Project Gemini: Web Crawling and Data Visualization

Created by :
Murtadha Marzouq
Miguel Morel
Willis Reid
Seth Adams
Ryan Hull

Table of Contents:

Project Overview Error! Bookmark not defined.

Goals: 2

Technical Specifications: 2

Architectural Overview 3

Subsystem Architecture 3

Behavioral Diagrams: 3

Persistent Data Storage 5

Project Milestones 9

Front End 10

Back End 14

Appendix: 15

Project Overview

Link: https://capstone-front-end-lime.vercel.app/

A major issue that college students face when it comes to writing research papers or doing research. They normally struggle and don't really know where to start. Their teacher might recommend them to go to the library or the professor might even suggest checking the Internet for more information. So, you might say to yourself how do we solve some of these problems that students are having here at UNC Charlotte? We have created a website called Project Gemini. Gemini will crawl and analyze the University of North Carolina website and other relevant sites and use data collected from the J. Murrey Atkinson library. Which will include repositories of academic publications to compile and show a body of information on UNC Charlotte researchers and their interests and skills. The goal for project Gemini is to use the automated method to provide a visualization similar to a faculty and connection site.

Goals:

1. Data extraction and parsing using web crawling and analysis to produce meaningful data files that can be used as a Dataset for UNCC staff and faculty.
2. Provide visualization to large data sets to aid users' and researchers' understanding of the body of knowledge and expertise present at UNCC.

Technical Specifications:

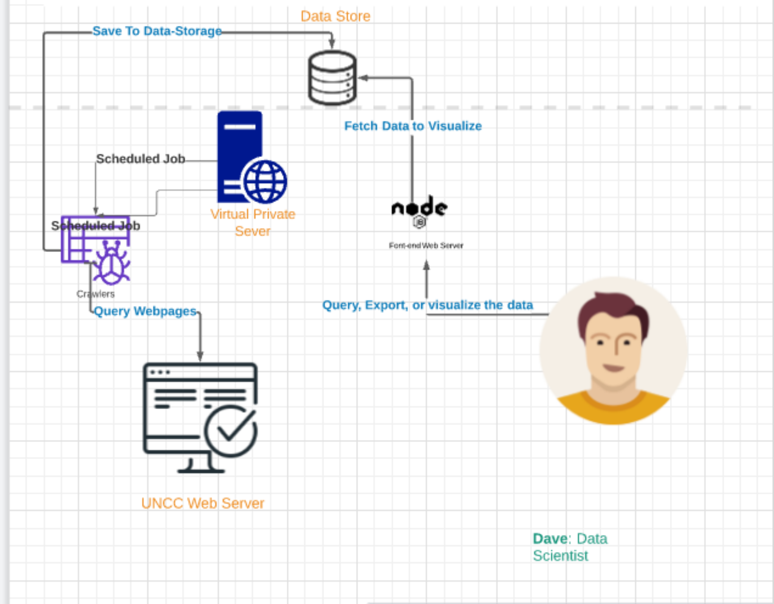


Figure C-1: Product Specifications

Gemini consists of two separate components:

Backend: Python, a cron job on the server to run every 24 hours to update the datasets.

Python Libraries: BeautifulSoup, pandas, requests, WebScraping

Frontend: Javascript (React, or Vanilla) for visualization and UX.

Javascript Libraries: Apache E-Charts, Next.js, Echarts-for-React (wrapper), React-Icons.

Plan of Action:

1. Find all the faculty and staff (https://pages.charlotte.edu/connections/)
2. Crawl each staff profile to scan for articles and research papers

3. Save the data to be analyzed using machine learning
4. Visualize the Data and save it as HTML to be used in a website.

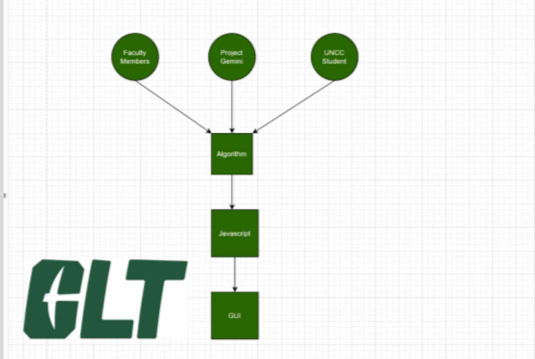
Security Discloser: Tor Browsing Feature:

The project supports connecting to the Onion Router's network to mask outgoing traffic and mitigate ip

Architectural Overview

When our group was thinking about how we wanted to design Gemini we wanted to make it easy for people to interact with. The team wanted it to where a person didn't have to worry about creating a login or I have to keep up with another password. We simply wanted a website that the user could use without wasting time and get the information they need without struggling. The group will be doing some data mining that will list all of the faculty and staff information on what they have written or published. Some of the tools that we will be using are HTML, JavaScript, and some Python to help display our data. We will also create a CSV file that contains relevant information about all staff members who work in the Akins library. Gemini will crawl the UNC Charlotte website and store the output response in a text file

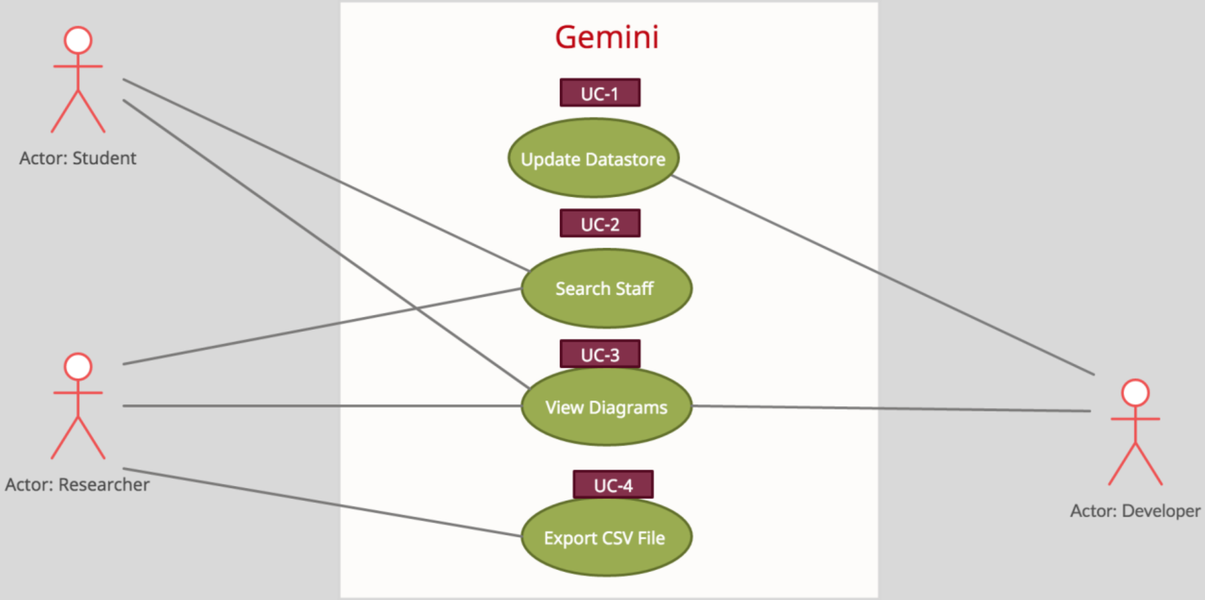
Subsystem Architecture



Behavioral Diagrams:

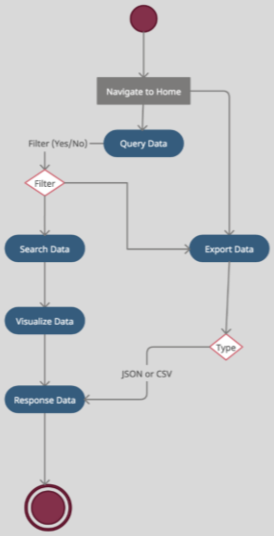
A Brief Use-Case

Actors:
Developers that support the application and update the datastores.
Researchers that want to use the datastore in JSON or CSV formats
Students gathering information about staff and departments in UNCC
Use case: UC-1. Search Staff
Actors: Student (primary), Developer
Description: The Student queries the datastore with a name of a staff member. By navigating to the search page, the user will be able to filter the data output. No login is required because this will be a public api



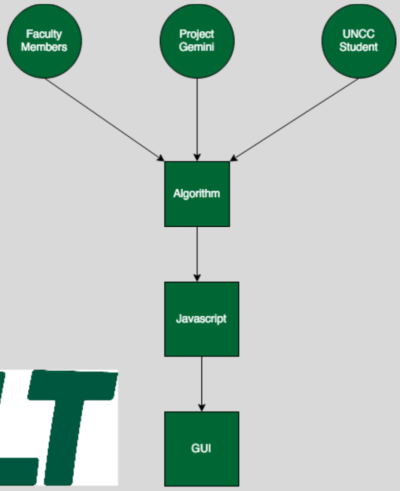
Link: Here

Activity Diagram:



Link: Here

Communication Diagram:



Project Gemini architecture has a frontend and a backend. The front end of the project will provide users to interact with our four different charts. The team decided to use a Sunburst, a tree graph, a treemap, and a pie chart. Each one of our graph users will have the ability to hover over the chart and it will show the information they also can click on the chart and it will go deeper in debt to explain what is going on. The back end of our website will be an Excel spreadsheet that will hold all of our faculty and staff members' information and their publication. We also will be using Python on the back end of our website to update the database every 24 hours.

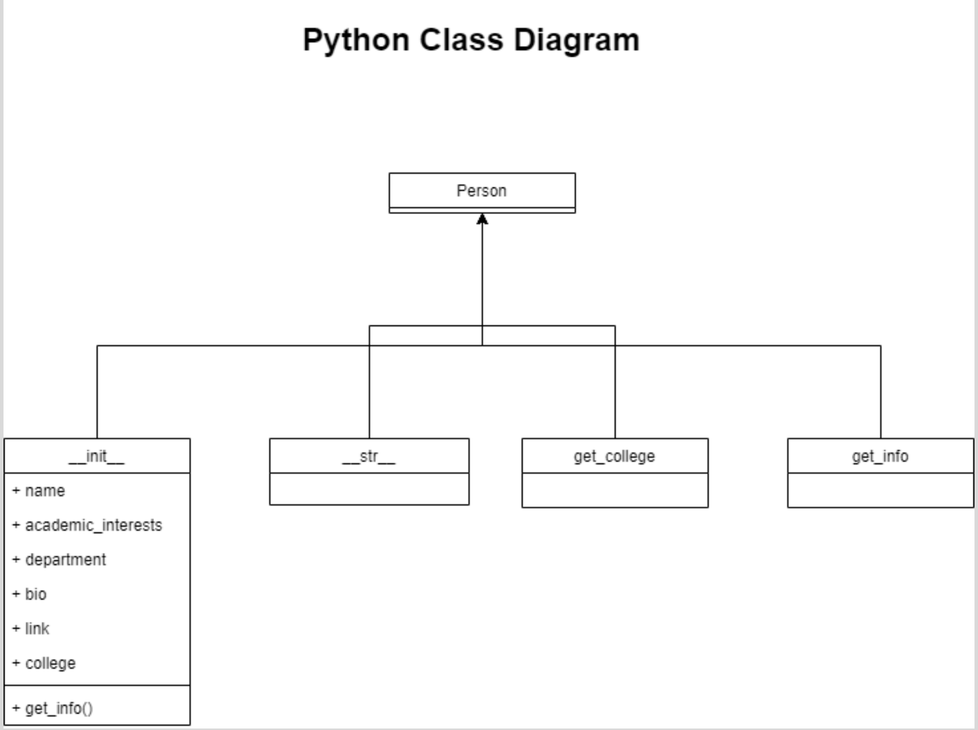
Persistent Data Storage

When it comes to storing and retrieving all of our information, basically all we do is crawl the UNC Charlotte database and get all the data that every faculty member has written and published. After getting the store data to a CSV, we wrote a JavaScript to convert it to JSON. In the JSON files, you will notice there's a list of arrays, which shows the information on the faculty and staff member like what college they're from, what information they write about, and their name. A link to their page and their bio info. After collecting all the information, we will run it into python to make it more user-friendly for a user so they can visualize data without having to read CSV files.

Global Flow Control

When it comes to the Gemini project, we want to make it basically easier for users to travel or visit websites without getting bogged down or overwhelmed. For example, a person can click on the data section of the website and search for faculty members and get all the information they need about that person. Another way for user interview information is to go to her charts and be able to click on four different charts to help them see which one they might prefer. Another way for our users to get information and data is by simply clicking on our charts to where they'll see four different charts, and the user can hover over a department and get information or click on the department and get information. We also made it to where a user can view or data or even download or data and use it for something else.

Static View (Semantics + Quality + Syntax)



There are many components in our project. Starting with the backend, our language of choice was Python. We chose this language because we were proficient and confident in our ability to successfully develop a crawler in order to gather data so that we could transform this data into graphs.

First off, our packages for the project's Python section. We decided to import the JSON package. The json.tool module provides a simple command line interface to validate and pretty-print JSON objects. The second package we imported was Beautiful Soup. Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping. Lastly, we imported the pandas package, which is useful for data manipulation and analysis.

Next, our Python code implemented a Person class which had four methods. The first method is a constructor: __init__(); it initializes the classes with the required parameters. The second method is __str__(), which converts the class object into a string. Third method is a getter method called get_college(), which returns the college name for a given department. The last method in the Person class is another getter method named get_info(). The job of this method is to collect information about the UNCC staff and save it to the variables declared in the __init__() constructor.

Our project implemented a single class in the backend side of things, which was just stated to be Person. The rest of the code was developed through independent methods and an array to set up the different variables used throughout the program. The two arrays are: urls[] and people[].

Now, the first method defined outside of the Person class was a getter called get_tor_session(). This method creates a tor session and returns it. Tor, short for The Onion Router, is free and open-source software for enabling anonymous communication, which we used to set up our scraper.

The next method is setup_initial_links(). This function is used to set up the initial links for the people in the website and it returns a dataframe of the links. This method also helps us fetch the html file for the pages.charlotte.edu/connections page, as well as save the files when we're done retrieving them.

Another method is get_college(). This function takes in a department and returns the college to which it belongs.

A method named visited_update() removes the url from the list of visited urls. It also returns an updated list of the visited urls.

There is also a method called get_json(). We used this method to get the JSON for a person after iterating through the dictionary.

We used get_all_links() to retrieve links. This function takes in a list of links and a limit on the number of links to be visited. It then calls the functions to visit the links and collect the data.

Similarly, we used read_UNCC() in order to read the retrieved staff data to a JSON file.

The method log_data() logs the obtained data in a JSON file.

Also, save_uncc() takes the same data and saves it in JSON format as well.

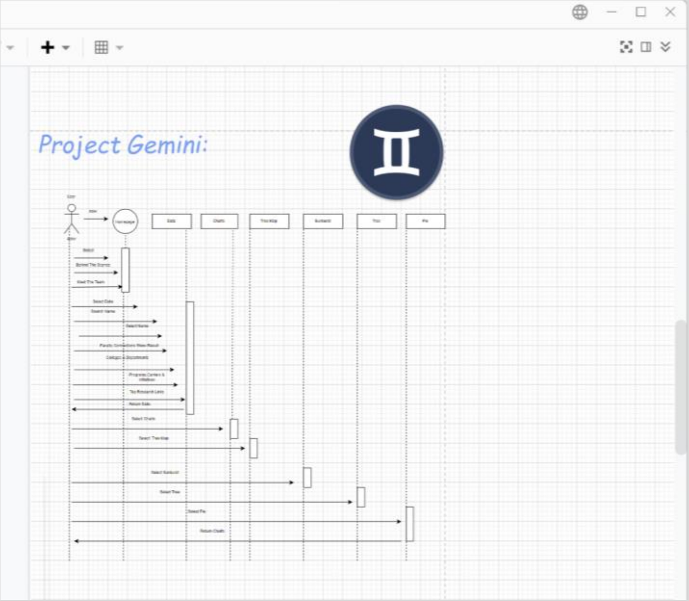
We used the add_to_UNCC() method to iterate through the UNCC saved dictionary and update the information contained therein.

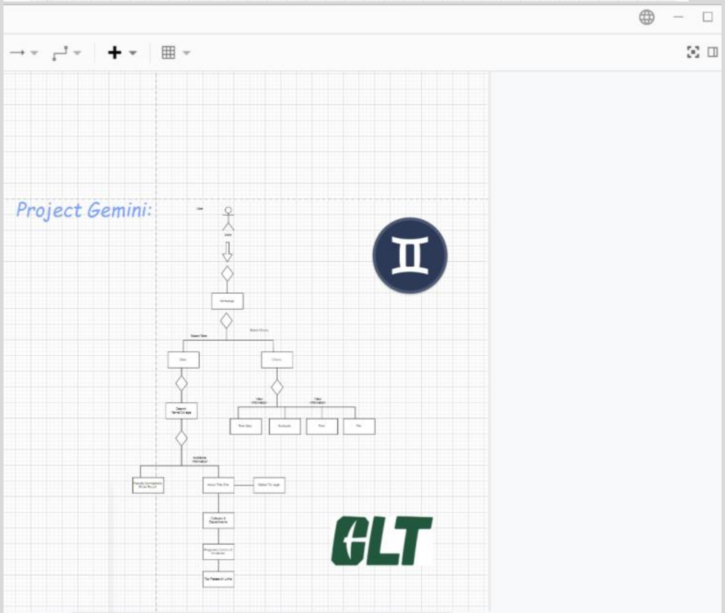
Lastly, and most importantly, we implemented a start() method that retrieves the staff information, updates the file, saves the file and then starts the application.

Evidently from the description of the functionality of our application in the paragraphs above, we spent an unbelievable amount of time in order to carefully plan, develop and test each method and each variable in the application. The quality of our solution is of utmost value and quality. Each module and method was designed to be intentionally loosely coupled, while maintaining high cohesiveness between our classes and modules.

Towards the end of the document, we have attached a “Front End” and “Back End” section as well as an “Appendix” section which includes minute details and screenshots showing every single part of our Python code. Please refer to it for more information.

Dynamic View (Semantics, Syntax)





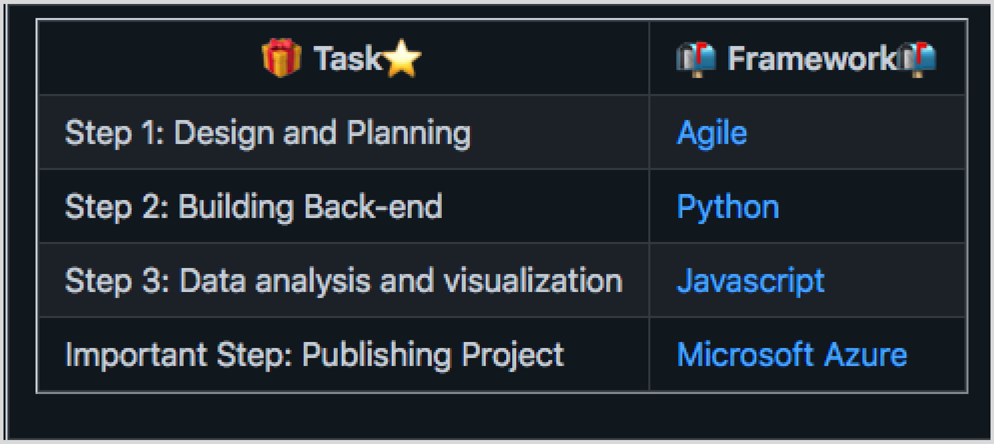
Project Gemini is all about being friendly with dynamic views, and the content shown above will show you how easy it is for our users to interact with our website. Once the user has reached our homepage, they’ll have two options: view data or view charts. If the user selects to view data, it will bring them to a new landing page where the person can search for a faculty and staff member by name or College of publication. If the user decides to click on the faculty and staff connection it will show you information about the publication, their office hours or phone number or their email, and a lot more information that will be helpful to the user. The website would also show other faculties that will have the same academic interest, for example you can click on the link and it will show you who else is interested in the same thing.

If the user selects a chart, they have four different charts, they can choose from Treemap, Sunburst, Tree, Pie chart. The user can hover over each department and get the information that they need with ease. If I'm a user, for example, I can click on the College of liberal arts and science and it will show me the different UNC Charlotte faculty and interests. This applies to all our charts, mostly we want users to control which chart is easier for them to navigate and work with the data. Another feature that our website has is that users can download a picture of the graph of the information they need and they also can view the data with the data view file, which will bring the user to a second page to read information easier.

Design Rationale

The design rationale behind our chosen design stack takes into account every factor relevant to the decision, namely each of the developers' skill and area of expertise when it comes to the planning, design, and development of the code that forms our project as a whole. Given the aforementioned variables, we decided it best to employ Agile for our project planning. For the backend, we decided to develop a crawler that would scrape the uncc.edu site every 24 hours in order to obtain data. After the crawler was ready, we then handed the data in JSON format to our Javascript developer, who employed Apache E-Charts in order to generate beautiful, informative graphs that illustrate the user-defined information after filters have been applied to it. Lastly, we decided to employ Microsoft Azure in order to publish our project. Although other design ideas were considered, this was the path that we decided on after much deliberation and consideration to all the factors mentioned above.

Project Milestones



🎁 Task⭐	📫 Framework📫
Step 1: Design and Planning	Agile
Step 2: Building Back-end	Python
Step 3: Data analysis and visualization	Javascript
Important Step: Publishing Project	Microsoft Azure

The team's Github Repository: Here

1. *Design Milestone: UML and User Stories (Sprint 1 and Sprint 0)*

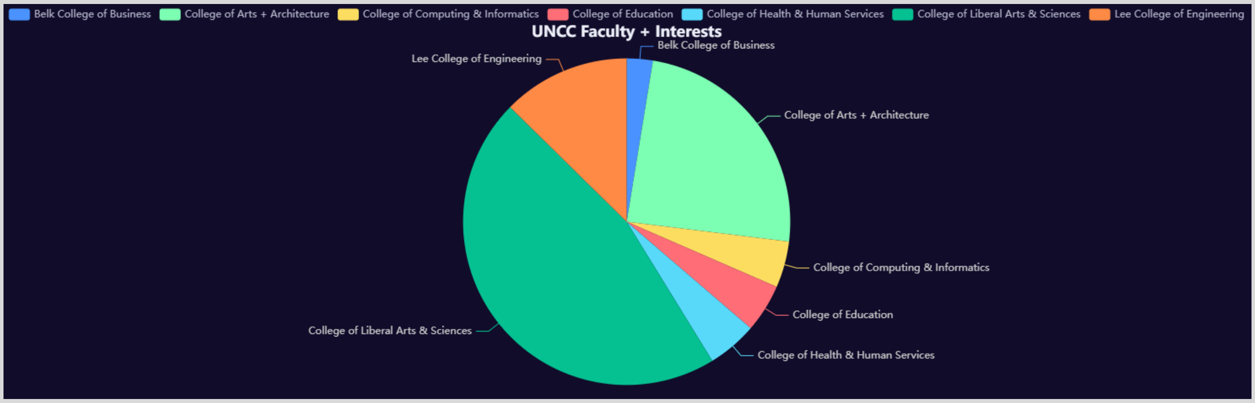
1. *Backend: Python implementation of data crawling (Sprint 2 and Sprint 3)*

2. *Front-End: Javascript to visualize the extracted data from the Connections website. (Sprint 4 and Sprint 5)*

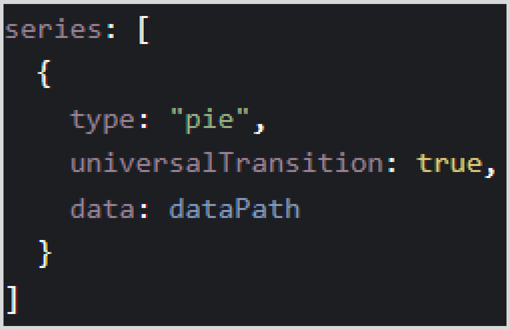
Front End

For our front end, we decided to use Javascript to implement the data we extracted from our backend in order to form beautiful, colorful, and illustrative displays.

Display 1: Pie Chart

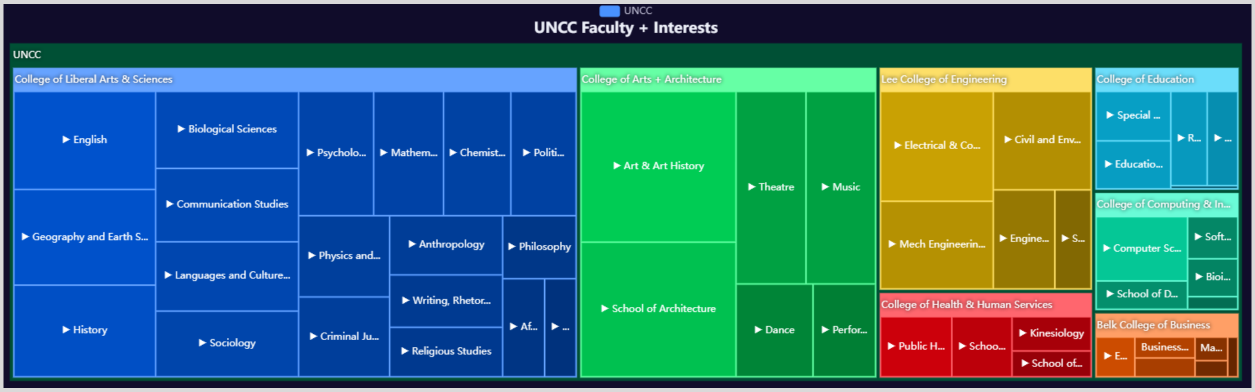


As for the styling of the Pie Chart, we used the following code:

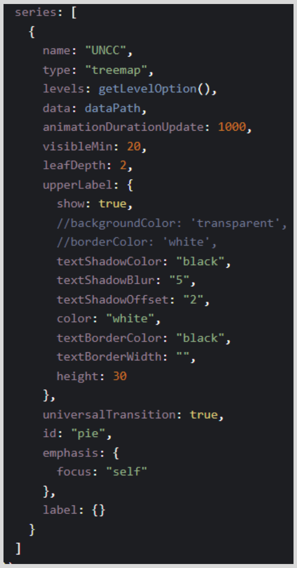


```
series: [
  {
    type: "pie",
    universalTransition: true,
    data: dataPath
  }
]
```

Display 2: Tree Map

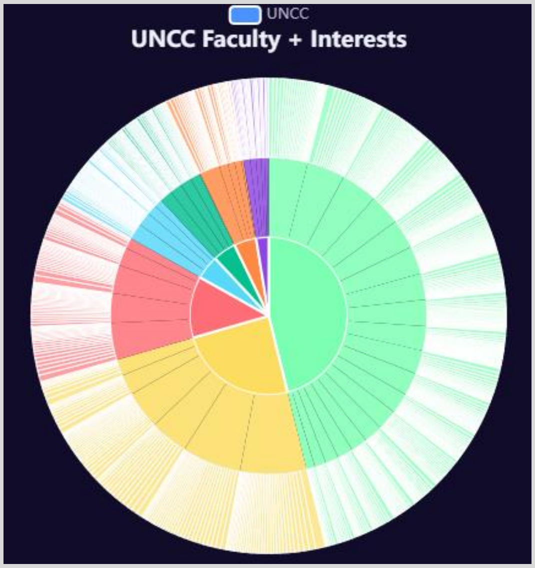


For the display of the Tree Map and to set the visual options we used the following:

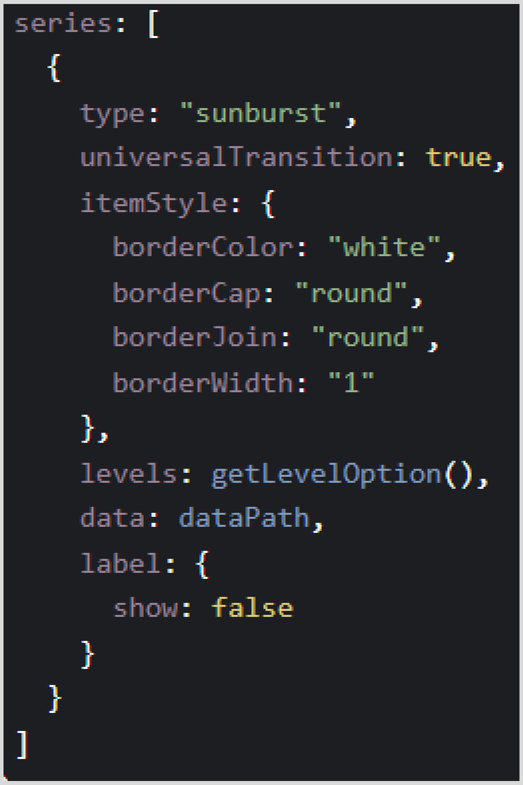


```
series: [
  {
    name: "UNCC",
    type: "treemap",
    levels: getLevelOption(),
    data: dataPath,
    animationDurationUpdate: 1000,
    visibleMin: 20,
    leafDepth: 2,
    upperLabel: {
      show: true,
      //backgroundColor: 'transparent',
      //borderColor: 'white',
      textShadowColor: "black",
      textShadowBlur: "5",
      textShadowOffset: "2",
      color: "white",
      textBorderColor: "black",
      textBorderWidth: "",
      height: 30
    },
    universalTransition: true,
    id: "pie",
    emphasis: {
      focus: "self"
    },
    label: {}
  }
]
```

Display 3: Sunburst

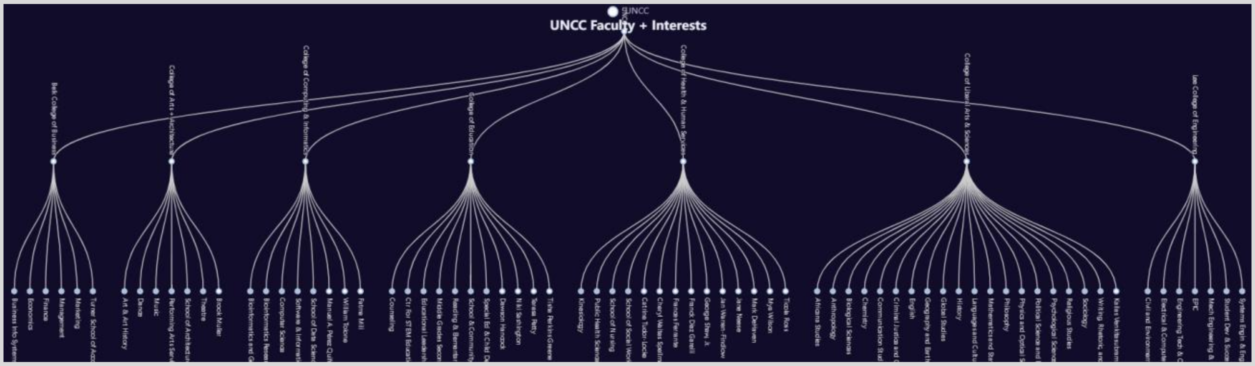


In order to set the properties of the Sunburst display, this is the code we employed:

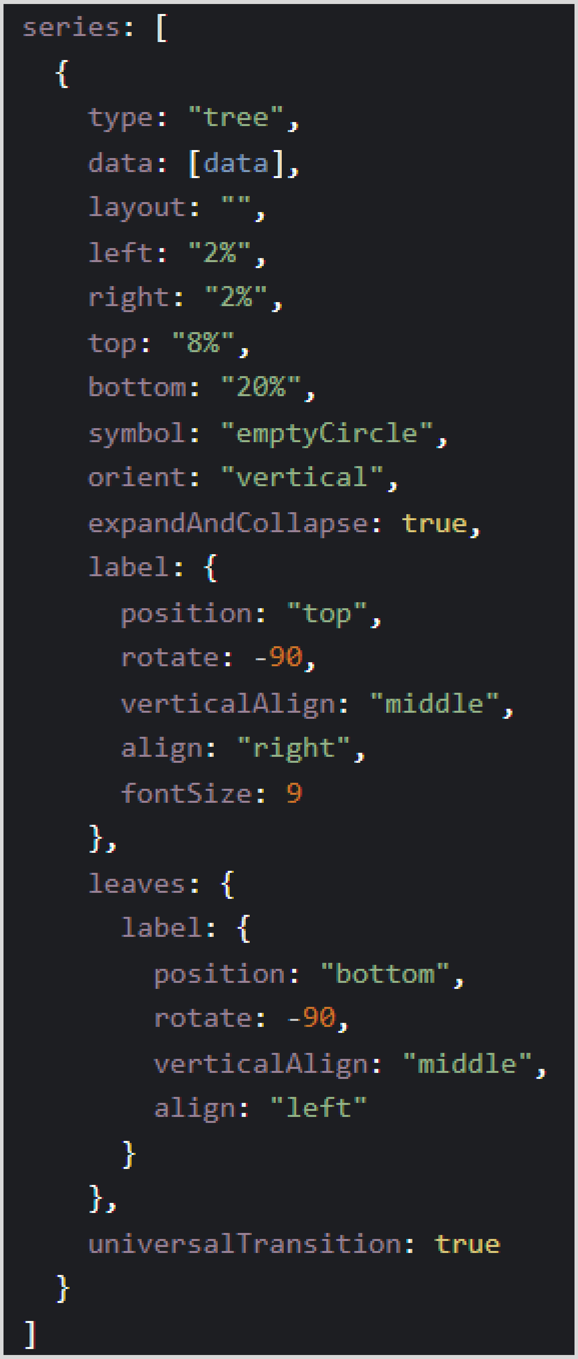


```
series: [
  {
    type: "sunburst",
    universalTransition: true,
    itemStyle: {
      borderColor: "white",
      borderCap: "round",
      borderJoin: "round",
      borderWidth: "1"
    },
    levels: getLevelOption(),
    data: dataPath,
    label: {
      show: false
    }
  }
]
```

Display 4:


UNCC
UNCC Faculty + Interests
Belk College of Business
College of Arts + Architecture
College of Computing & Informatics
College of Education
College of Health & Human Services
College of Liberal Arts & Sciences
Lee College of Engineering

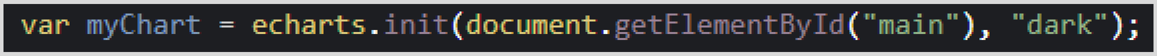
To display the tree the way we wanted to, we did the following:



```
series: [
  {
    type: "tree",
    data: [data],
    layout: "",
    left: "2%",
    right: "2%",
    top: "8%",
    bottom: "20%",
    symbol: "emptyCircle",
    orient: "vertical",
    expandAndCollapse: true,
    label: {
      position: "top",
      rotate: -90,
      verticalAlign: "middle",
      align: "right",
      fontSize: 9
    },
    leaves: {
      label: {
        position: "bottom",
        rotate: -90,
        verticalAlign: "middle",
        align: "left"
      }
    },
    universalTransition: true
  }
]
```

These charts took an incredibly long time to make, but we spared no time in working to deliver a final product we could be proud of. We used Javascript and React for the visualization and UX design. As for the Javascript libraries, we used Apache E-Charts, Next, Echarts-for-React, and React-Icons.

In order to initialize the E-Charts based on the prepared DOM, we used the following code:



```
var myChart = echarts.init(document.getElementById("main"), "dark");
```

Back End

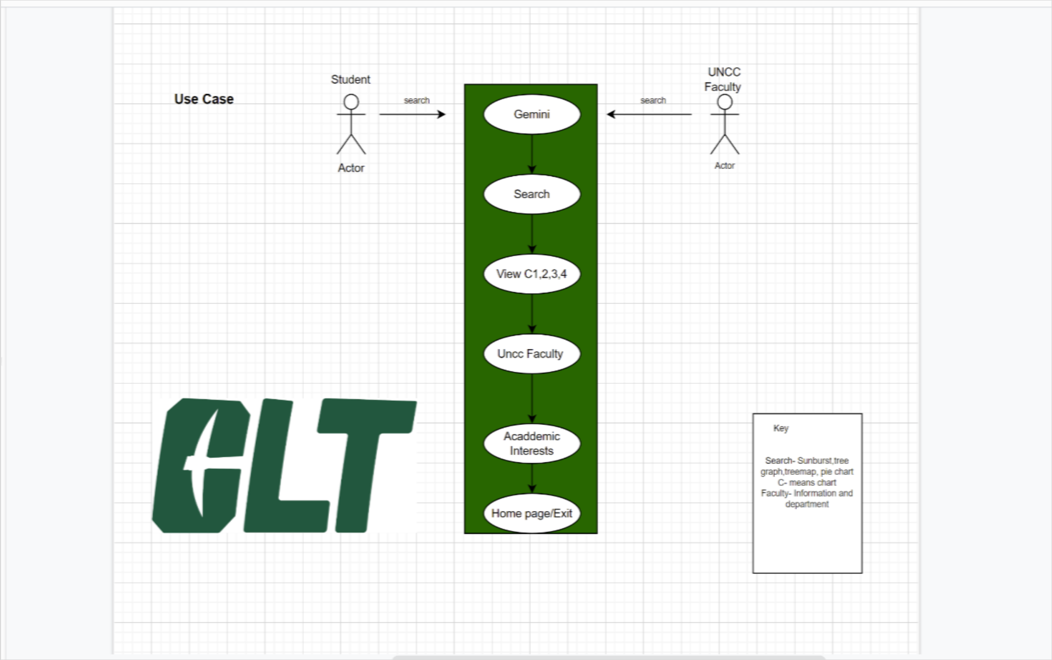
Note: Please refer to the Appendix section at the end of this document for the Figures mentioned in this section.

For our project's backend, we chose to use python in order to implement our desired functionality. The reason being that python makes it easy to work in conjugation with JavaScript, JSON, HTML and CSS.

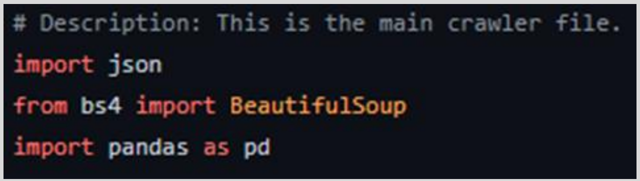
- The first step we took in developing the back end, was importing json and pandas. Pandas is a fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language. (See Figure B-1)
- Next, we set a set of 4 tasks that we needed to complete in order to successfully set up the back end. (See Figure B-2)
- This step consisted in declaring all of the variables that we'd be using later on and storing them all in a master array. (See Figure B-3)
- Next up, we decided to start up the tor service on port 9050, which we used to route our requests. (See Figure B-4)
- This is our class declaration containing several functions that define the functionality of our application. We use this to fetch information from the website we are scraping. (See Figure B-5)
- This section of the code aims to convert the class object into a string and to furthermore return the college name for a specific department. (See Figure B-6)
- This is how we fetch information from the website and store it in the class defined previously. (See Figure B-7)
- We use this portion of code to set up the initial links for the people in the website and store said links as files in the URL Storage section. (See Figure B-8)
- We implement this function in order to take a given department and return the name of the college it belongs to. (See Figure B-9)
- The goal of this function is to remove the URL from the list of already visited URLs. (See Figure B-10)
- This function iterates through the URLs and gets the pertinent information from the website. (See Figure B-11)
- The goal of these functions is to read, log, and save data to a JSON file. (See Figure B-12)
- This function is used to iterate through the UNCC saved dictionary and to update the existing information. (See Figure B-13)

- Finally, this function is used to get staff information, update their file, and save their updated information. Lastly, and most importantly, we use this function to start the application. (See Figure B-14)

This is the power of what we can accomplish by choosing Python for our project's backend. It was not an easy feat, by any means, but a thrilling journey, nonetheless.

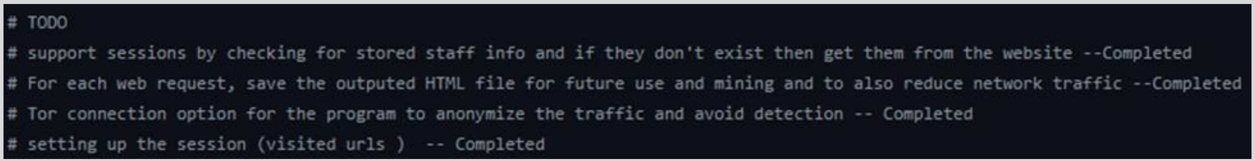


Appendix:



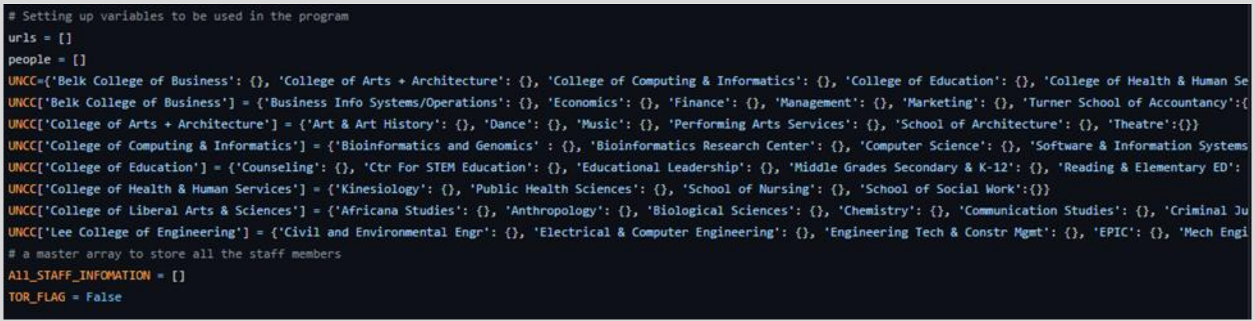
```
# Description: This is the main crawler file.
import json
from bs4 import BeautifulSoup
import pandas as pd
```

Figure B-1:



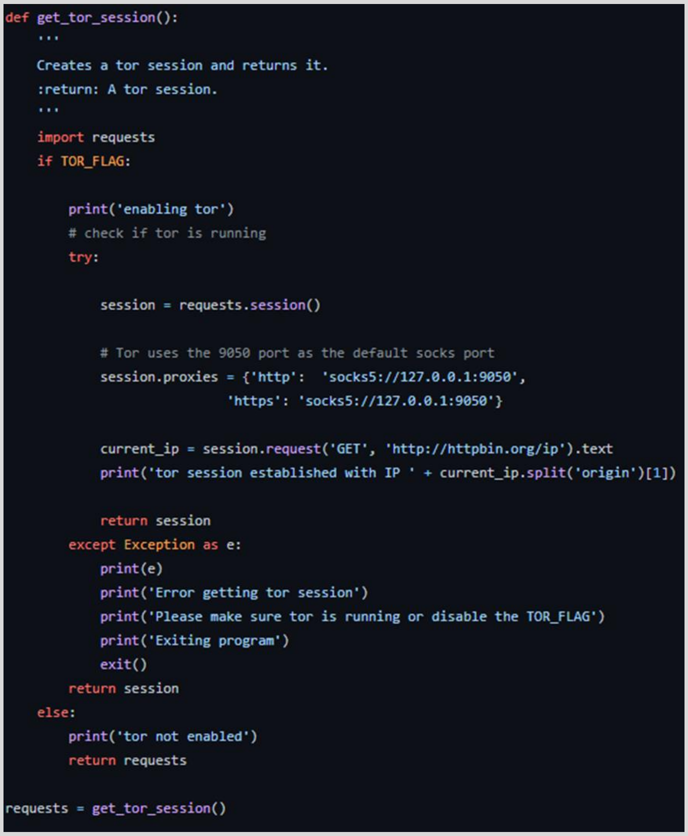
```
# TODO
# support sessions by checking for stored staff info and if they don't exist then get them from the website --Completed
# For each web request, save the outputed HTML file for future use and mining and to also reduce network traffic --Completed
# Tor connection option for the program to anonymize the traffic and avoid detection -- Completed
# setting up the session (visited urls )  -- Completed
```

Figure B-2



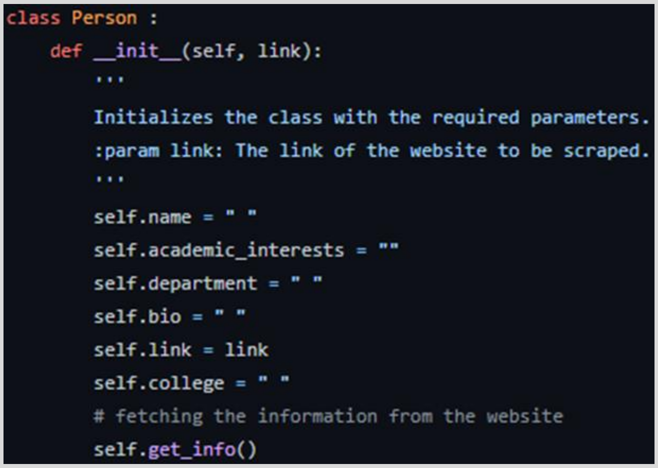
```
# Setting up variables to be used in the program
urls = []
people = []
UNCC={'Belk College of Business': {}, 'College of Arts + Architecture': {}, 'College of Computing & Informatics': {}, 'College of Education': {}, 'College of Health & Human Se
UNCC['Belk College of Business'] = {'Business Info Systems/Operations': {}, 'Economics': {}, 'Finance': {}, 'Management': {}, 'Marketing': {}, 'Turner School of Accountancy':{
UNCC['College of Arts + Architecture'] = {'Art & Art History': {}, 'Dance': {}, 'Music': {}, 'Performing Arts Services': {}, 'School of Architecture': {}, 'Theatre':{}}
UNCC['College of Computing & Informatics'] = {'Bioinformatics and Genomics' : {}, 'Bioinformatics Research Center': {}, 'Computer Science': {}, 'Software & Information Systems
UNCC['College of Education'] = {'Counseling': {}, 'Ctr For STEM Education': {}, 'Educational Leadership': {}, 'Middle Grades Secondary & K-12': {}, 'Reading & Elementary ED':
UNCC['College of Health & Human Services'] = {'Kinesiology': {}, 'Public Health Sciences': {}, 'School of Nursing': {}, 'School of Social Work':{}}
UNCC['College of Liberal Arts & Sciences'] = {'Africana Studies': {}, 'Anthropology': {}, 'Biological Sciences': {}, 'Chemistry': {}, 'Communication Studies': {}, 'Criminal Ju
UNCC['Lee College of Engineering'] = {'Civil and Environmental Engr': {}, 'Electrical & Computer Engineering': {}, 'Engineering Tech & Constr Mgmt': {}, 'EPIC': {}, 'Mech Engi
# a master array to store all the staff members
All_STAFF_INFOMATION = []
TOR_FLAG = False
```

Figure B-3



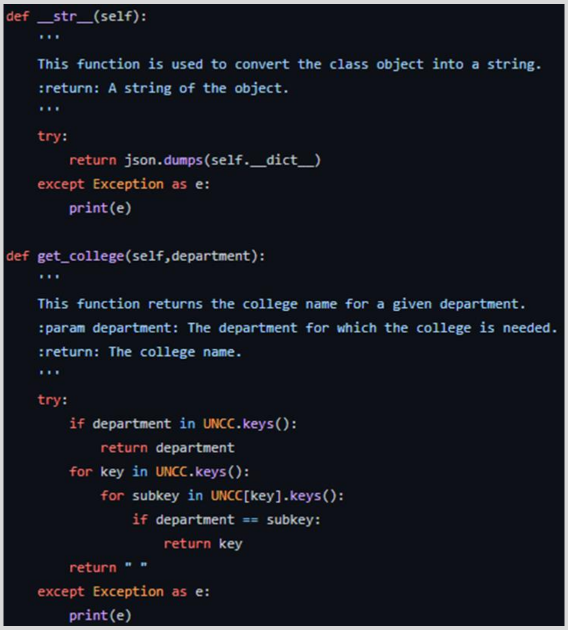
```
def get_tor_session():
    '''
    Creates a tor session and returns it.
    :return: A tor session.
    '''
    import requests
    if TOR_FLAG:

        print('enabling tor')
        # check if tor is running
        try:

            session = requests.session()

            # Tor uses the 9050 port as the default socks port
            session.proxies = {'http':  'socks5://127.0.0.1:9050',
                                'https': 'socks5://127.0.0.1:9050'}

            current_ip = session.request('GET', 'http://httpbin.org/ip').text
            print('tor session established with IP ' + current_ip.split('origin')[1])

            return session
        except Exception as e:
            print(e)
            print('Error getting tor session')
            print('Please make sure tor is running or disable the TOR_FLAG')
            print('Exiting program')
            exit()
        return session
    else:
        print('tor not enabled')
        return requests

requests = get_tor_session()
```

Figure B-4



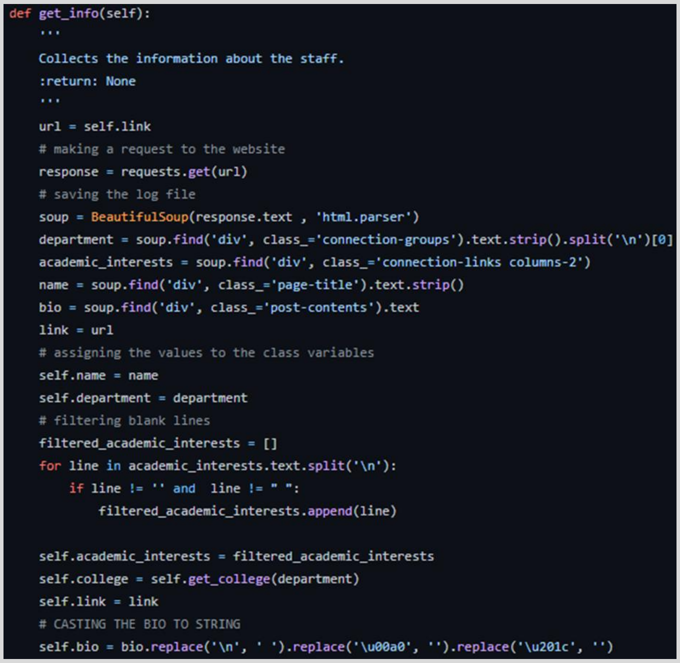
```
class Person :
    def __init__(self, link):
        '''
        Initializes the class with the required parameters.
        :param link: The link of the website to be scraped.
        '''
        self.name = " "
        self.academic_interests = ""
        self.department = " "
        self.bio = " "
        self.link = link
        self.college = " "
        # fetching the information from the website
        self.get_info()
```

Figure B-5



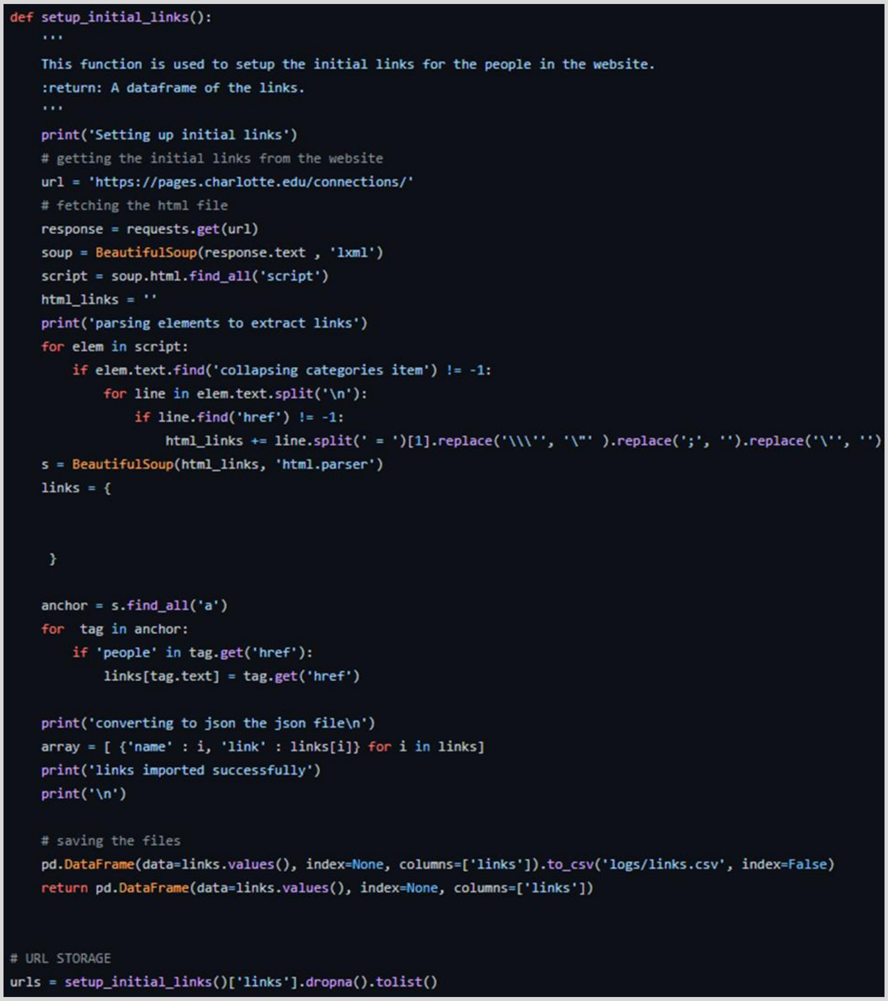
```python
def __str__(self):
    '''
    This function is used to convert the class object into a string.
    :return: A string of the object.
    '''
    try:
        return json.dumps(self.__dict__)
    except Exception as e:
        print(e)

def get_college(self,department):
    '''
    This function returns the college name for a given department.
    :param department: The department for which the college is needed.
    :return: The college name.
    '''
    try:
        if department in UNCC.keys():
            return department
        for key in UNCC.keys():
            for subkey in UNCC[key].keys():
                if department == subkey:
                    return key
        return " "
    except Exception as e:
        print(e)
```

Figure B-6



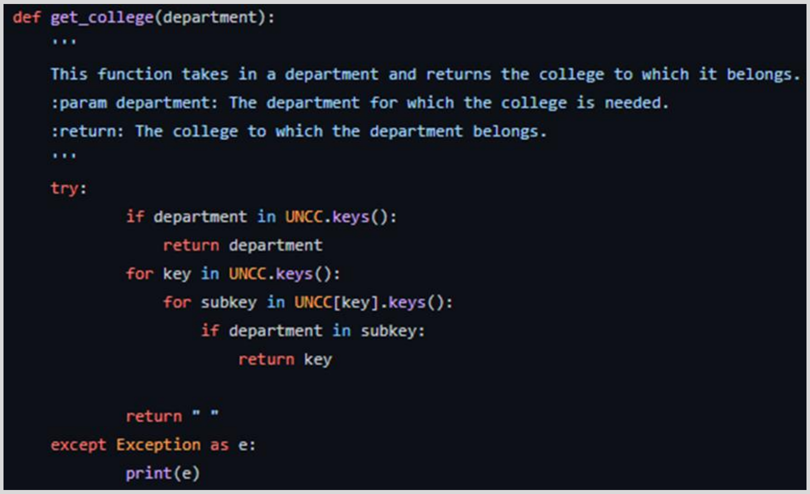
```python
def get_info(self):
    '''
    Collects the information about the staff.
    :return: None
    '''
    url = self.link
    # making a request to the website
    response = requests.get(url)
    # saving the log file
    soup = BeautifulSoup(response.text , 'html.parser')
    department = soup.find('div', class_='connection-groups').text.strip().split('\n')[0]
    academic_interests = soup.find('div', class_='connection-links columns-2')
    name = soup.find('div', class_='page-title').text.strip()
    bio = soup.find('div', class_='post-contents').text
    link = url
    # assigning the values to the class variables
    self.name = name
    self.department = department
    # filtering blank lines
    filtered_academic_interests = []
    for line in academic_interests.text.split('\n'):
        if line != '' and  line != " ":
            filtered_academic_interests.append(line)

    self.academic_interests = filtered_academic_interests
    self.college = self.get_college(department)
    self.link = link
    # CASTING THE BIO TO STRING
    self.bio = bio.replace('\n', ' ').replace('\u00a0', '').replace('\u201c', '')
```

Figure B-7



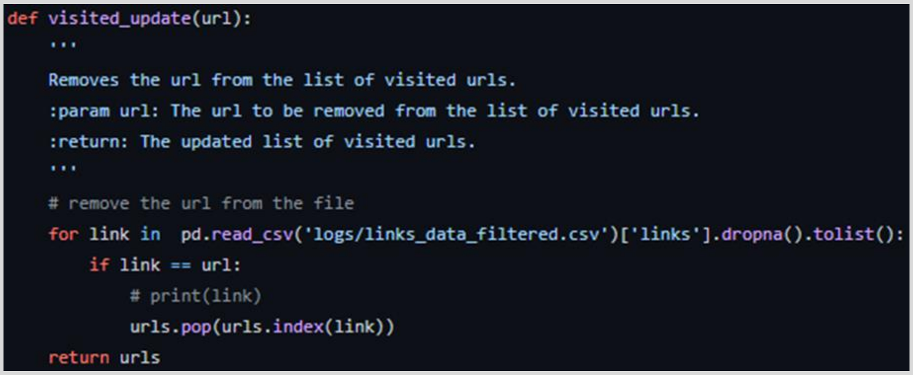
```
def setup_initial_links():
    '''
    This function is used to setup the initial links for the people in the website.
    :return: A dataframe of the links.
    '''
    print('Setting up initial links')
    # getting the initial links from the website
    url = 'https://pages.charlotte.edu/connections/'
    # fetching the html file
    response = requests.get(url)
    soup = BeautifulSoup(response.text , 'lxml')
    script = soup.html.find_all('script')
    html_links = ''
    print('parsing elements to extract links')
    for elem in script:
        if elem.text.find('collapsing categories item') != -1:
            for line in elem.text.split('\n'):
                if line.find('href') != -1:
                    html_links += line.split(' = ')[1].replace('\\\'', '\"' ).replace(';', '').replace('\'', '')
    s = BeautifulSoup(html_links, 'html.parser')
    links = {


     }

    anchor = s.find_all('a')
    for  tag in anchor:
        if 'people' in tag.get('href'):
            links[tag.text] = tag.get('href')

    print('converting to json the json file\n')
    array = [ {'name' : i, 'link' : links[i]} for i in links]
    print('links imported successfully')
    print('\n')

    # saving the files
    pd.DataFrame(data=links.values(), index=None, columns=['links']).to_csv('logs/links.csv', index=False)
    return pd.DataFrame(data=links.values(), index=None, columns=['links'])


# URL STORAGE
urls = setup_initial_links()['links'].dropna().tolist()
```

Figure B-8



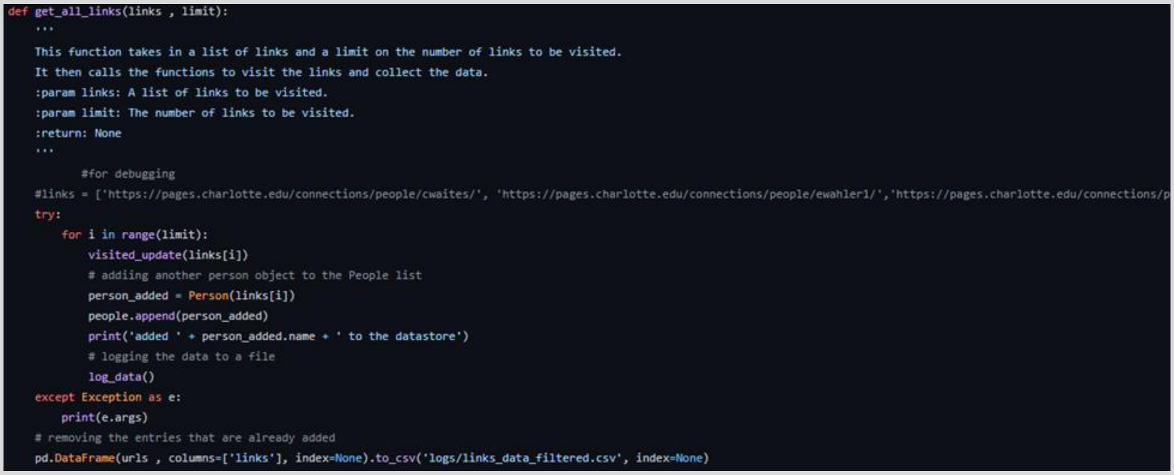
```
def get_college(department):
    '''
    This function takes in a department and returns the college to which it belongs.
    :param department: The department for which the college is needed.
    :return: The college to which the department belongs.
    '''
    try:
            if department in UNCC.keys():
                return department
            for key in UNCC.keys():
                for subkey in UNCC[key].keys():
                    if department in subkey:
                        return key

            return " "
    except Exception as e:
            print(e)
```

Figure B-9



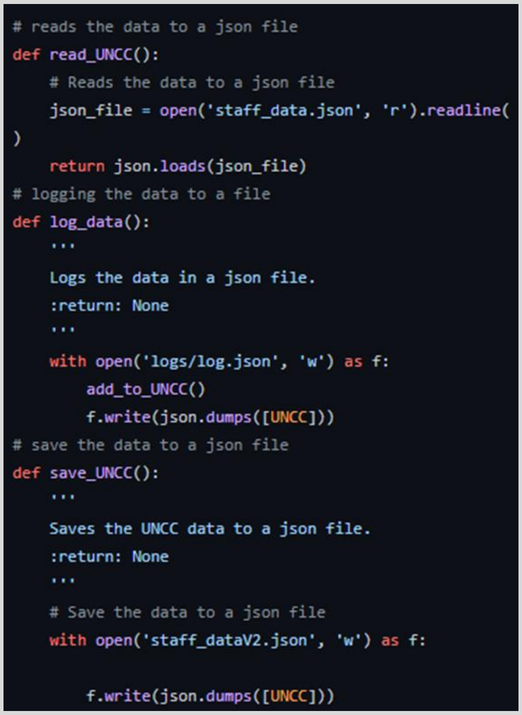
```
def visited_update(url):
    '''
    Removes the url from the list of visited urls.
    :param url: The url to be removed from the list of visited urls.
    :return: The updated list of visited urls.
    '''
    # remove the url from the file
    for link in  pd.read_csv('logs/links_data_filtered.csv')['links'].dropna().tolist():
        if link == url:
            # print(link)
            urls.pop(urls.index(link))
    return urls
```

Figure B-10



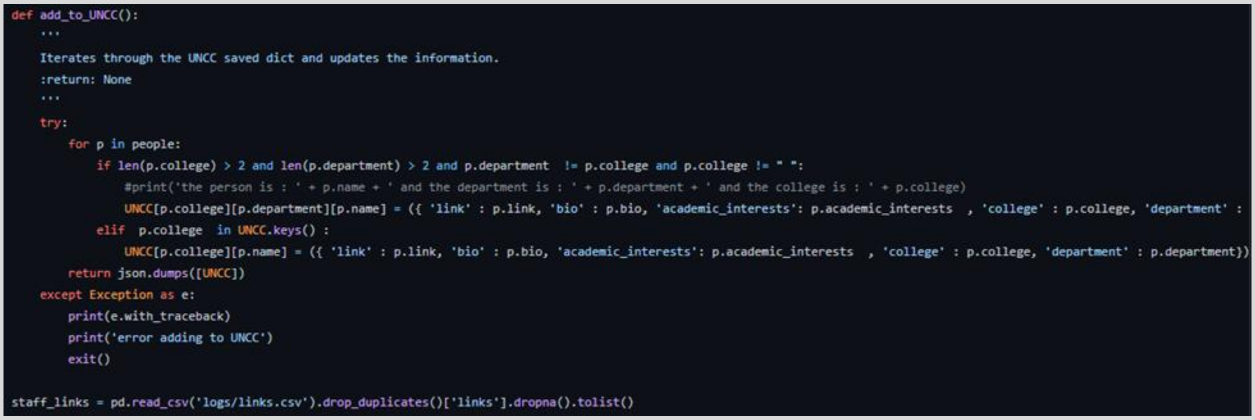
```
def get_all_links(links , limit):
    '''
    This function takes in a list of links and a limit on the number of links to be visited.
    It then calls the functions to visit the links and collect the data.
    :param links: A list of links to be visited.
    :param limit: The number of links to be visited.
    :return: None
    '''
        #for debugging
    #links = ['https://pages.charlotte.edu/connections/people/cwaites/', 'https://pages.charlotte.edu/connections/people/ewahler1/','https://pages.charlotte.edu/connections/p
    try:
        for i in range(limit):
            visited_update(links[i])
            # addiing another person object to the People list
            person_added = Person(links[i])
            people.append(person_added)
            print('added ' + person_added.name + ' to the datastore')
            # logging the data to a file
            log_data()
    except Exception as e:
        print(e.args)
    # removing the entries that are already added
    pd.DataFrame(urls , columns=['links'], index=None).to_csv('logs/links_data_filtered.csv', index=None)
```

Figure B-11



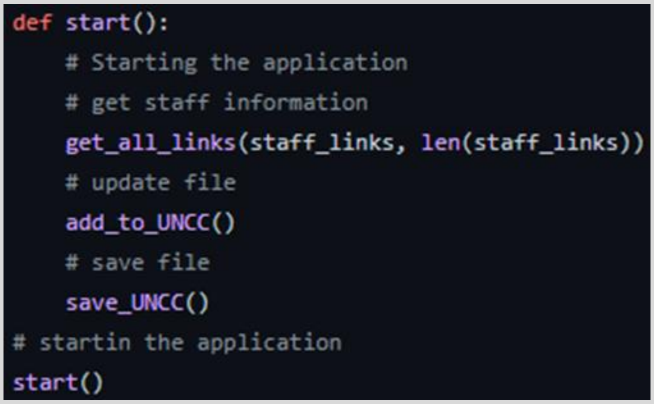
```
# reads the data to a json file
def read_UNCC():
    # Reads the data to a json file
    json_file = open('staff_data.json', 'r').readline(
)
    return json.loads(json_file)
# logging the data to a file
def log_data():
    '''
    Logs the data in a json file.
    :return: None
    '''
    with open('logs/log.json', 'w') as f:
        add_to_UNCC()
        f.write(json.dumps([UNCC]))
# save the data to a json file
def save_UNCC():
    '''
    Saves the UNCC data to a json file.
    :return: None
    '''
    # Save the data to a json file
    with open('staff_dataV2.json', 'w') as f:

        f.write(json.dumps([UNCC]))
```

Figure B-12



```
def add_to_UNCC():
    '''
    Iterates through the UNCC saved dict and updates the information.
    :return: None
    '''
    try:
        for p in people:
            if len(p.college) > 2 and len(p.department) > 2 and p.department  != p.college and p.college != " ":
                #print('the person is : ' + p.name + ' and the department is : ' + p.department + ' and the college is : ' + p.college)
                UNCC[p.college][p.department][p.name] = ({ 'link' : p.link, 'bio' : p.bio, 'academic_interests': p.academic_interests  , 'college' : p.college, 'department' :
            elif  p.college  in UNCC.keys() :
                UNCC[p.college][p.name] = ({ 'link' : p.link, 'bio' : p.bio, 'academic_interests': p.academic_interests  , 'college' : p.college, 'department' : p.department})
        return json.dumps([UNCC])
    except Exception as e:
        print(e.with_traceback)
        print('error adding to UNCC')
        exit()

staff_links = pd.read_csv('logs/links.csv').drop_duplicates()['links'].dropna().tolist()
```

Figure B-13



```
def start():
    # Starting the application
    # get staff information
    get_all_links(staff_links, len(staff_links))
    # update file
    add_to_UNCC()
    # save file
    save_UNCC()
# startin the application
start()
```

Figure B-14