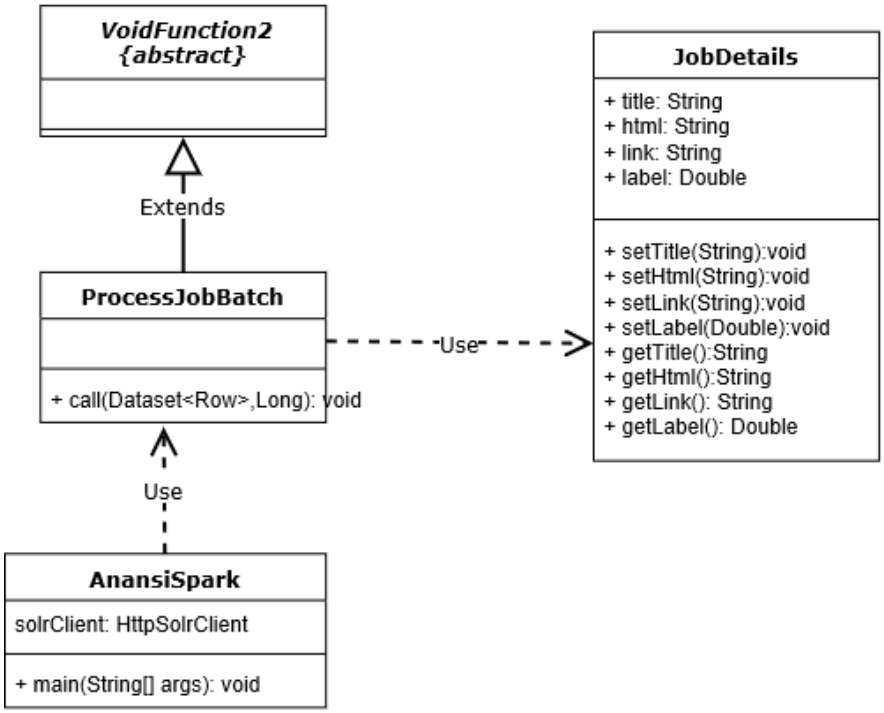


Figure 26: AI Class Diagram

AnansiSpark

In order to create an application that must be executed by Spark, we need to create a main class, which must have a main method. The main class must specify when we launch the application using spark-submit, in case that the class has not an entry point: main, it will raise an exception. The control flow of the program is the following:

[11] GitHub. (2019). Solr support for Structured Streaming · Issue #185 · lucidworks/spark-solr. [online] Available at: https://github.com/lucidworks/spark-solr/issues/185 [Accessed 12 Jan. 2019].

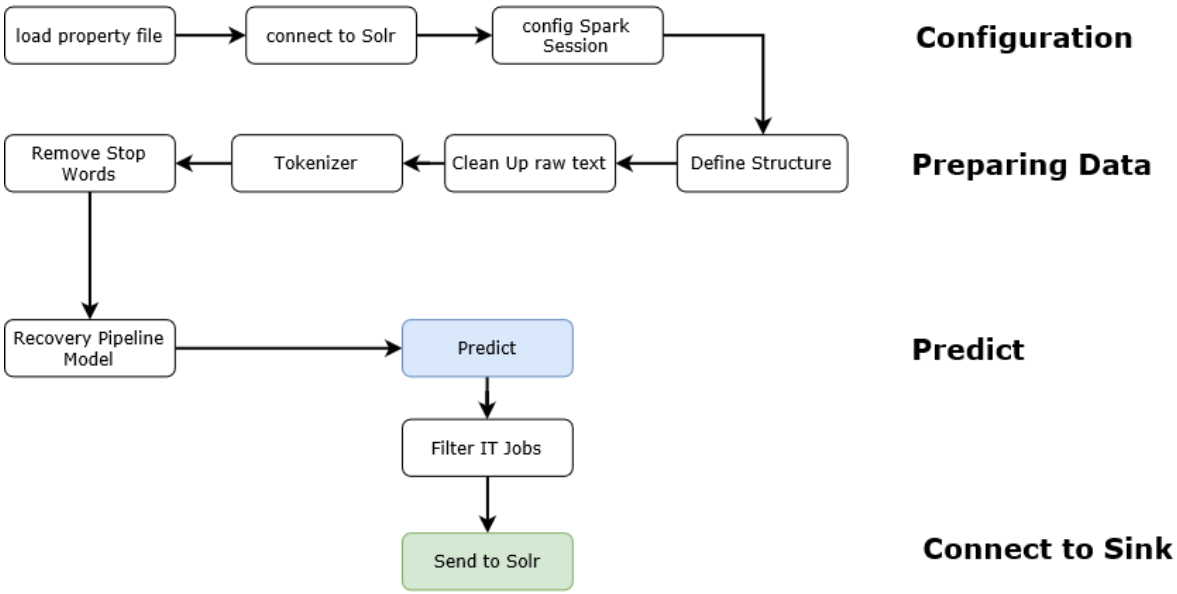


Figure 27: AI Flow Control

JobDetails

This is the Java Bean class needed by Spark to convert a Data Set into an RDD. As we can see in the class diagram, it has four attributes. Each attribute corresponds with one of the columns of our data set, after we have applied the Pipeline Model, and selected only title, htmltext, link and prediction.

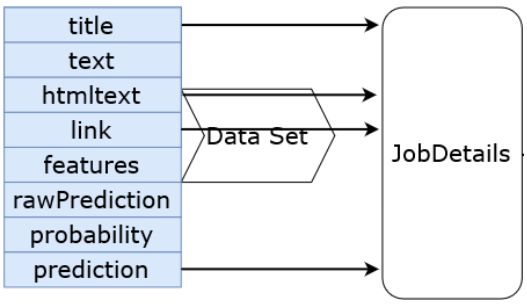


Figure 28: JobDetails Mapping

ProcessJobBatch

In order to be able to select a sink that is not part of the Spark API, we must create our own solution. One of the choices is using the method foreachBatch of the DataStreamWriter. This method allows us to implement a logic which will be applied to each batch of our data set. This method needs as parameter a class which implements org.apache.spark.api.java.function.VoidFunction2. ProcessJobBatch must implement the method *call*, this method is the one that will be called for each batch. The batch will be a Data Set. We will convert this data set in a collection of RDD. For each RDD, we will create a Solr Document, using the SolrJ API, and will send to Solr. We have to do commit, if we want that the document is saved into Solr. We can see the control flow of the program in the following figure.

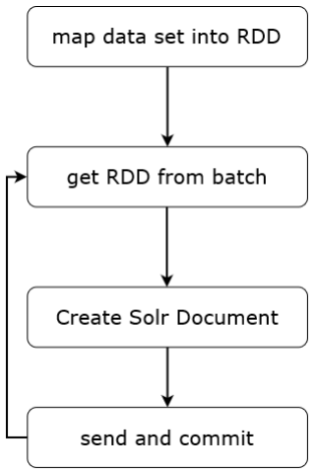


Figure 29: ProcessJobBatch Flow Control

Architect of the system

We can see the whole system and how is integrated in the following diagram.

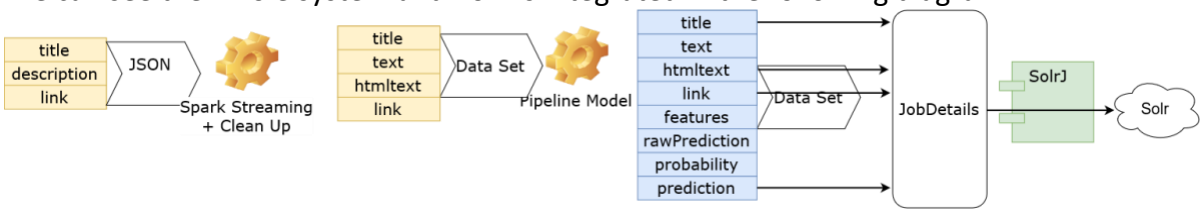


Figure 30: Architect of AI

Spark Streaming is listening the folder of Hadoop for new files, in which the crawler is saving new jobs in JSON format. Every ten seconds (configuration stablished in our code ApacheSpark class), Spark Streaming get every new file and create a new micro batch. It loads every job that is provided as a JSON format into a row of a new data set. It cleans up, tokenizer and remove the stop words in the same way that we described in the Training Model section. Once this has finished, it uses the Pipeline Model that we create during the training to transform the dataset in a new one with, among other, a prediction column. This new column would be 1.0 in case it is an IT Job, and 0.0 in otherwise. The program filters those with the column equal to 1.0, so we keep only the IT Jobs. Finally, we got our dataset with IT Jobs, so it is time to send them to Solr. We use a Java Bean class to transform our data set into a collection of RDD, as each row will be converted into one RDD. For each RDD, that is a Job, we create a new Solr document using the SolrJ API and send them to Solr. This process is repeated every ten seconds if there is any new file into Hadoop folder.

Chapter 6: Search Engine

Requirements

[Req:026]

The system must be fault tolerance, in case that there is any error, it should be able to continue.

[Req:027]

The system must be highly scalable. In case that the volume of work increase, it should be possible increase its capacity increasing the number of resources.

[Req:028]

The system must be able have any connector which allows to integrate with a Java application.

[Req:029]

The user must be able to access to the system through their favourite browser.

[Req:030]

The user must be able to introduce a sentence or set of key words to the system using an interface and perform a search.

[Req:031]

The system must retrieve a list of jobs which match the search performed by the user.

[Req:032]

The user must be able to get a score about the best match, so they can figure out how good was the search, in terms of relevance.

[Req:033]

The user must be able to see how many jobs have been found for the current search.

[Req:034]

The interface to the user must be simple, with the search box in the middle and a button. Below, it will show the list of jobs that match the search.

[Req:035]

The user must be able to go to the source of the job to get more details and apply, in case they are interested in.

[Req:036]

The terms of the search that appears in the text of the match, should be highlighted, so the user can easily recognize them.

[Req:037]

The title of the job should be in the top of the text of the job, so, the user can know what the job is about.

Overview

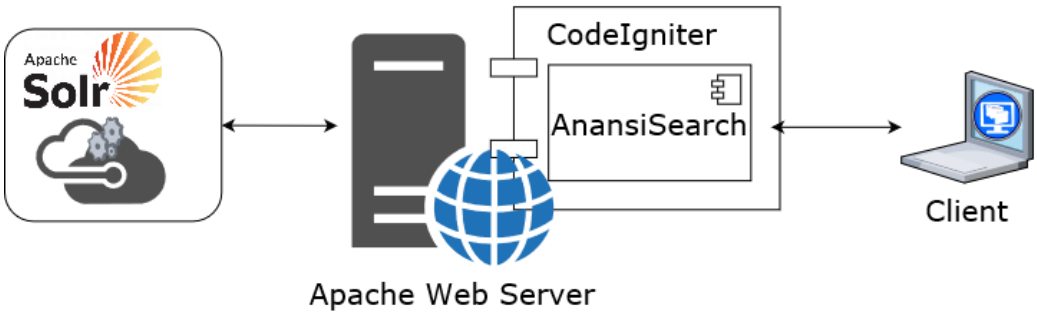


Figure 31: Search Engine Design Overview

Module	Function
Apache Solr	Apache Solr will be the search engine of our system. It will automatically index the text and process the searching. It will calculate the score and retrieve the list of jobs that match the search. [Req:026] [Req:027] [Req:028] [Req:032] [Req:033]
Apache Web Server	This will provide the pages to each browser client which connect to our system. [Req:029]
CodeIgniter	This will be the framework for our PHP application. It will allow us to use the programming pattern Model View Controller
AnansiSearch	This is the application built on the apache web server which will contain the logic of our system.
Client	This module will send each processed job to Apache Solr using JSON format [Req:022]

Software Design

User Interface

User Flows

The first page will be to introduce the search, after clicking on the search button, the user will get the retrieving job list page.

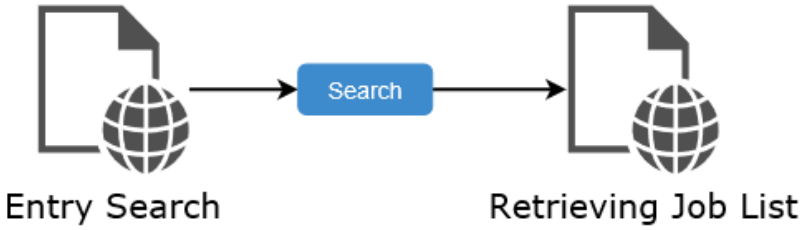


Figure 32: Search Engine User Flows

Entry Search

The user interface will have a box in the middle and a button to do the search. The following will be the look when the user access to the web site [Req:030] [Req:034].

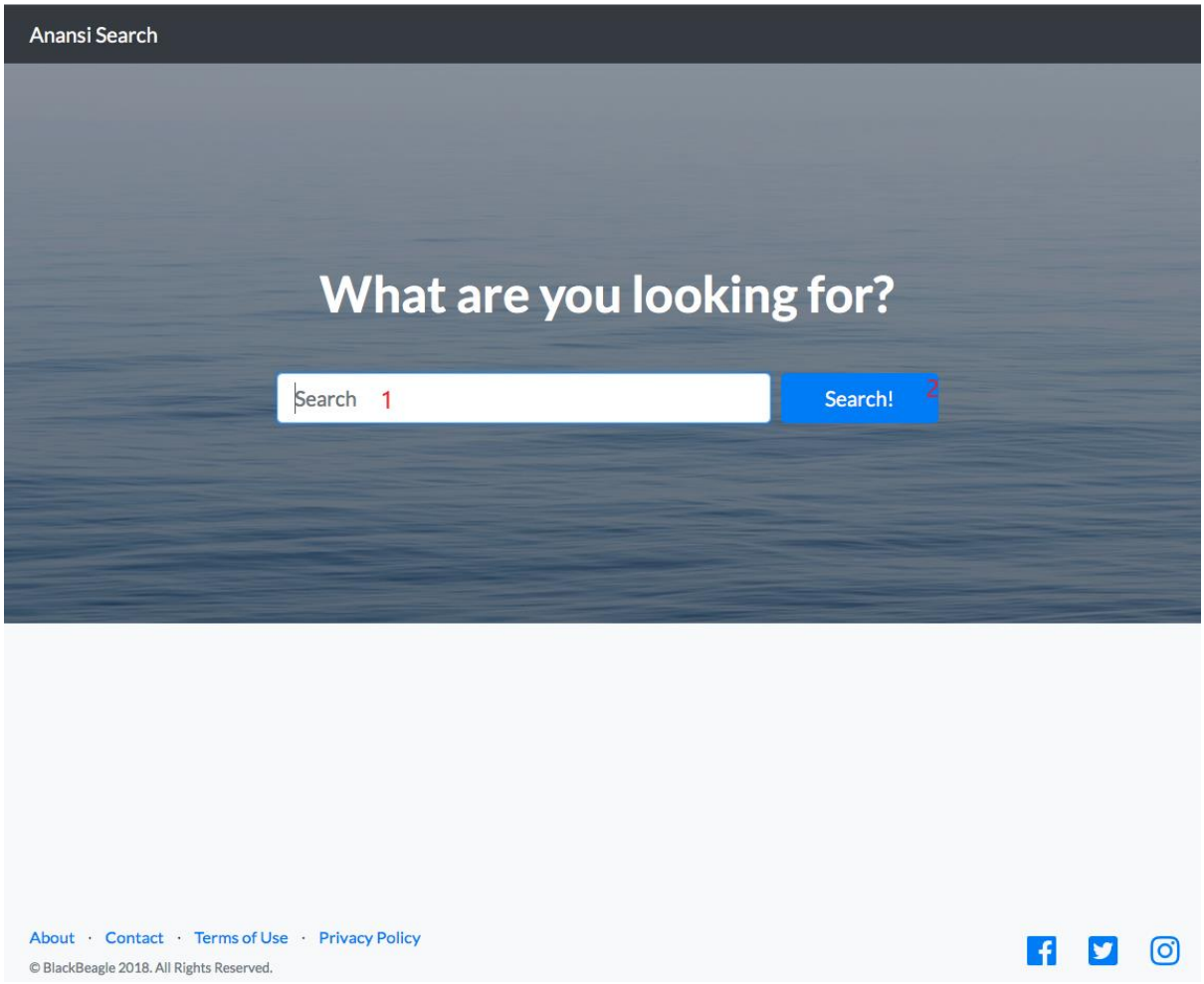


Figure 33: Search Engine Home Page

1. Box to entry a sentence or a set of keywords.
2. Search button. After the user click on the button, the system will perform the search.

Retrieving Job List

After performing the search, the user will get a list of jobs that match their entry.

1. Max score shows the relevance with the search text introduced by the user for the first job shows in the list [Req:032]. This number is coming from Solr and is using the Lucene scoring model (solr-search-relevancy, n.d.).
2. Number of documents found for the search [Req:033].
3. Title of the job [Req:037].
4. Text of the job, with the terms of the search highlighted [Req:036].
5. This list will show a link to the URL of the source of the job [Req:035].

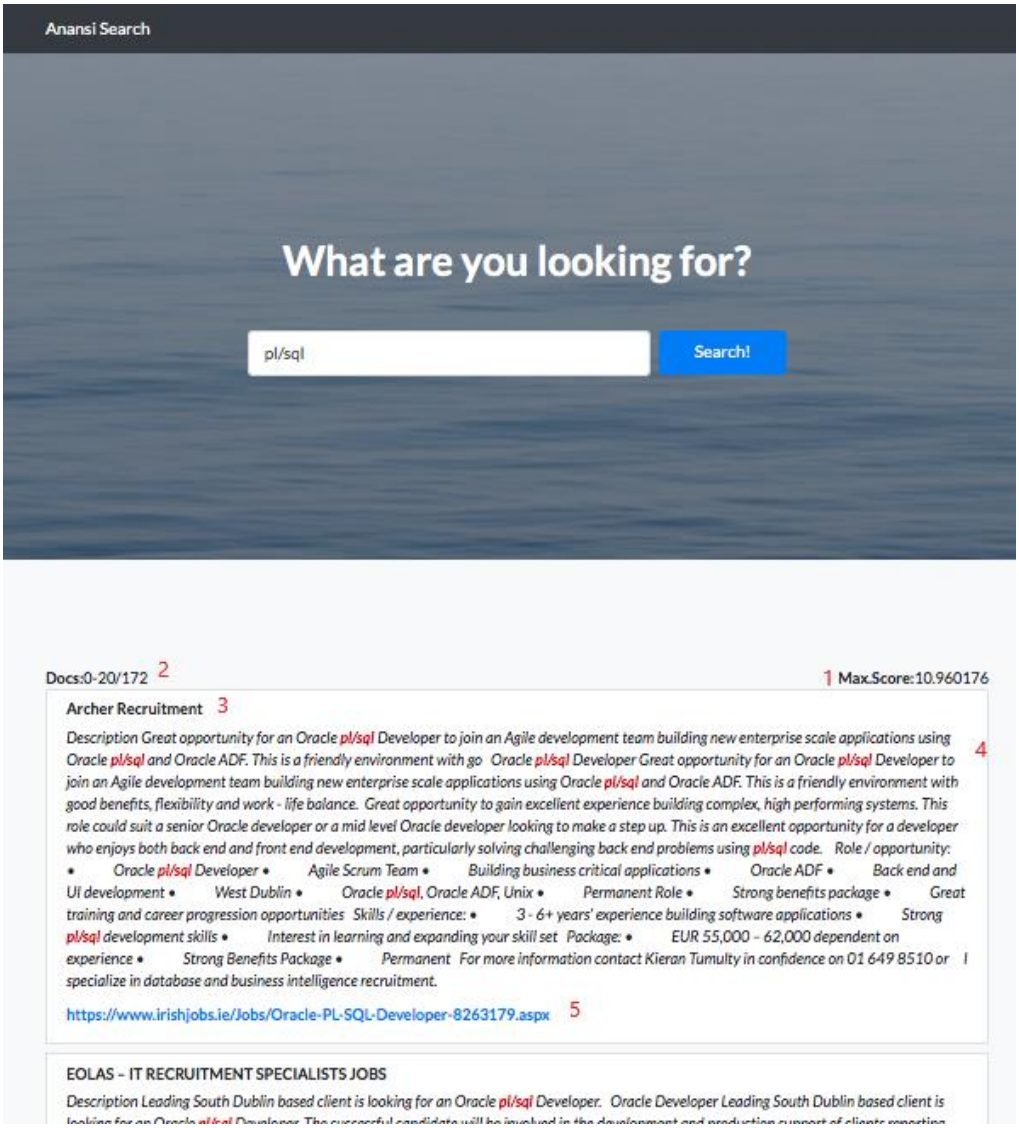


Figure 34: Search Engine Retrieving Job List

Components

We have used the programming pattern Model View Controller. It is not a complex website, as it has no so many components.

Controller

We have two controllers, one which will handle the first status of the page, when the user access to the site. A second to handle the search.

View

We have three views.

- Home.php: the first view when there is not request performed.
- No_response.php: when there is no match
- Response.php: shows the list of jobs

Model

We only have one model, to access the data in Solr: Anansi_model.php

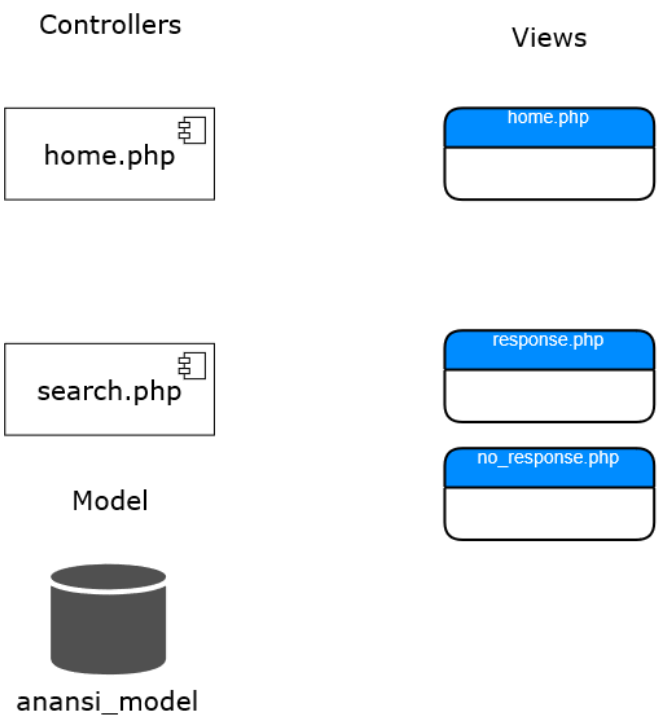


Figure 35: Search Engine Components Overview

Chapter 7: Results and Evaluation

The purpose of this project was to create a portal web, so everyone could be able to search for IT jobs in Europe. Not only that, but to create a system capable to crawl the web using a distributed system, highly scalable and fault tolerance, perform machine learning on those data and send to a NoSQL database in which the text will be indexed. In order to test the system, I have chosen one domain: www.irishjobs.ie. I have extended the crawler to be able to parse the pages of this domain and download its content in Hadoop.

We launch the web to admin the domain our crawler is going to get information from.

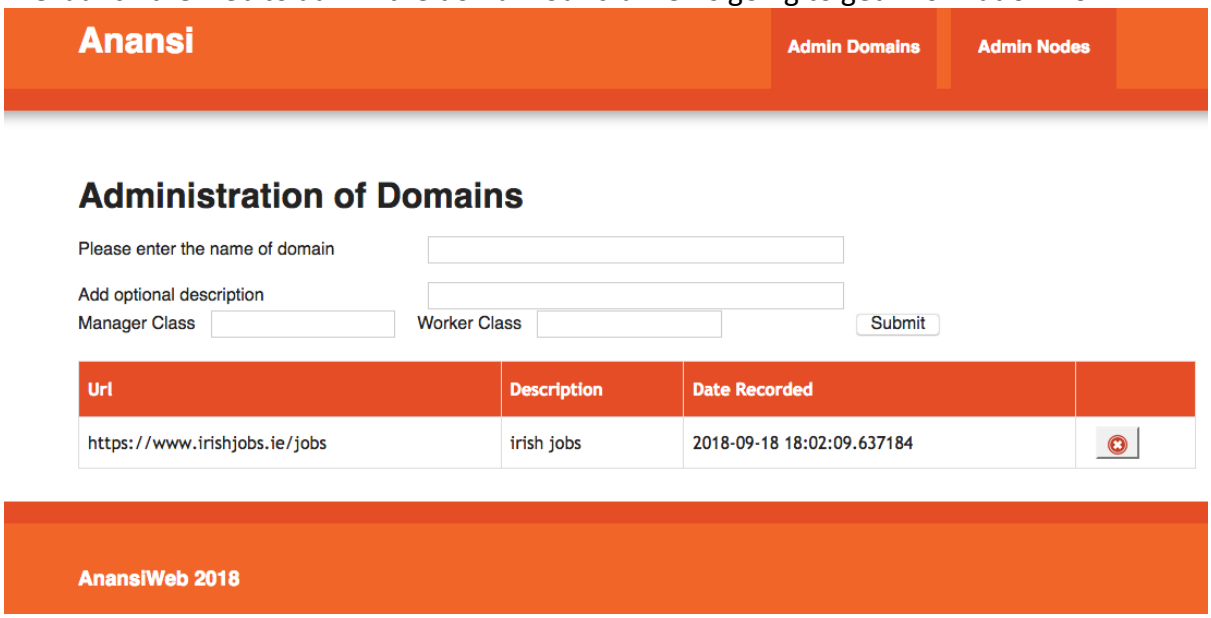


Figure 36: Testing Anansi Web Domain Control

We run the Master, which start to listen for new requests. Every minute, it will retrieve a TimeOut, so we know that is working.

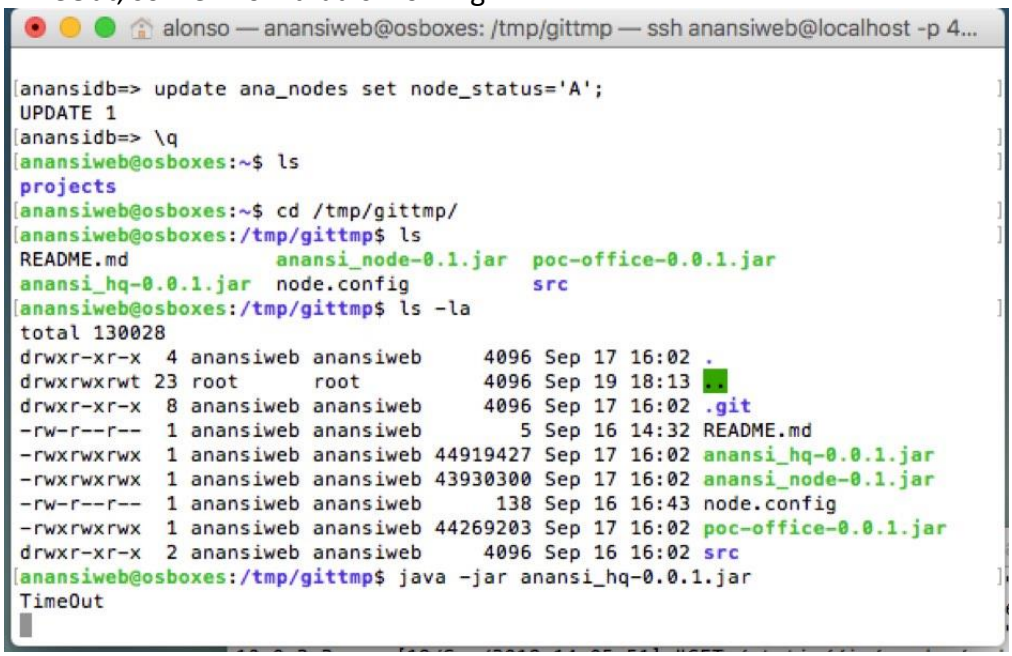


Figure 37: Testing Executing Master

After the node has been registered, we can see the message between one node and the server:

Figure 38: Testing Communication Node

The first message is the Node sending the message to register: INIT|Name of the node|IP
The second is the answer from the server: INIT|12, telling to the node which is its Node Id.
After this, the node send a message REP|NodeID|4|4. We can see as the node has received correctly which is going to be its identified for communications: 12, and it is indicating its capacity: 4, so it has 4 teams working. This node can do 4 jobs at the same time, and with the last number: 4, it is telling to the server that it has no jobs currently, so, the server can assign 4 jobs to it.
After that, the server send a job: JOB|12|…. The JOB command, followed by the ID of the node, the url domain, and the classes.

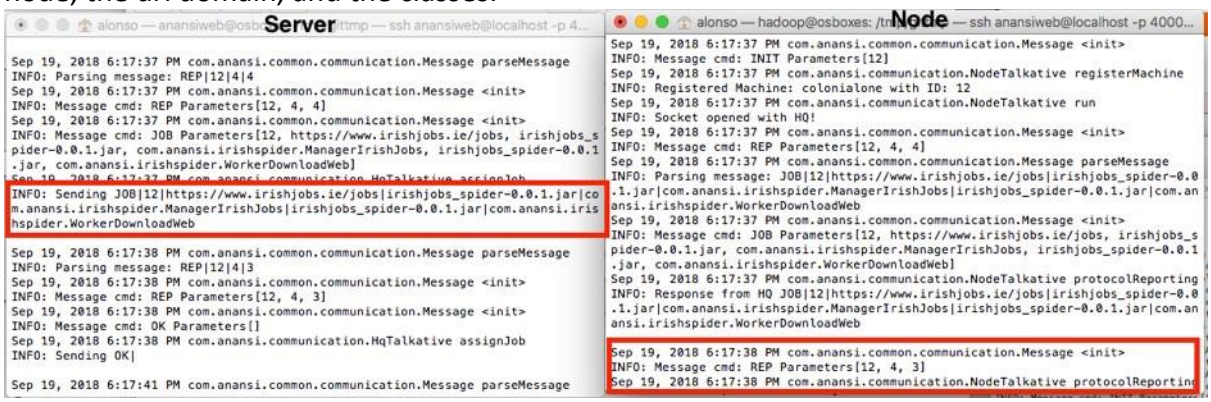


Figure 39: Testing Communication Master-Node

The Node receive that, and update its spare capacity, telling the server that REP|12|4|3, it is the Node with id 12, it has 4 teams, and it can take 3 jobs more.

We can stop the server and the nodes, with a telnet connection, sending the command STOP. As the connection is with format ASCII, we can communicate using this kind of tools.

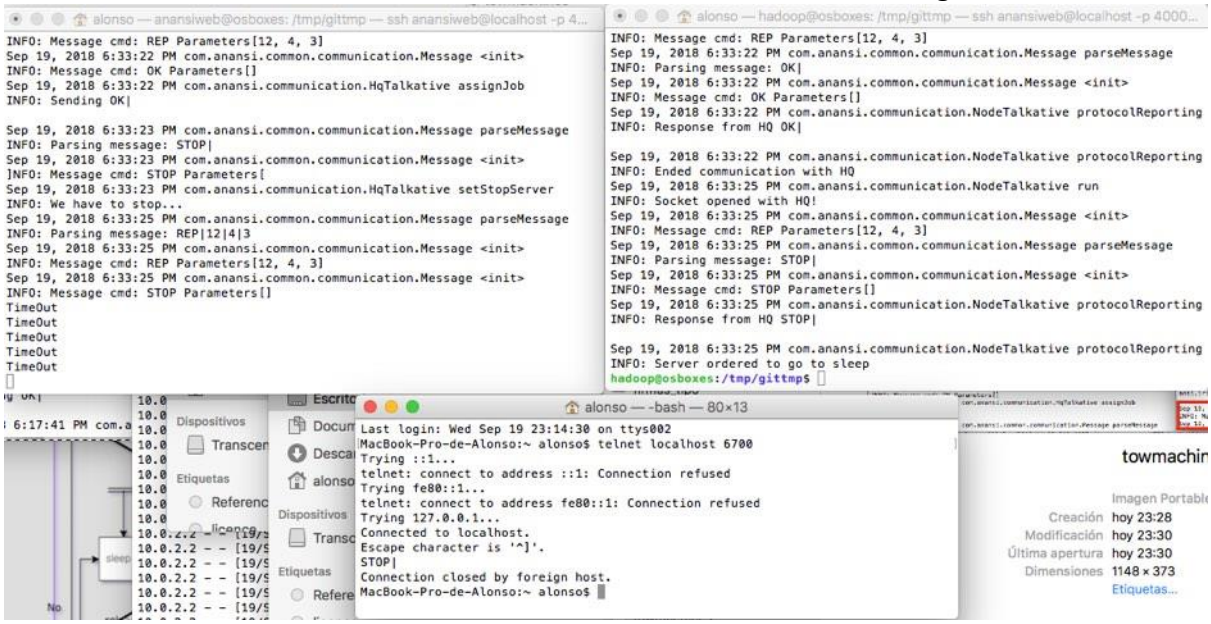


Figure 40: Testing Stop Command for Server

The server receives the message, and then, it will communicate to the Nodes, when they send the status report.

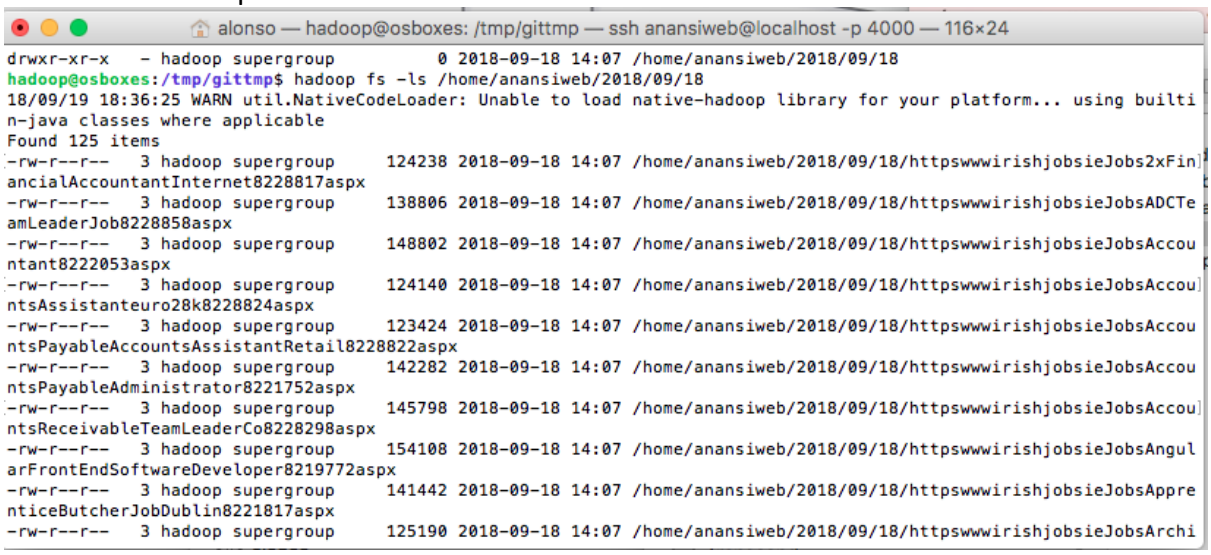


Figure 41: Testing Hadoop Integration

After the node has finished, we can see how the documents have downloaded and saved into the HDFS file system.

Then we launch Spark Streaming which will start to check for each new file that is the folder. After being running in background for a while, we check Solr:

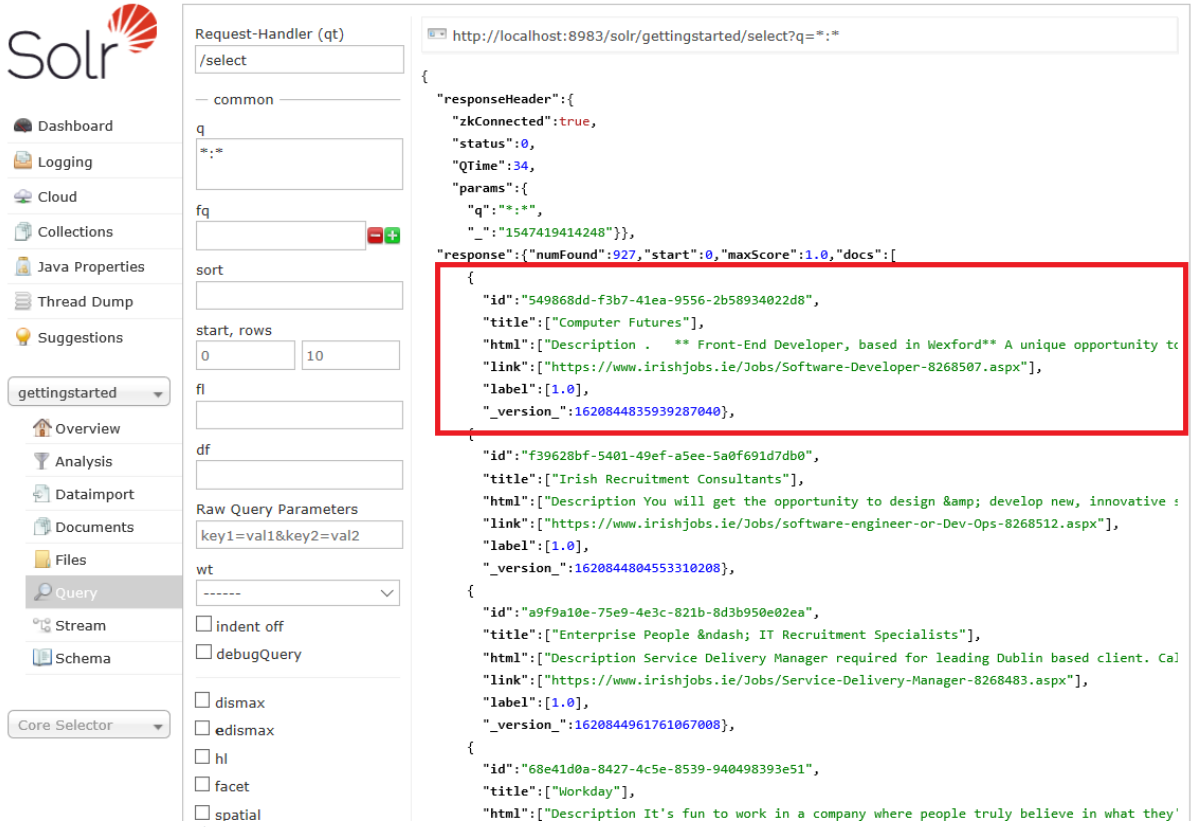
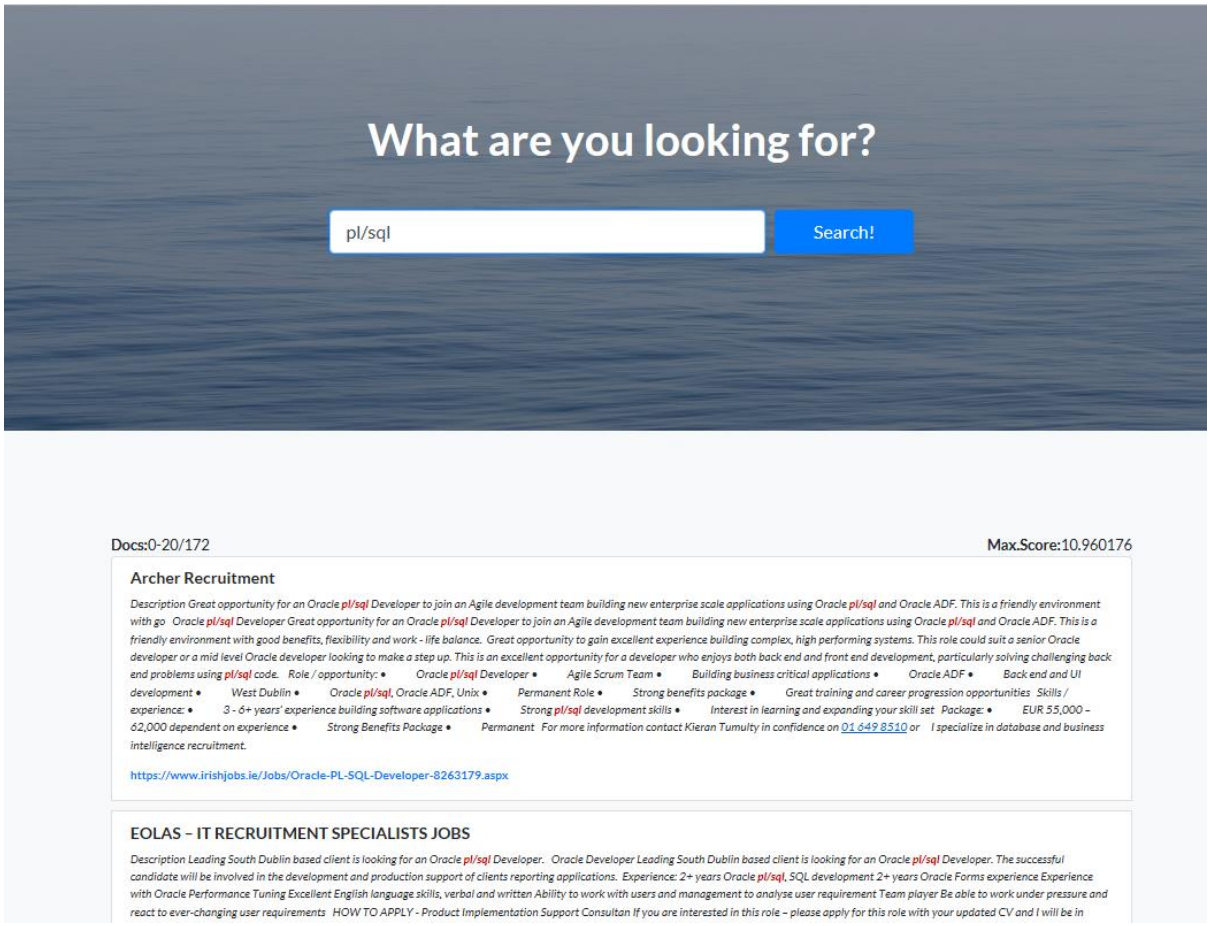


Figure 42: Testing Solr

We can see that documents have been loaded for each job, with the JSON structure.
We go to the searching web and introduce the keywords pl/sql:



It retrieves 172 documents.

The system is working, as expected.

Chapter 8: Conclusions and Future Work

In order to get a system such, I have designed, I have analysed different approaches, and technologies in the market. All the technologies that I took into account must be open source and free of licence. The target of the project was too ambitious, trying to get a portal which gives users statistics for different languages, considering security, and so on. However, the main objective that was put in place an application able to collect information from the web and make it available to the user through a web, has been accomplish from end-to-end. Even, when spark, solr and other technologies are giving so good results in production environments, it seems to me that there is a gap on its capabilities of integrations. There is no so much documentation about then, and most of the knowledge are in forums and Stackoverflow.

For future work, there are many possibilities. The system is working only for one web, and this web has jobs in English. One of the first things would be training a model to be able to distinct between an English job and a job in another language. After getting this done, it would be possible to start creating parsers for web sites in other countries of Europe, so we can filter only those in English.

Another thing would be doing a multi-label classification of the jobs with languages that are required for that job position: Java, Pl/SQL, SQL Server, Python, Scala, and so on. Doing that, would be possible to Solr do search by languages and technologies. Also, it could be possible to connect Spark to Solr, this time to read from Solr, and analyse what technologies are trendy, and adding an attribute with the country, classified most important technologies for countries. As you can imagine, there are a lot of possibilities, once we have a system like this working. The extension and future works are huge.

Chapter 9: References and Bibliography

Lerdorf, R. (n.d.). *sitepoint.com*. Retrieved from https://www.sitepoint.com/rasmus-lerdorf-php-frameworks-think-again/

solr-search-relevancy. (n.d.). Retrieved from solrTutorial: http://www.solrtutorial.com/solr-search-relevancy.html

Spark.apache.org. (2019, 01 01). Retrieved from https://spark.apache.org/docs/1.6.0/api/java/org/apache/spark/ml/tuning/TrainValidationSplit.html

Word2vec. (n.d.). Retrieved 1 12, 2019, from Wikipedia: The Free Encyclopedia: http://en.wikipedia.org/wiki/Word2vec

xAppendices

Crawler

Common libraries

anansi_common-master.zip

Node Code

anansi_node-master.zip

Master Code

anansiHQ-master.zip

Anansi Web in flask to administrate domains

anansiweb-master.zip

Classes to parse irishjobs [extended Manager and worker classes]

irishjobs_spider-master.zip

AI

Code in python to train the model

training_model.py

Code to run in Spark to classify jobs and load them into Solr

jobs-spark-master.zip

Search Engine Application

Code using CodeIgniter for the web to perform searching

anansi_search-master.zip