Hippo

finding business ideas using AI and big data analysis

Architecture Document

Document revision 3.0

10 June 2018

Client

Juicy Story

Teaching Assistant

Hichem Bouakaz

Team

Levi van Rheenen

Jean Paul Donovan Meijer

Said Faroghi

Natalia Karpova

Thijs Baksteen

Andreea Glavan

Sardor Khashimov

Table of contents

Introduction

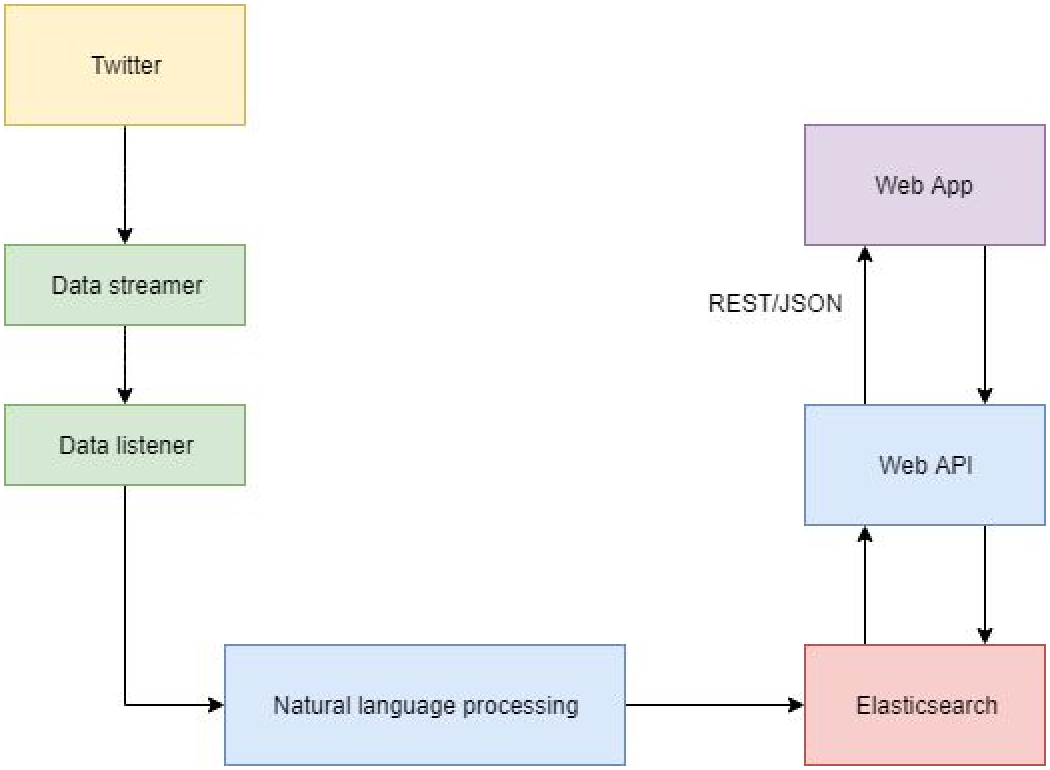
Hippo is a web based application for entrepreneurs. Entrepreneurs can search for keywords and find tweets that display desire that have to do with those keywords. In order to do this, we need to get data from twitter, filter it, be able to search through it, and display it on a website.

We split the project into three separate parts: front end, data processing, and back end. The front end will focus on the interface of the website and user interaction. It will get its data from the back end, which is going to focus on the data delivery. The data processing part will input data from Twitter, filter it, and process it. The back end will search through the data gathered from this processing part.

This document will delve deeper into these parts. It will go into the implementation and design details, and also give some insight on the decisions made.

Architectural Overview

The application consists out of three major components, that are data analysis, the web API and the web application. Data analysis can be divided into three smaller components which all have their own function. The architecture of the other two main components is more trivial, and are both eventually just single applications. Furthermore, both data analysis and the web API will make use of the database system Elasticsearch.



The architecture of data analysis can be split into three components. The data streamer connects to the Twitter streaming API and the data listener sends the new data received to natural language processing. This then processes the tweets, using natural language processing techniques and stores all the relevant data in Elasticsearch.

The web API provides functionality to search, categorise, and present the data that has been analysed. This is initiated by calls from the web application which runs in the users internet browser. The web application provides an interface the user can use to interact with any data stored on the platform. The web API and the web application communicate over REST where content will be encode in JSON format.

Connections between the database and the components will be done using the official client libraries of Elasticsearch, elasticsearch-py and elasticsearch_dsl.

Technology Stack

Data analysis

The data will be captured in real time form Twitter, the (real time) streaming API is less restricted by rate limiting. Practically eliminating this burden and allowing us to access a much greater variety of data to search through. This data is captured by a small application written in Python. The raw data from Twitter is will be filtered using natural language processing, then saved directly in Elasticsearch.

For data analysis part Python programming language together with NLTK (Natural Language Toolkit) platform will be used. Natural language processing will be done in several steps. This is a preliminary layout of an analysation process and may change during the later development and optimization of Hippo application.

Both users search queueries and tweets themselves will be analyzed using nltk. As the first step, keywords extraction will be performed on user queueries. This will include several steps, namely, tokenization (the exact tokenizer will be determined later by experimenting with the data) of a queuery, POS-tagging (part-of-speech tagging), named entity detection (using, presumably, chunking, tag patterns and chinking) and relation detection (at the early stages it would be done with rule-based system). WordNET will be used to identify the exact meaning of keywords and to find synonyms to improve consequent search.

The same keywords extraction will be performed on tweets. What is more, the acora Python library will be used to extract other synonyms based on the keywords received. In case of tweet analysis, relation detection part will be subjective to particular search query entered by a user. We also consider to make tweet filtering based on their similarity to each other so that user will get a bunch of different ideas as an output.

Pre-fatorial strategy is following:

Search query by itself may act as centroid document. Each tweets that passes previous “filter” , in other words, each tweet which keywords are related to search query keywords, will be treated as a separate “document” and will be compared to centroid document and other “documents” (tweets). This will be done with a version of CSIS (Cross Sentence Information Subsumption) and will result in exclusion of identical ideas.

Front-end development

The front-end side of the project will mostly use Javascript (in conjunction with HTML & CSS) to build the web app. The choice of Javascript, HTML, and CSS is pretty clear, since they are standard in web development. What is more, as the initial web app is pretty simple, containing just searching through ideas, this provided a good starting point. As extra functionality was added, this choice wasn't changed due to the team being familiar with it, as well as its' efficiency.

We will use two Frameworks - Vue.js with webpack for JavaScript and Bootstrap for CSS. Vue.js is currently one of the industry's favorite frameworks because it has clear advantages over others (Angular, React), and is simple to learn and code in. Vue.js will also be coupled with the third-party Axios, a promise-based HTTP client library for REST api consumption. Vue had "vue-resource" that was able to do this, but since Vue 2.0 this has been retired, and the Vue.js + Axios combo is the most recommended setup.

Bootstrap was chosen as the front end framework due to its' fast prototyping and many templates, as well as the extensive components. What is more, the popularity of Bootstrap means it is easy to get started in, and there are many learning resources available.

Back-end development

For the back-end, we will use Python with Flask. Python is chosen because most of the team is familiar with the language, and it serves our purposes well. Flask is a microframework for Python, meaning that it provides a core framework that is relatively small, but extensible with a variety of different libraries. This avoids the clutter one would have with a larger framework that includes functionality that is not needed for the application.

Elasticsearch will be used for our user searching needs. Elasticsearch is an open source search engine/database that can store the many ideas we collect, and allow search options to find ideas that the user wants.

We will use Docker to support our architecture, and provide secure containers for the different components of the back-end to run in. Docker will also simplify the process of scaling the application to more servers if needed in the future.

Team Organisation

We chose to split into 2 teams: front end and back end, with 2 and 5 people, respectively. The reason for not choosing a 3:4 split was due to the fact that the front end design is on the minimalistic side, based on the demos we have seen.

Our front end team consists of Said Faroghi and Sardor Khashimov, and our back end team consists of Levi van Rheenen, Jean P.D. Meijer, Natalia Karpova, Thijs Baksteen, and Andreea Glavan.

As the names indicate, the first team will be in charge of the front end which includes the web design, implementation of this design, creating a logo, and maintaining the site.

The back end team is in charge of data processing and web back end. The back end focus is on data processing and filtering of ideas and categories, as we believe this is the key to achieving the end goals of this project.

The data processing part includes creating a data streamer to crawl Twitter for ideas, updating the database with the received ideas, calculate each ideas' stat points, filtering these ideas into their respective categories, and filtering ideas which are the same but worded differently into the same database entry.

The web back end is in charge of communicating with the front end and sending ideas, plain or in the form of sets/categories, from the database based on the given key words. What is more, the web back end deals with users' accounts and stats functionality.

Change Log

The record of the changes made to the architecture document tagged with date and author.

Author	Date	Description
Jean P.D. Meijer	10-03-2018	Create the architecture document and layout the structure.
Andreea Glavan	10-03-2018	Updated the team organisation section.
Natalia	11-03-2018	Added section about the data analysis tech stack.
Said & Sardor	12-03-2018	Added description for front-end tech stack.
Levi	13-03-2018	Added introduction.
Andreea Glavan	13-03-2018	Reviewed and updated existing sections.
Jean P.D. Meijer	13-03-2018	Added the architecture overview section and extended the data analysis tech stack.
Thijs	13-03-2018	Added description for back-end tech stack.