Intelligent Scraper Python Design

JIE Team 1350

Hadiya Kodvavi, James Carpenter, Mehmet Islamoglu, Zoe (Phuong) Le, Thao Tran, and Thinh Pham

Client: Timothy Brown

GitHub: https://github.com/JIE-1350/ISpy

Table of Contents

List of Figures

Terminology

API	Application Programming Interface (API) is a software intermediary that allows two applications to talk to each other.
Backend	The part of the application that is hidden from the user that is responsible for operating the application.
CSV	Comma Separated Values is a delimited text file used to store multiple fields of data records
Data filtering	The process of choosing a smaller part of your data set and using that subset for viewing or analysis
Data Scraping	Data scraping is a technique to extract unstructured data from a website or application to into structured data.
DataFrame	A python object created by Pandas that store two-dimensional and potentially heterogenous tabular data
Electron	A desktop graphical user interface application using web-based technology
Flask	Flask is a Python web framework that we use as an API for the backend
Frontend	The graphical user interface of application
JavaScript	Programming language that we used for our Frontend
JSON	Also known as JavaScript Object Notation, JSON is a file format for data transfer that uses human-readable text to store and transmit data objects
Material-table	Prebuilt react component for the table in the search tab.
Material-UI	JavaScript library that contains pre-built React components.
Node.js	Node.js is a JavaScript runtime environment that executes JavaScript code outside a web browser
Pandas	Data analysis and manipulation tool for Python
Python	Programming language that we used for our Backend
React	JavaScript library that we use to build components of the user interface.
Redux	Global state manager the frontend
REST API	An API that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services

Twint	Python library that we use to scrape data from Twitter without the use of Twitter's API
XLSX	Microsoft Excel Open XML Spreadsheet (XLSX) are files that can be opened using Microsoft Excel

Introduction

Background

Team 1350 is working to create an intelligent Python-powered scraper that will be able to extract and process data from Twitter. Data scraping is a relatively new, evolving phenomenon in which we use automated tools to collect large amounts of information. Our client, Timothy Brown, requested a big data tool that allows for data accumulation, enabling the ability to manipulate and interpret the collected data. Our main goal is to provide an easy user interface allowing for a decreased time in searching the web and more time studying and analyzing trends. The collection of information will enable various users (such as social scientists and academics) to view top trends, identify solutions, and perform insight analysis.

Looking at the significant impact social media has had on the 21st-century global society, we can see how easily and quickly misinformation can spread through platforms such as Twitter. This misinformation can lead to a significant amount of divisive and harmful opinions. For example, the past years of the global pandemic have shown how misinformation such as conspiracy theories about COVID-19 vaccines can quickly spread and lead to harmful effects like preventing many people from getting vaccinated.

Using our Python scraper, users will be provided with accurate and unbiased data; as a result, allowing for the common benefit of society. Some of the key features within our application will be collecting data from Twitter, applying filters to sort and organize data, providing various insights on collected data, and saving collected information for later use.

Document Summary

This report documents the Intelligence Scraping with Python (ISpy) project in four main parts:

- **System Architecture**: The System Architecture section describes the static and dynamic elements of our system's frontend and backend.
- **Data Storage Design**: This section describes the data format of our application that will be saved locally or on Google Drive. This includes data scraped from Twitter, user settings, and user search history.
- **Component Design Detail**: This section describes the relationship of components in the frontend and backend and how they interact with each other.
- **UI Design:** The UI Design section presents the current state of the two major UI screens that our users will interact with (the search and insight page), as well as our plans to iterate on their design in the sprints to come.

System Architecture

The system architecture of this application was chosen so that each object in the system is only responsible for one task. This is evident in the fact that there are two layers in the application. The frontend of the application is responsible for the parsing user inputs and making sure that the correct instruction is passed to the backend to do the actual work. Similarly, within each of these layers, components in both the frontend and backend only have a single responsibility. For example, electron was used to convert the React application into a desktop application for any operating system. Similarly, Flask was used to manage communication from the frontend via RESTful API. The architecture of system is designed so that each part of the application is responsible for a different task. When similar functions need to be update, it is easy to localize the changes, and it allows the application to be robust to changes.

To better understand the structure of our Intelligent Scraper Python, we have displayed the overall system architecture below in **Error! Reference source not found.** and Figure 2. These diagrams represent a high-level description of the structural components of the scraper and the relations between them. The application is defined by four core functional components. First is the frontend which the user interfaces to perform data collection or process insights. This component of the application allows users to view the data in an organized format, allowing for an easy user interface. Next is the backend which is used to carry out all the functions. This component of the application performs the scraping of data from Twitter amongst many other functions. The local storage component of the application connects to the backend to allow users to perform insights on previously saved data searches. Lastly, the most important component, Twitter, which is connected to the backend component of the application, from which we collect our data from.

Static Elements

The static system architectural design shown in ***Error! Reference source not found.*** displays the functional view using the four major components of the application. The diagram provides a visualization that could be used to aid in understanding how each element of the application is connected to each other. The direction of the dependency arrow from X to Y indicates that X depends on Y.

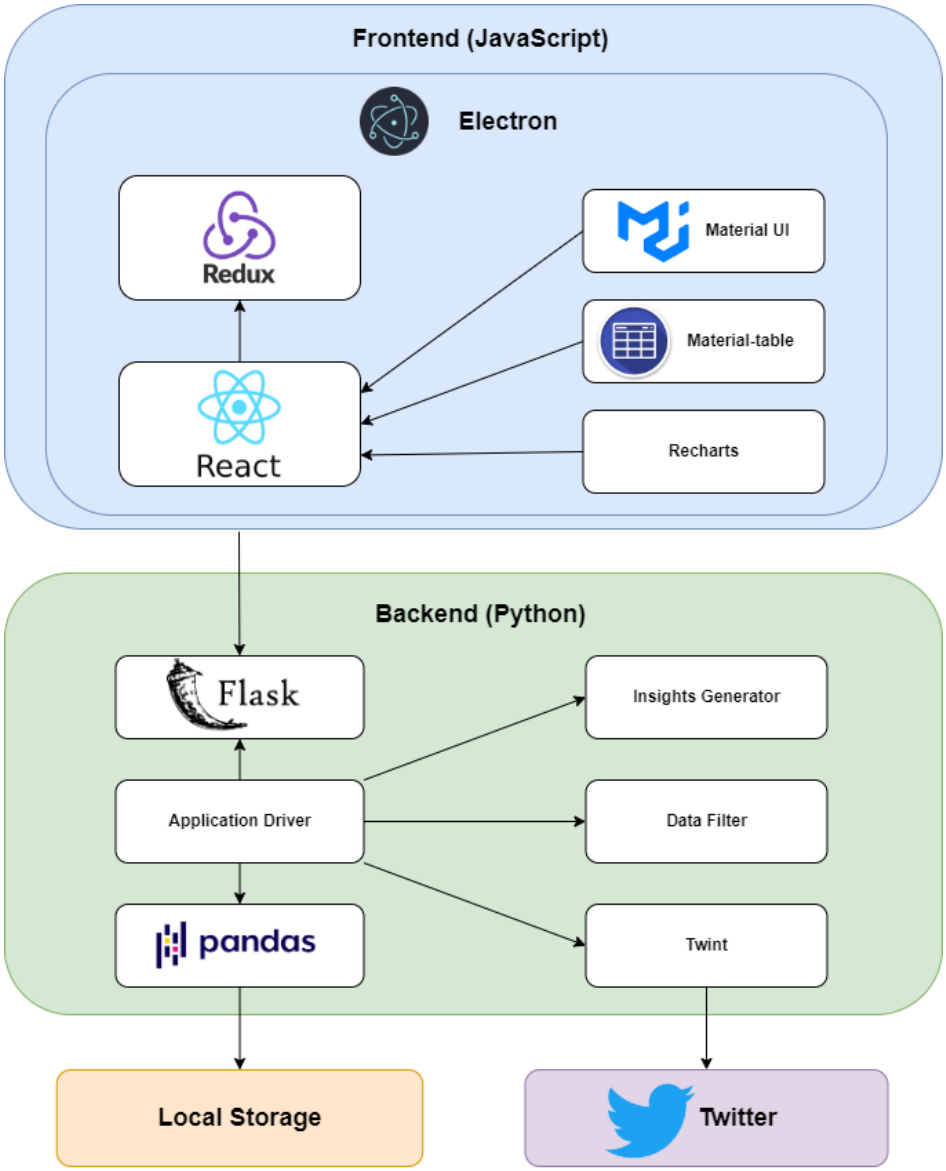

Figure 1 Static elements of the application

Frontend (NodeJS)

Electron

Electron is an open-source library for building cross-platform applications with HTML, CSS, and JavaScript. Using this technology, we were able to create the GUI of the application with our preferred frontend framework and web UI toolkit. Electron provides a flexible abstraction for native operating system features. Therefore, we can maintain a single codebase and still have an application that will run on most popular platforms like Mac, Windows, and Linux.
https://www.electronjs.org/

React

The client stressed on the importance of an easy user interface. To provide that, we found that using React will allow us to reach that goal. With limited development time, we wanted to use this technology because of its ease of use and quick ability to learn. ReactJS is highly intuitive to work with. It offers interactivity to the layout of any user interface. Not only this, but it also allows fast, high-quality, and scalable application development that saves time.
https://reactjs.org/

Redux

Redux is an open-source library used to manage the application state. Using this along with React allows us to increase the performance of our application. React Redux implements many performance optimizations internally, so that the component only re-renders when it needs to instead of when there are any updates to the component.
https://redux.js.org/

Material UI

Material UI is a library allowing us to import and use different components to create a user interface in our React application. This saves our developers a significant amount of time as we don't have to write everything from scratch. Some components we have used are buttons, drop lists, and text fields.
https://mui.com/

Material-table

Material-table is a React table component that is based on Material UI. This component allows us to display the data scraped in an organized table format. It also allows us to perform additional filtering, searching, and sorting on the data in the table.
https://material-table.com/#/

Recharts

Recharts is a composable charting library built on React components. It allows us to quickly build charts and graphs with decoupled, reusable React components. These charts and graphs help display the insights collected in an organized and appealing manner.
https://recharts.org/en-US/

Backend (Python)

Flask

Flask is a Python web framework that provides useful tools and features that make creating web applications in Python easier as developers don't have to worry about low-level details. Because it is one of the most popular frameworks, it is up-to-date and modern. You can easily extend its functionality and scale it up for complex applications. This component of the application is used to connect to the frontend component which allows the information collected to be displayed.
https://flask.palletsprojects.com/en/2.0.x/

Application Driver

The application driver handles all the logic in the backend and distributes the tasks to various parts of the application. This creates a level of abstraction that allows us to easily maintain the code.

Pandas

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. Pandas is fast and efficient for manipulating and analyzing data and allows data from different file formats to be stored and uploaded. For example, it takes data from local storage (like a CSV file) and creates a Python object with rows and columns that can be easily manipulated.
https://pandas.pydata.org/

Insight's generator

This part of the application handles all our insight logic. When requested by the user, this will generate all the data necessary for the frontend to display the result of the insight. Having a component handles this task prevents us from overloading the application driver with code not related to its responsibility.

Data filter

This part of the application handles the data filter process and keeping track of the filters for a data file. This allows us to not overload our application driver with code not related to its responsibility.

Twint

`Twint` is an advanced Twitter scraping tool written in Python that allows for scraping tweets from Twitter profiles without using Twitter's API. It allows us to scrape tweets from specific users, relating to certain topics, hashtags, and trends. Some of the benefits of using `Twint` instead of the Twitter API is that it can fetch almost all tweets while Twitter API limits to 3200 tweets only. It is also fast and easy to set up and can be used anonymously and without a Twitter sign up.
https://github.com/twintproject/twint

Local storage

Because this application interacts with large amounts of data, any bottleneck in the data storing and loading process will affect the whole application performance. Local storage offers us the best performance for our use case. The only drawback of using local storage is that it will take up space in the user's hard drive. However, this tradeoff was accepted for the best performance of the application.

Dynamic Elements

The dynamic system architecture displays the control flow within the system and the interactions of the system as information is shared between the components of the application. Figure 2 displays a system sequence diagram which shows sequences for the specific use case of a user performing a search and generating insights from the tweets collected. The diagram is used to display the order of the use cases and the events occurring inside the system. The color of each component corresponds to its associated component shown in Figure 1.

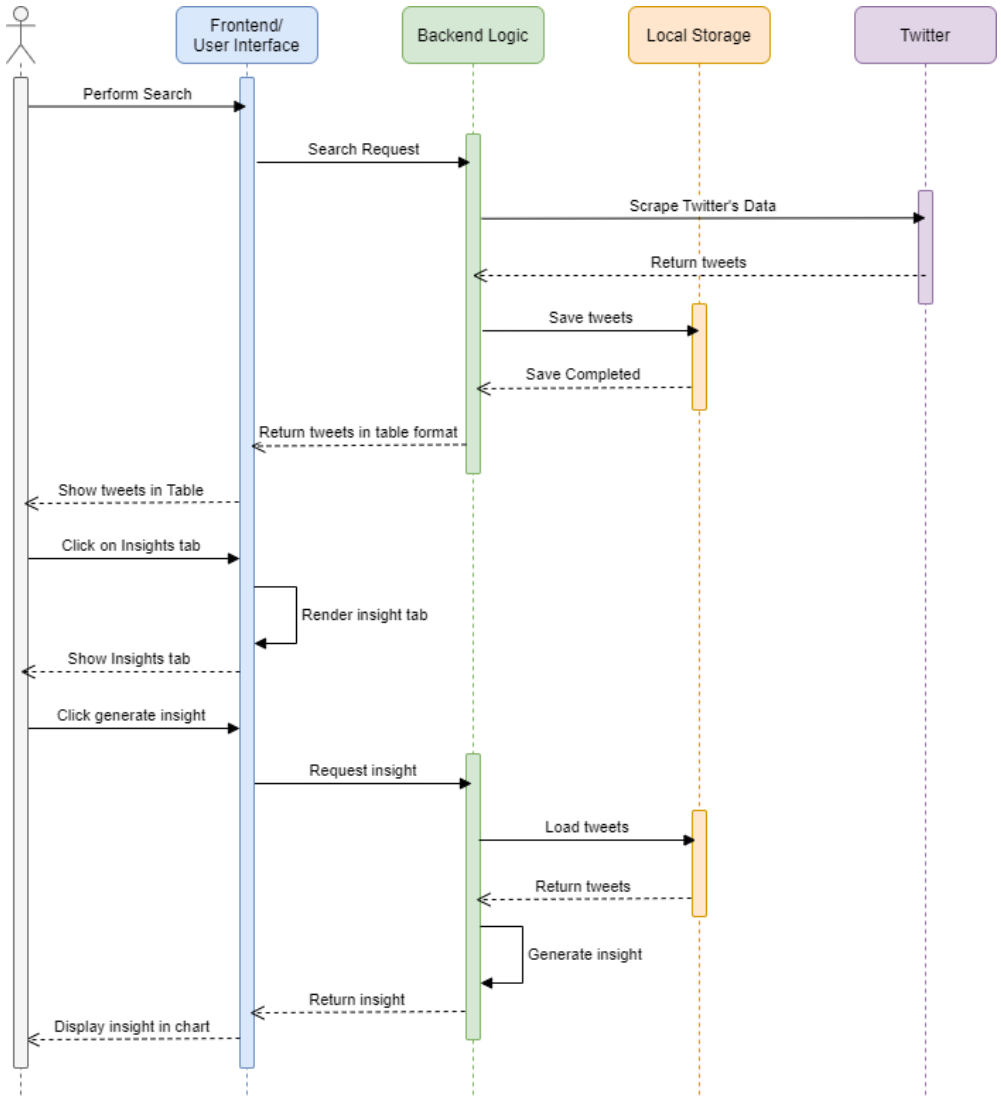

Figure 2 SSD of user performing a search and generating insight from tweets

Figure 2 represents how the components in the application interact when a user performs a search and collects insight about it. To perform the search, the user accesses the frontend of the application and enters certain information about the type of search they would like to do. Once they submit that, the request is sent to the backend of the application where it scrapes tweets from Twitter and returns them. Once the tweets are collected, they are saved and sent back to the frontend and displayed in a table format where the user can easily view them. Next the user accesses the insights page to view certain insights on the tweets scraped. On the frontend, the user chooses to generate the insights and the request is sent to the backend. From there, the tweets that insights need to be generated for are collected from the local storage and returned to the backend. Once that is complete, the insights are generated and sent back to the frontend of the application and shown in various types of graphs and diagrams for the user to view.

Data Storage Design

We will be using local storage for this project instead of a database. This section will contain four subsections: database use, file use, data exchange, and security. The database use subsection will provide information about the use of any specific databases within our application. The next subsection, file use, will contain information about the use and format of files within the application. The data exchange subsection will explain how the application transfers data and what protocols it uses. Lastly, the security section will discuss any security concerns the development team has and ways to address them.

Database Use

For this project, we chose not to use a traditional database due to the following reasons:

- Flexibility: A traditional database format is not flexible enough for file conversion.
- Convenience: With CSV, JSON, or XLSX file format, it is easy for users to condense and move around the files without the hassle of interacting with the database. Moreover, a CSV file is easier for business users to understand compared to relational databases.
- Complexity: although a SQL Database is more powerful for complex database related tasks, it is easier to use Pandas for this project without much effect on speed or performance.

However, as our project is heavily data centric, we will cover the usage of file formats, data source, data processing in the next sections.

File Use

Data Folder

This folder stores all the tweet's data when the user performs a search or saves filtered data. In this section, we will discuss how we get these data, how they are stored, and what we use the data for.

Data source

`Twint` is a library which allows users to scrap tweets from Twitter without using Twitter's API. For this project, we use `Twint` to develop a search function to scrap data based on our input conditions. There are many benefits of using `Twint` compared to Twitter API:

- Can fetch more Tweets compared to the limit of Twitter API (3200 tweets)
- Can be used anonymously without Twitter sign up.
- No rate limitations.

The search function will take the following input parameters from users:

- `Keyword`: The target keyword which must include in the tweets.
- `UserID`: this userid is used to collect all tweets from the target user.
- `Since`: search for data from this date
- `Until`: search for data up to this date
- `Days`: Search for tweets within a number of days from current date.

This function will collect the tweets data and save into a local file in system for ease of access.

Data Storage and Format

Data files are stored and saved to local storage. After a twitter search has been completed, the application user has the option to save the results onto their local computer in CSV, JSON, or XLSX format. This gives users the flexibility to save the data collected and view it at a later time on their preferred file format. To do this, we must export the Pandas DataFrame to the file chosen by the user. In Figure 3, you can see how the Pandas methods `to_csv()`, `to_excel()`, and `to_json()` to save the data. For example, `to_json()` will convert a DataFrame object to a JSON string or store it as an external file (similar process for the other methods).

```
def save(self, file_type):
    file_name = self.setting('Data Path') + get_file_name([self.file.split('.')[0]],
        file_type, self.files)
    if file_type == ".csv":
        self.data.to_csv(file_name)
    elif file_type == ".xlsx":
        self.data.to_excel(file_name)
    elif file_type == ".json":
        self.data.to_json(file_name)
    self.update_files()
    return {'files': self.files}
```

Figure 3 Code displaying functionality to save data in various file formats

Users can also view this saved data on the application. The application is able to read files formatted in CSV, JSON, or XLSX and display the data in a table format or perform insight analysis on the data in the file. To read files, we must convert the specific file type back to DataFrame format using Pandas (as displayed in Figure 4). Using Pandas functions `read_csv()`, `read_excel()`, and `read_json()` we are able to import the specific file as a DataFrame.

```
def read_file(file):
    _, file_type = file.split('.')
    if file_type == 'csv':
        return pd.read_csv(file)
    if file_type == 'json':
        return pd.read_json(file)
    if file_type == 'xlsx':
        return pd.read_excel(file, engine='openpyxl')
```

Figure 4 Code displaying functionality to read data from various file formats

Data Processing

Once the data is in a DataFrame, a column of data can be easily access using `DataFrame['column_name']`. For example, to perform sentiment analysis, the tweets can be accessed using `data['tweet']` which returns a list of tweets that can be iterated to get the sentiment for each tweet. Another use case for the DataFrame is to filter the data. This can be done by passing in an array of Booleans for each entry in the data. Having the data in a DataFrame not only makes it easy to load and store, but it also allows the data to be easily accessed to generate insights and filtered to remove unwanted entries.

Insights Folder

For every file in the data folder, we also have a file in the insights folder to keep track of the insights. This file is generated when the application discovers a new file presented in the data folder. These insights files are managed by the insight's generator class.

Insights Storage and Format

These files are saved and loaded using Python built-in function `open()` and the built-in library `pickle`. The loading and saving methods are written in `InsightGen.py` and are defined as `save_insights()` and `load_insights()`. The files are saved as a python object converted into a byte stream in a `.pkl` file. The structure of the object is a Python array of dictionaries.

Here is an example of an insights file object with only the sentiment analysis insight generated

```
[
  {
    'type': 'Sentiment Analysis',
    'list': [
      {
        'color': 'gray',
        'string': '-0.041 sentiment score'
      },
      {
        'color': 'green',
        'string': '6% positive tweets'
      },
      {
        'color': 'gray',
        'string': '86% neutral tweets'
      },
      {
        'color': 'red',
        'string': '8% negative tweets'
      }
    ],
'graph': [
      {
        'name': 'Negative',
        'Negative Count': 35
      },
      {
        'name': 'Neutral',
        'Neutral Count': 397
      },
      {
        'name': 'Positive',
        'Positive Count': 29
      }
    ],
    'layout': {
      'w': 6, 'h': 3, 'x': 0, 'y': 0,
      'i': '1:Sentiment Analysis',
      'moved': False,
      'static': False
    }
  }
]
```

Figure 5 Example Insight data object

Each insight in the array is a dictionary usually with the keys `type`, `list`, `graph` and `layout`. The "`type`" key store the name of the insight. The "`list`" key is a list that will be display on the frontend. The "`graph`" key store the information of the graph such as a bar chart. Finally, the "`layout`" key store the layout of the panel that will contain the insight.

Saving these will allow the application to be reopened without having to regenerate the insights and change the layout of the insight dashboard.

Data Exchange

The backend server is used to collect data, either from local storage or Twitter. The application uses `Twint` to collect data from Twitter, and then provides the data in CSV format. We use Pandas to read data from local storage and create a DataFrame, a Python object.

To display the data in the frontend, we need to convert it to the appropriate JSON format so that our table component can display the data. To do this, we used the `get_table()` function in `utils.py` to convert the DataFrame to the following JSON format.

```
[
    {'id': ..., 'conversation_id': ..., 'created_at': ..., 'date': ..., ...},
    {'id': ..., 'conversation_id': ..., 'created_at': ..., 'date': ..., ...},
    {'id': ..., 'conversation_id': ..., 'created_at': ..., 'date': ..., ...},
    {'id': ..., 'conversation_id': ..., 'created_at': ..., 'date': ..., ...},
]
```

Figure 6 Tweets format

Each element in the array is a tweet or a single row in the data. The tweet is represented as a dictionary with each key corresponding to a column or a feature of the tweet.

Flask is used to connect the backend to the frontend to display the data from local drive or Twitter. To do this, the react application uses the function `fetch()` to send the request over to the Flask API to retrieve the table data.

In this application, the data is not exchanged between different devices.

Security

After discussing any possible security/privacy concerns, the team has found there to be none of which could lead to significant issues as of now.

The data collected and displayed in our application is already readily available for the public to view on Twitter, thus no encryption is needed. If the application were to collect private information, such as the tweets from a private account, we would need to encrypt the data to prevent any unauthorized person to gain access from it; however, this is not the case for this application. Therefore, all data that is collected and that may be saved onto a user's local storage pose no security or privacy concerns.

Additionally, since the application is hosted on a local server (a server that is used on your own computer), the information collected does not leave the local machine. Therefore, there is no need for a secure data exchange channel.

Currently, the team has decided to not implement a login system into the application, as we found it to be unnecessary with the existing functions. However, in the future if a login feature is requested, the development team can use a common authentication system such as Google authentication to make it secure.

The application does collect the name and username of the tweets collected which can be viewed as personally identifiable information (PII) in some circumstances. Within our application, as mentioned above, these names and usernames are already available to the public on Twitter, thus there are no current needs to protect this data.

Component Design Detail

This section of the document will provide a more in-depth view of the component level of our system, compared to that in the System Architecture section. Just like the System Architecture section, this section will provide both a static view and dynamic view of our system.

Static

The design class diagram shown in Figure 5 describes the system by visualizing the different types of objects within the application (specifically the frontend and backend) and the kinds of static relationships that exist among them. The diagram is color coded to match the diagrams shown in the System Architecture section. You can see the two main sections highlighted: the frontend (blue) and backend (green).

The user interfaces with the blue portion of the diagram, which highlights key components of the application such as the search tab and insights tab. These frontend components depend on the backend green portion of the diagram, which is shown using the <<fetch>> arrows. The backend components include the data processing and collection portions of the application. For example, the application calls the data filtering component to add or remove certain data shown, which is represented by the <<uses>> arrow from the Application to DataFilter. It is also visible that there is an aggregate relationship between the Application and the Data and Insights. This implies that the Data and Insights classes can exist independently of the main Application class.

Within the diagram, you can also view all the inter-component relationships, specifically the operations and attributes of each of the classes. For example, `Twint` uses the method `twint_search()` to scrape tweets from Twitter, which is highlighted within that component.

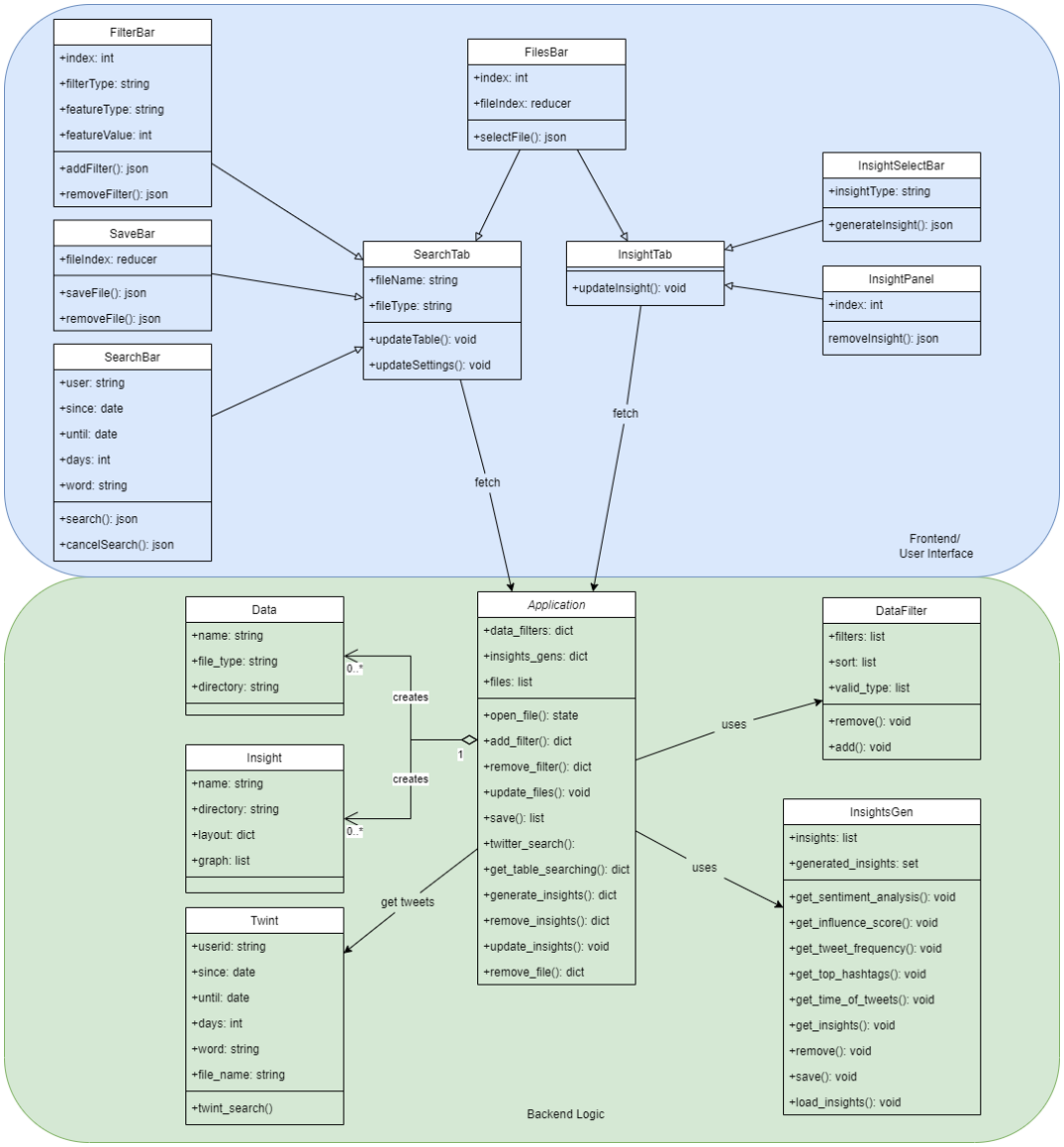

Figure 7 DCD showing static relationships of low-level components

Dynamic

The dynamic components of the application can be seen through a robustness diagram in Figure 6. This diagram highlights the runtime interactions between the user and the system while also displaying the dynamic runtime interaction between the components. The colors highlighted represent a separate component of the application and correspond to the diagrams shown in the System Architecture section.

The user initially has two options on the frontend of the application (highlighted in blue): search tweets or generate insights. If the user selects search tweets, a query is sent to the backend search controller, which sends a query to Twitter. Then the tweets are returned and displayed in a table format for the user to visualize. Just like this, the flow of other use cases such as filtering/sorting data, generating insights, and updating data table are represented in the diagram.

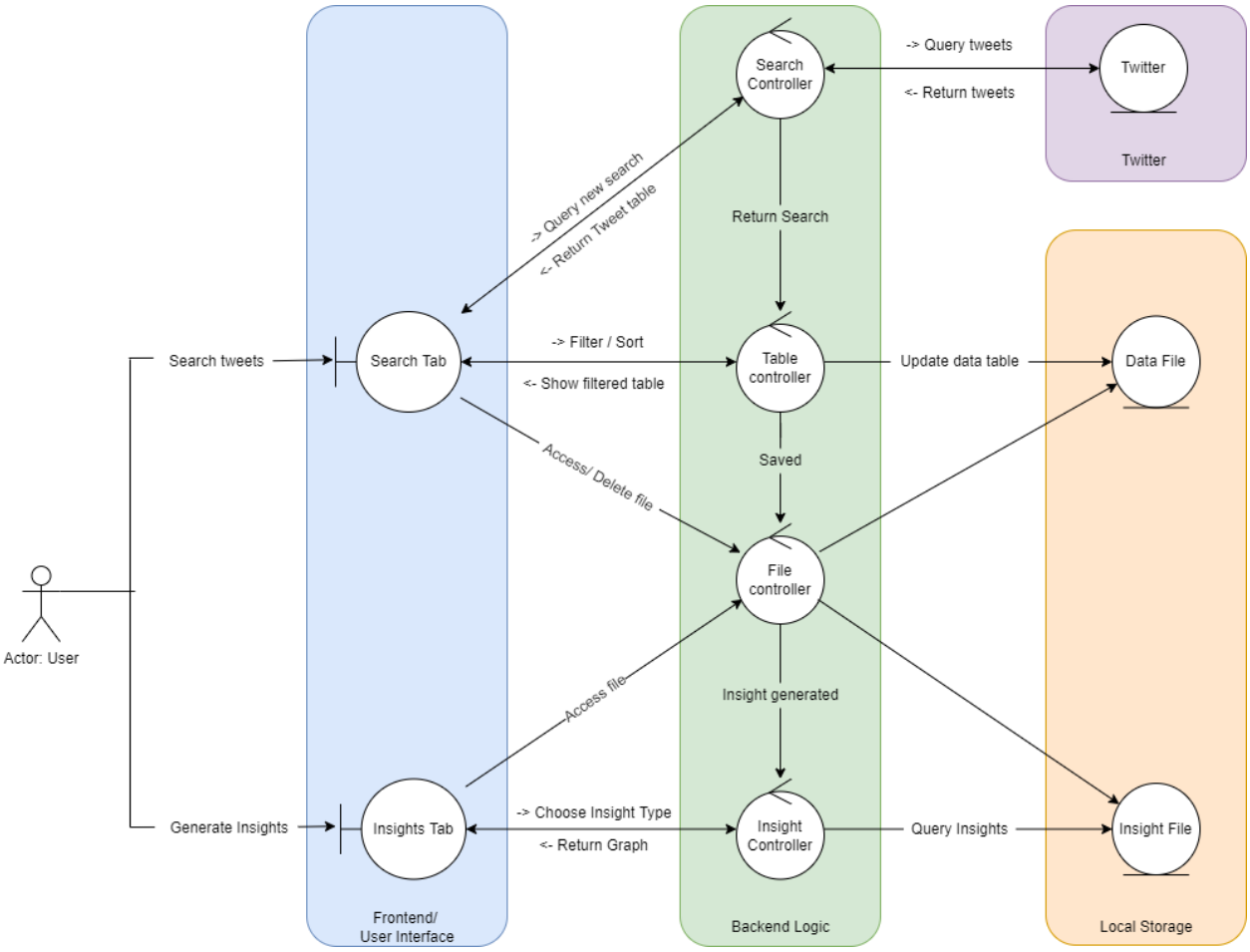

Figure 8 Robustness diagram showing dynamic runtime interaction between components

UI Design

In this section we present the User Interface (UI) of the application, which users will use when they would like to collect data or perform insight analysis on previously collected data. Our application has two main screens, the Search tab and the Insight tab. This section will provide detail on the screens and explanation for its design choices.

One of the biggest goals for this application was to ensure an easy UI for our users. To do this, we upheld major design principles such as clarity and consistency. For example, we ensured that the navigation from and to different screens are intuitive. Like most web applications, our application features a menu bar at the top of the page to allow users to easily switch to the different screens. Buttons throughout the app are highly visible as they use a contrasting color compared to the white background. All the elements throughout the application have a clear label or symbol to indicate its function. We upheld a strong visual hierarchy to ensure users can understand the level of importance for each element and easily complete their desired action.

Search Tab

When users first open the application, they will be directed to the search tab, shown in Figure 9. This screen has one purpose, searching for tweets by keyword or hashtag, allowing the elements on this tab to be intuitive for the user.

Firstly, the top of the tab holds the search bar which is used to carry out the search feature. Placing this at the top of the page shows users its importance in performing the search. Once the user selects the search button, they will shift their view to the chart below the search bar.

This chart contains all the collected tweets and any other information accompanied with the tweet, such as its date and time. Right above the chart is a search bar that users can use to filter out certain tweets. Next to the search bar is an icon indicating columns, which is used to filter out any undesired columns. When the user selects this icon, they can check or uncheck the columns they would like to view or not view, as shown in Figure 10.

Users can also sort the data alphanumerically by hitting the column header. An arrow pointing in the direction of the sort will help indicate to users if it is in increasing or decreasing order, as shown in Figure 11.

Once the user is done filtering and sorting, they can save the data collected using the dropdown and button below the chart. Once this dropdown is selected, shown in Figure 12, the user can select the file type and hit the save button. This file will then be shown in the file list on the left side of the screen. Users can navigate between different searches in this file list. The file that is currently being viewed will be indicated by a box surrounding the file. Users can also delete the file they have currently selected using the delete button below the chart.

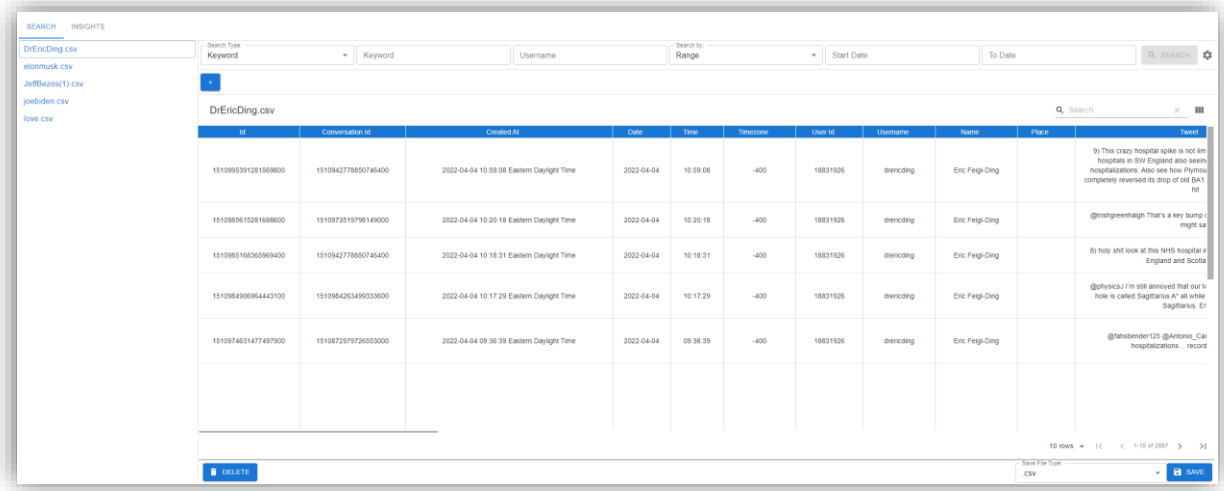

Figure 9 Search Tab after results from a search by keyword

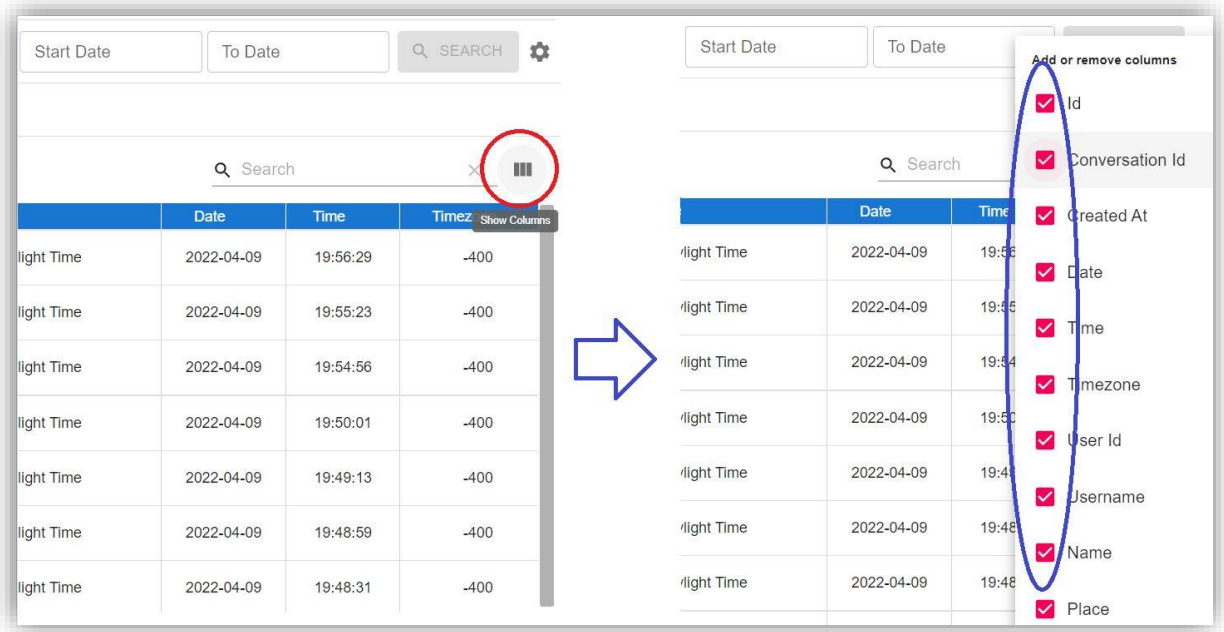

Figure 10 Highlights the column visibility feature

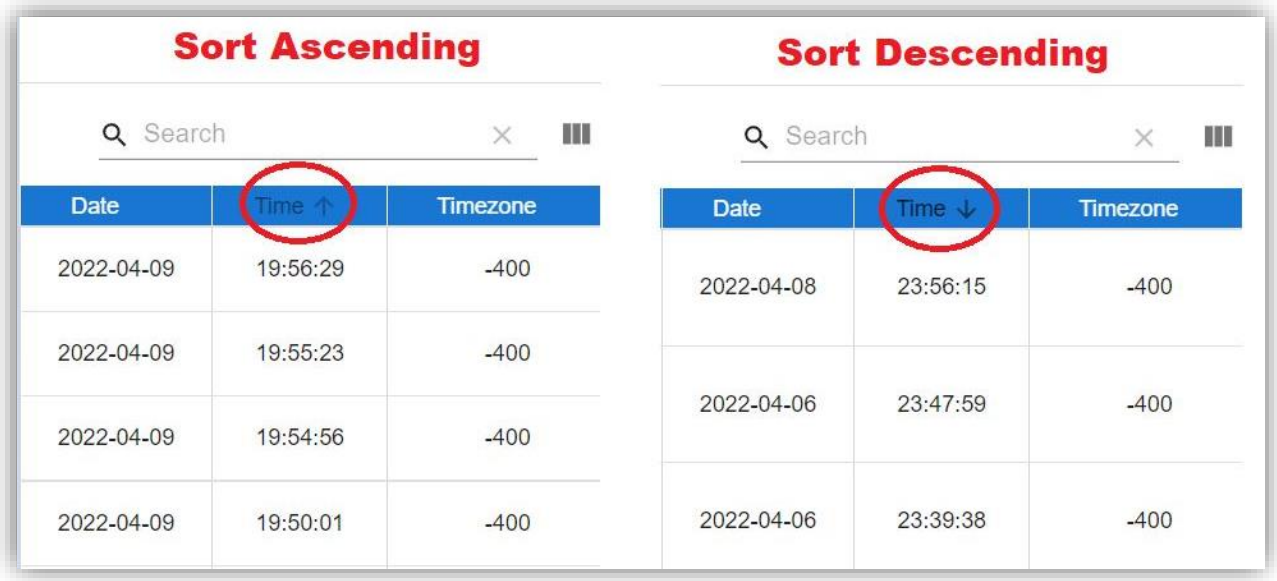

Figure 11 Highlights feature to sort columns in ascending and descending order

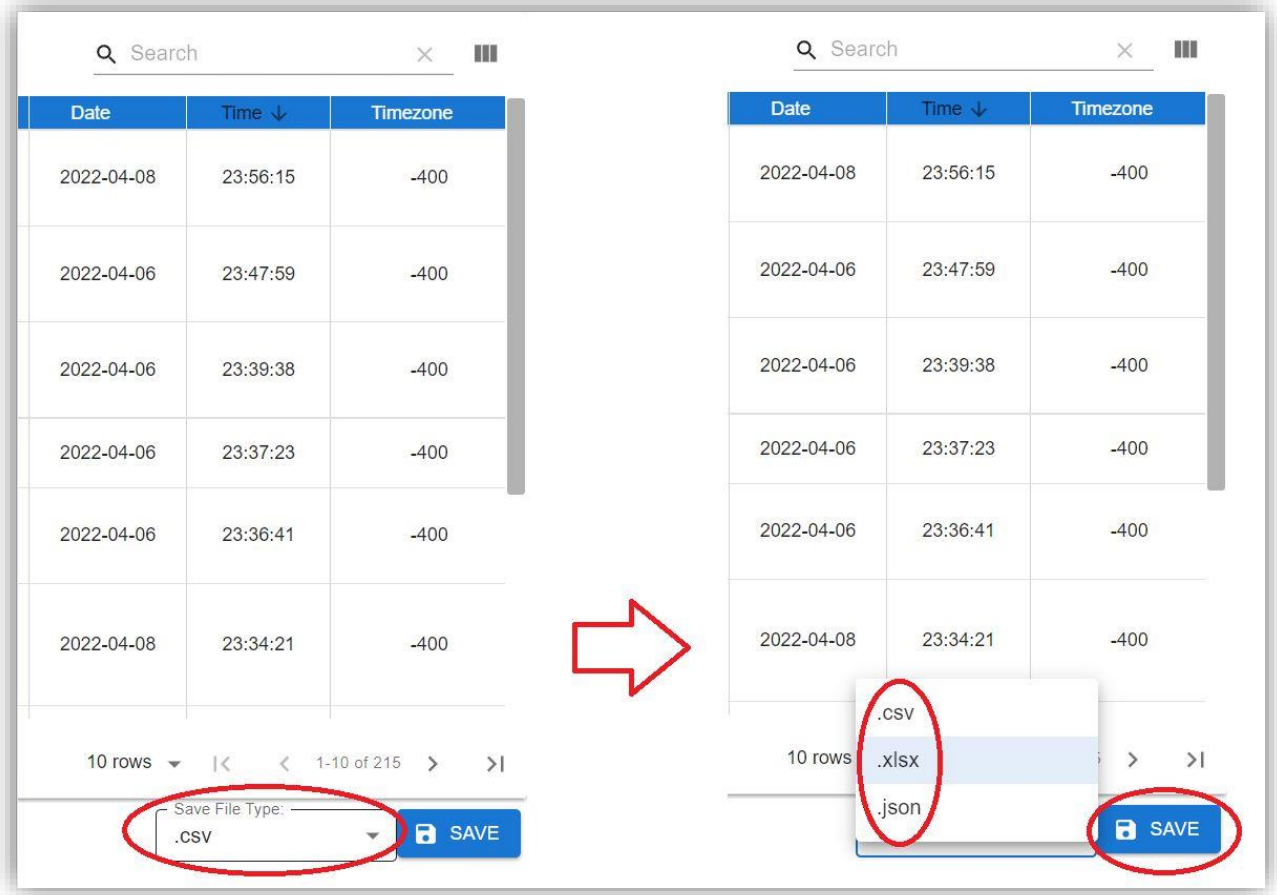

Figure 12 Highlights feature to save collected data in various file formats

Insight Tab

While using the application, users can switch to the Insight tab by clicking on the INSIGHTS button next to the SEARCH button on the top left corner of the application, as shown in Figure 13. As we can see, this tab is sharing the file folder with the Search tab, on the left column. The purpose of this screen is to generate insights, including charts and graphs for the corresponding data file.

Firstly, the user needs to choose a data file from the file folder. Then, an Insight Type will be selected from the dropdown menu which has five options: Sentiment Analysis, Influence Score, Top Hashtags, Tweets Frequency, and Time of Tweets. Finally, by clicking on the GENERATE button, the application will return the chosen insight type for the desired data file. As we can see from Figure 13, each insight type will have each own chart and statistic representation with distinct color design. Also, users can resize and relocate the chart within the screen for better visibility.

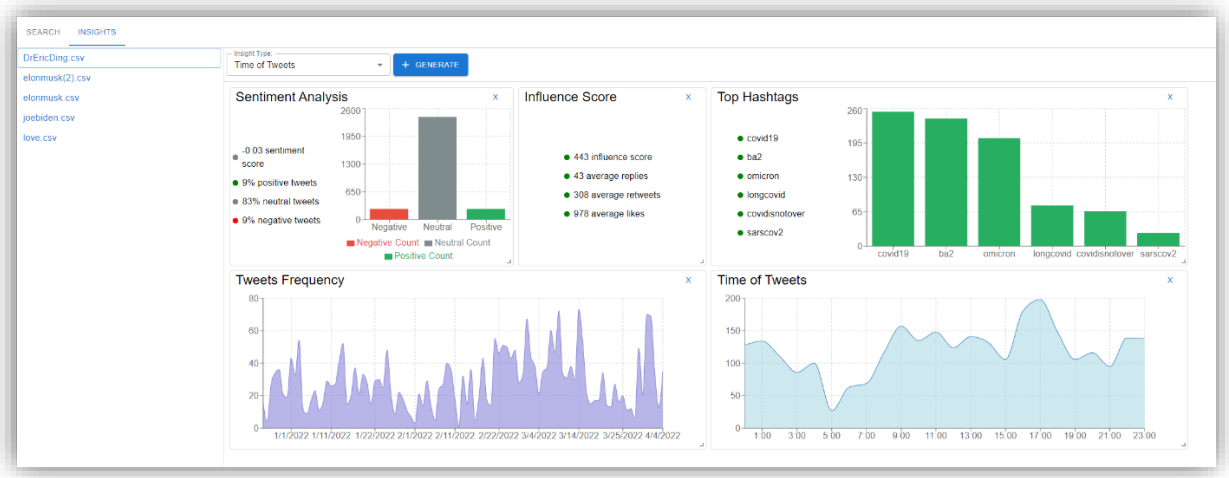

Figure 13 Insight Tab showing different Insight types

Appendix A RESTful API Documentation

Response Codes (Flask)

Status Codes	Description
1xx	Informational - This class of status code indicates a provisional response. There are no 1xx status codes used in REST framework by default.
2xx	Successful - This class of status code indicates that the client's request was successfully received, understood, and accepted.
3xx	Redirection - This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request.
4xx	Client Error - The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.
5xx	Server Error - Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Response code details: https://github.com/flask-api/flask-api/blob/develop/docs/api-guide/status-codes.md

GET search/start

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
`word`	String: keyword that the user wants to search
`user`	String: Username of the account
`days`	Number of days to search from current date
`since`	Search from this date, format: yyyy-mm-dd
`until`	Search to this date, format: yyyy-mm-dd
`type`	String that determines to search by keyword or hashtag. The default search type is keyword. To search by hashtag set `type` to '`hashtag`'.

Example Request

```
http://127.0.0.1:8000/search/start?user=elonmusk&word=&days=&since=&until=&type=
```

Example Response

```
{
  "data": {
    "files": [
      "elonmusk.csv"
    ],
    "filters": [],
    "insights": [],
    "settings": {
      "Data Path": "C:\\Users\\username\\Desktop\\ISpy\\backend/data/",
      "Insights Path": "C:\\Users\\username\\Desktop\\ISpy\\backend/insights/",
      "Searching number of new rows": 3,
      "Searching number of old rows": 5
    },
    "table": {
      "data": <Tweets object>
    }
  },
  "status": "success",
  "status_msg": "Search successfully"
}
```

GET search/cancel

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters
None

Example Request
`http://127.0.0.1:8000/search/cancel`

Example Response


```
{
  "data": {
    "files": [
      elonmusk.csv
    ],
    "filters": [],
    "insights": [],
    "settings": {
      "Data Path": "C:\\Users\\username\\Desktop\\ISpy\\backend/data/",
      "Insights Path": "C:\\Users\\username\\Desktop\\ISpy\\backend/insights/",
      "Searching number of new rows": 3,
      "Searching number of old rows": 5
    },
    "table": {}
  },
  "status": "success",
  "status_msg": "Canceled search successfully"
}
```

GET filter/add

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
`type`	Operator for the filter
`feature`	Feature name of the data
`value`	Number value

Example Request
`http://127.0.0.1:8000/filter/add?type=equal&feature=language&value=en`

Example Response
```
{
  "data": {
    "filters": [
      {
        "feature": "language",
        "type": "equal",
        "value": "en"
      }
    ],
    "table": {
      "data": <Tweets object>
    }
  },
  "status": "success",
  "status_msg": "Successfully added filter"
}
```

GET filter/remove

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
index	Number: remove filter at index from current file

Example Request
`http://127.0.0.1:8000/filter/remove?index=0`

Example Response

```
{
  "data": {
    "filters": [],
    "table": {
      "data": <Tweets object>
    }
  },
  "status": "success",
  "status_msg": "Successfully removed filter"
}
```

GET file/save

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
`fileType`	File type to save the data as. “`.csv`”, “`.xlsx`”, or “`.json`”

Example Request
`http://127.0.0.1:8000/file/save?fileType=.json`

Example Response

```
{
  "data": {
    "files": [
      "elonmusk.csv",
      "elonmusk.json"
    ]
  },
  "status": "success",
  "status_msg": "Successfully saved csv file"
}
```

GET file/remove

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
index	Number: remove file at index

Example Request
```
http://127.0.0.1:8000/file/remove?index=1
```

Example Response

```
{
  "data": {
    "files": [],
    "filters": [],
    "insights": [],
    "table": {}
  },
  "status": "success",
  "status_msg": "Successfully removed file"
}
```

GET file/select

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
`filename`	String that determines which file to select

Example Request
`http://127.0.0.1:8000/file/select?filename=love.csv`

Example Response

```
{
  "data": {
    "files": [
      "love.csv"
    ],
    "filters": [],
    "insights": [],
    "settings": {
      "Data Path": "C:\\Users\\username\\Desktop\\ISpy\\backend/data/",
      "Insights Path": "C:\\Users\\username\\Desktop\\ISpy\\backend/insights/",
      "Searching number of new rows": 3,
      "Searching number of old rows": 5
    },
    "table": {
      "data": <Tweets object>
    }
  },
  "status": "success",
  "status_msg": "Successfully selected file"
}
```

GET insight/generate

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
type	String that determines the type of insight to generate

Example Request
`http://127.0.0.1:8000/insight/generate?type=Sentiment Analysis`

Example Response


```
{
  "data": {
    "insights": [
      {
        "graph": [
          {
            "Negative Count": 58,
            "name": "Negative"
          },
          {
            "Neutral Count": 865,
            "name": "Neutral"
          },
          {
            "Positive Count": 577,
            "name": "Positive"
          }
        ],
       "list": [
          {
            "color": "red",
            "string": "4% negative tweets"
          }, ...
        ],
        "type": "Sentiment Analysis"
      }
    ]
  },
  "status": "success",
  "status_msg": "Successfully generated insight"
}
```

GET insight/remove

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters

Parameter	Description
index	Number: remove insight at index from current file.

Example Request
`http://127.0.0.1:8000/insight/remove`

Example Response


```
{
  "data": {
    "insights": [
      {
       {
        "graph": [
          {
            "time": 1649304000.0,
            "value": 107
          }
        ],
        "layout": {
          "h": 3,
          "i": "0:Tweets Frequency",
          "moved": false,
          "static": false,
          "w": 6,
          "x": 0,
          "y": 3
        },
        "type": "Tweets Frequency"
      }
    ]
  },
  "status": "success",
  "status_msg": "Successfully removed insight"
```

POST insight/update-layout

Request information
Request format: JSON

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters
None

Example Request
`http://127.0.0.1:8000/insight/update-layout`

JSON:
```
[
  {
    'w': 6,
    'h': 3,
    'x': 0,
    'y': 0,
    'i': '0:Sentiment Analysis',
    'moved': False,
    'static': False
  }
]
```

Example Response

```
{
  'status': 'success',
  'status_msg': "Successfully update layout"
}
```

GET settings/reset

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters
None

Example Request
`http://127.0.0.1:8000/settings/reset`

Example Response


```
{
  "data": {
    "files": [
      "love.csv"
    ],
    "filters": [],
    "insights": [],
    "settings": {
      "Data Path": "E:\\Tech\\JD2\\ISpy\\backend/data/",
      "Insights Path": "E:\\Tech\\JD2\\ISpy\\backend/insights/",
      "Searching number of new rows": 5,
      "Searching number of old rows": 5
    },
    "table": {
        "data": <Tweet Object>
        }
      ]
    }
  },
  "status": "success",
  "status_msg": "Successfully reset settings"
}
```

POST settings/update

Request information
Request format: JSON

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters
None

Example Request
`http://127.0.0.1:8000/settings/update`
JSON:

```
{
  'Data Path': 'C:\\Users\\username\\Desktop\\ISpy\\backend/data/',
  'Insights Path': 'C:\\Users\\ username \\Desktop\\ISpy\\backend/insights/',
  'Searching number of new rows': '0',
  'Searching number of old rows': 5
}
```

Example Response

```
{
  'data': {
    'files': ['love.csv'],
    'filters': [],
    'table': {
      'data': <Tweets object>
    },
    'insights': [],
    'settings': {
      'Data Path': 'C:\\Users\\username\\Desk top\\ISpy\\backend/data/',
      'Insights Path': 'C:\\Users\\username\\Desktop\\ISpy\\backend/insights/',
      'Searching number of new rows': 0,
      'Searching number of old rows': 5
    },
    'status': 'success',
    'status_msg': 'Successfully update settings'
  }
```

GET update-table

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters
None

Example Request
`http://127.0.0.1:8000/update-table`

Example Response
```
{
  "data": {
    "files": [
      "love.csv"
    ],
    "selectedIndex": 0,
    "table": {
      "data": <Tweets object>
    }
  },
  "status": "success",
  "status_msg": "Successfully updated table"
}
```

GET state

Resource Information
Response Format: JSON
Requires Authentication: No
Rate Limited: No

Parameters
None

Example Request
`http://127.0.0.1:8000/state`

Example Response

```
{
  "data": {
    "files": [],
    "filters": [],
    "insights": [],
    "settings": {
      "Data Path": "E:\\Tech\\JD2\\ISpy\\backend/data/",
      "Insights Path": "E:\\Tech\\JD2\\ISpy\\backend/insights/",
      "Searching number of new rows": 5,
      "Searching number of old rows": 5
    },
    "table": {}
  },
  "status": "success",
  "status_msg": "Successfully retrieved state"
}
```

Example response objects:

Tweets object:


```
[
  {
    "cashtags": "[]",
    "conversation_id": 1511808969165185035,
    "created_at": "2022-04-06 17:42:58 Eastern Daylight Time",
    "date": "2022-04-06",
    "geo": null,
    "hashtags": "[]",
    "id": 1511821793933373448,
    "language": "en",
    "likes_count": 13570,
    "link": "https://twitter.com/elonmusk/status/1511821793933373448",
    "mentions": "[]",
    "name": "Elon Musk",
    "near": null,
    "photos": "[]",
    "place": null,
    "quote_url": null,
    "replies_count": 1007,
    "reply_to": "[{'screen_name': 'remedygames', 'name': 'Remedy Entertainment', 'id': '18429711'}]",
    "retweet": false,
    "retweet_date": null,
    "retweet_id": null,
    "retweets_count": 585,
    "source": null,
    "thumbnail": null,
    "time": "17:42:58",
    "timezone": -400,
    "trans_dest": null,
    "trans_src": null,
    "translate": null,
    "tweet": "@remedygames Great games",
    "urls": "[]",
    "user_id": 44196397,
    "user_rt": null,
    "user_rt_id": null,
    "username": "elonmusk",
    "video": 0
  }
]
```