Tamahau Brown (https://github.com/TamahauBrown)

PROJECT SCRAPER

Contents

Tamahau Brown (https://github.com/TamahauBrown)

Introduction:

Purpose:

Python Reddit Twitter, Facebook, etc… scraper then check if scraper API does not exist then grab from HTML of page

Main Goal:

To integrate Python for scraping with Rust to create a command line interface which will be improved into a front end interface.

Python's Job:

Use API call to generate information and pass in a way that Rust can accept the information.

Rust's Job:

Use the data and display the data to the front end.

Languages:

Front end: Rust

Back end: Python

Intended Audience:

The project is designed to be used by people of all skill levels

Permissions:

Everyone will have the same permissions as this is meant to be a project for 1 user with no permissions or access.

The information will be stored into a PDF file with the relevant information with graphs and other data structures but the data itself will be put into a data spreadsheet (examples: .xlsb, .csv, .xls, xlsx, xslm)

The project is designed to be used by people of all skill levels and since this will be intended for private use where the user will insert their specific API key to be able to access the information.

Tamahau Brown (https://github.com/TamahauBrown)

Everyone:

- Be able to choose the scraper of choice
- Insert developer key
- View interface with relevant information (in file for the console version)
- View database (console version only)

File naming conventions:

The file naming convention will be the title of the page in which the user provides e.g.: “Final Fantasy 14 Reddit.pdf” and “Final Fantasy 14.csv” this way the user will be able to reference the information quickly, whether we make a folder and add these files within it is a discussion for another point.

Document Conventions:

Not applicable

System Features (Functional Requirements):

Note: All SF’s mentioned within this section are component-specific and quantitative requirements.

SF1: Information Scraper

The user inputs the scraper of choice and the API key to be able to access the information. The system stores the information for all uses on this device and extracts information.

Example Scenario:

“Hello, where are you planning to gather information from?

1.: Reddit

2. Twitter

3. Facebook

4. Other”

The information scraper takes in the API key **if** one of the official options are chosen and **if other is chose**n asks for a link to the page and extracts the HTML of the page and the relevant information.

Case 1: An scraper with an API key is chosen

Front end:

“Can you please provide the API key, Client Secret, etc…” (varies from API to API depending on the libraries we use)

Tamahau Brown (https://github.com/TamahauBrown)

Back end (Python information):

Extracts information regarding the API and takes the relevant call, turns it into a **Pandas Dataframe**, the users will then be **asked for a specific file extension** of their choice (refer to Permission subsection) then extracts the information into a file to reference for later use.

Example prompt:

"Please provide the file option you wish to use

1: .xslx

2: xlsb

3: xslm

4: xls

5: .csv"

Two potential options:

A: Rust route to extract information and transform the data. This could be done with a dataframe library of choice and a graph library of choice.

B: Python route using Pandas and a graph library of choice (example py_graph note: could potentially be deprecated)

Case 2: Information from gathering HTML

This will be a little more complicated as this involves gathering the relevant HTML information of each page. Potentially this is where you would train a scraper to be able to recognize information about a page and gather it (if a library does not already exist) and use the information to determine the key words within a sentence.

The user will provide a link to potentially a forum group or a specific page which the system will need to be able to adapt to.

Example scenario:

"Please provide the link to the information you wish to access:"

After the user has provided the link, the system will check for **whether the link is relevant**. If the link is valid, if it is then it begins the information scraping process.

Discussion around training a model and plan of attack (potentially scrap this section)

SF2: Graphing the information

Once the information has been gathered and stored in a relevant file the graphs will be shown depending on what the decision was in section 1 and stored within a PDF file within a folder location where the user is calling the command.

SF3: Amalgamating Large Data to produce high quality graphs

If a user has accessed sufficient information, they may wish to create their data into a large sum and find the common patterns within a file. E.g.: Most common word used in all phrases OR most common keyword in a phrase. And displaying this to the user and **using SF2** to display and store the data. **SF1 is optional with this step.**

Example of how this would be done:

"Hello, please add all the following files you wish to merge into to be able for important information. Alternatively you can give the name of the folder with all the relevant information"

After user has input information:

"So to be clear these are all the files you wish to add? F1, F2, F3, etc…" OR "So to be clear these are all the files you wish to add? Folder"

If user inputs multiple folders or files that are not the accepted files (see permissions):

"I'm sorry but we can only accept one folder, please add all files into one folder and ensure there are no subfolders"

LIMITATION:

This will not be able to delete a file but will incorporate a process which will allow you to restart the process which can be frustrating.

Multiple folder amalgamation could be messy so will not be included in this project.

Techniques that will be relevant:

Gradient Descent with feature scaling on multiple features.

Source to learn Machine Learning (Course is free)

Tamahau Brown (https://github.com/TamahauBrown)

Non-Functional Requirements:

Safety:

Incorrect API keys are a reality that is bound to happen and with the application storing the information, ensuring that people are able to delete the API key without removing all relevant information is important.

Security:

Users are entering in their specific API keys and this information is stored, a convention needs to be secure using encryption to securely store and access their data to protect the integrity of their information.

Performance:

Main things that would break the system would be to prevent overloading the application with excess information. This would be applied using a performance monitoring application (JMeter, tutorial can be found here) and determining the competence of an application and documenting the results.

Design:

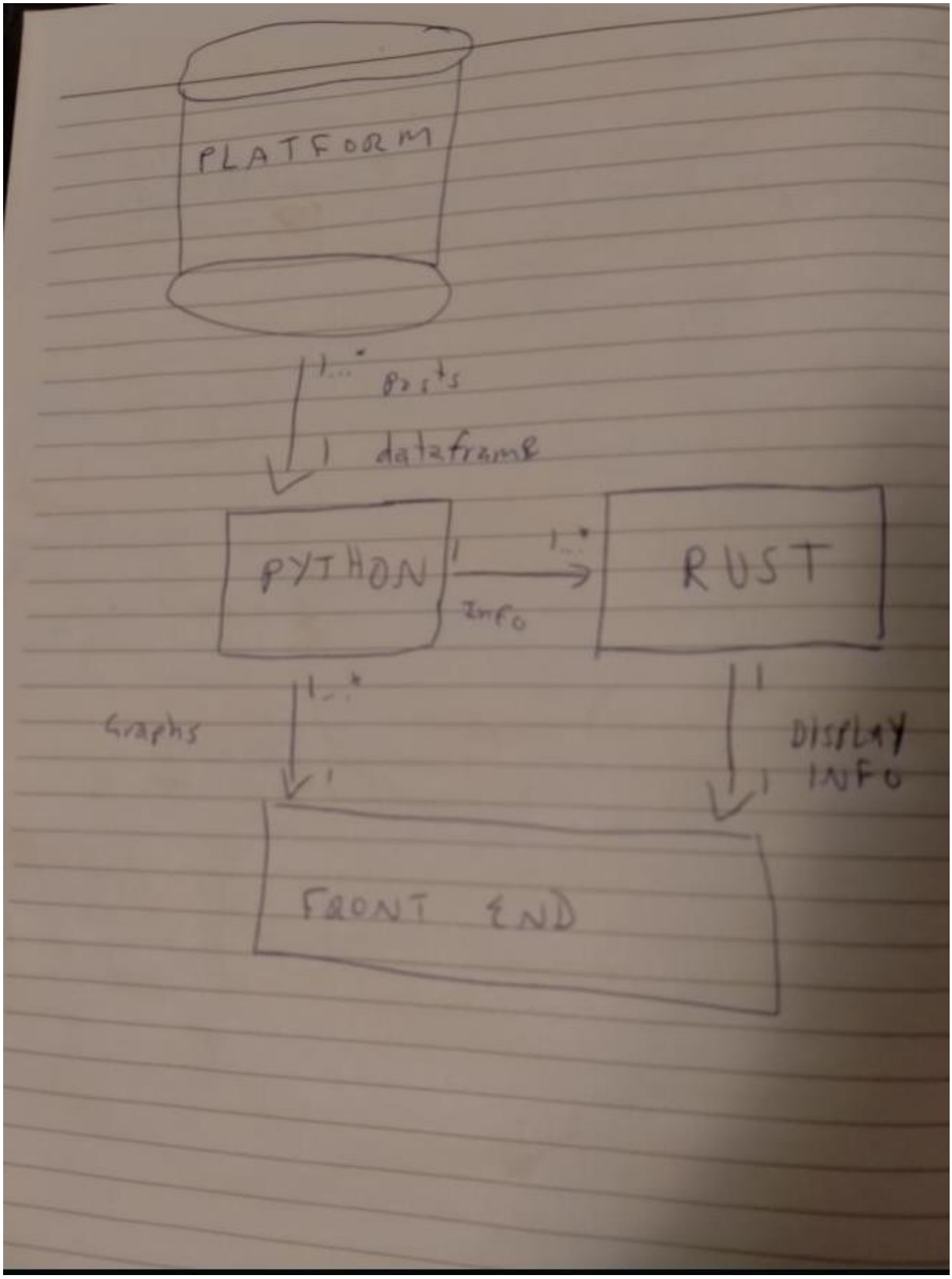
Constraints:

The main constraint is due to this application being reliant on the free use of APIs such as: Twitter, Facebook, etc… the information can be outdated with an API update which means maintenance needs to be considered.

The second main constraints is due to the project having no funding this will involve trying to avoid technologies which will have costs which limits the strength of the IDEs and testing environments we will have access to.

Tamahau Brown (https://github.com/TamahauBrown)

System Architecture:



Paper Prototype:

Tamahau Brown (https://github.com/TamahauBrown)