Project Gemini: Web Crawling and Data Visualization

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

Table of Contents:

Project Overview

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:

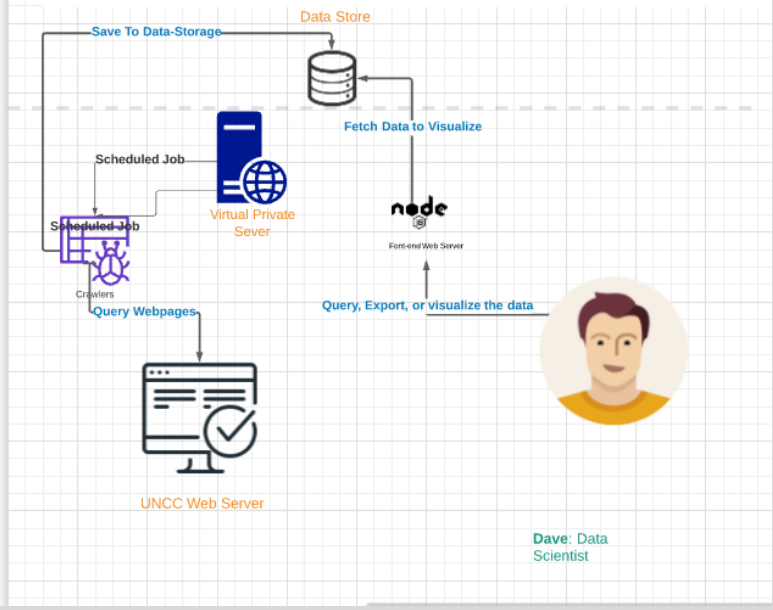


Figure C-1: Product Specifications

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.

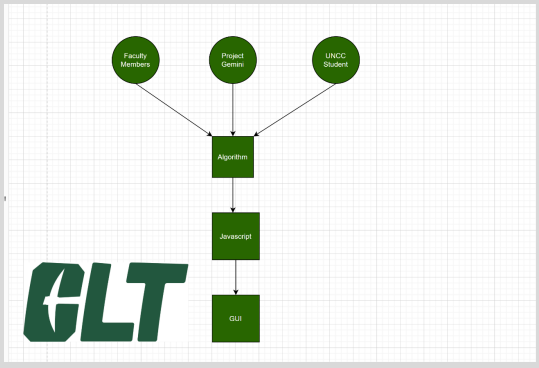
Plan of Action:

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.

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

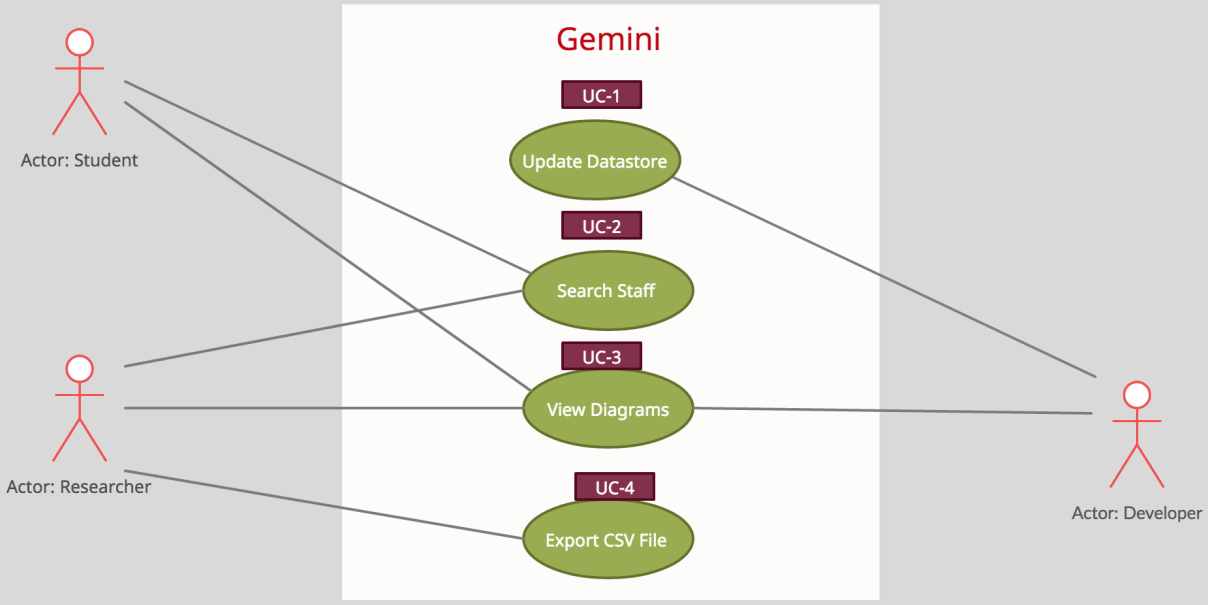


Behavioral Diagrams:

A Brief Use-Case

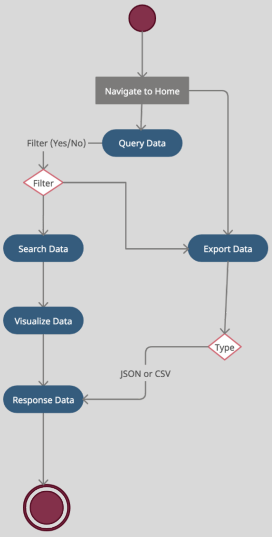
Actors:
Developers that support the application and update the datastores.
Researchers that want to use the datastore in JSON or CSV formats

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



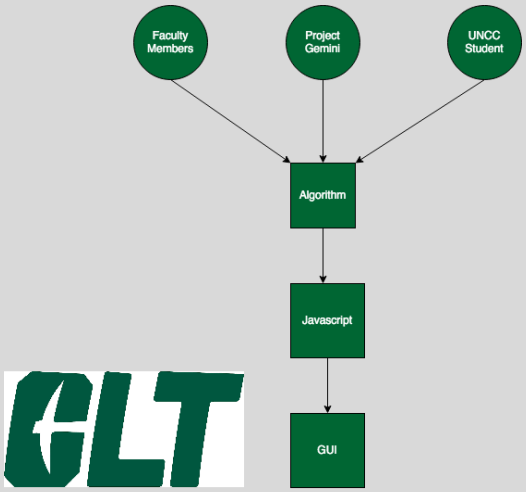
Link: Here

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

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)*

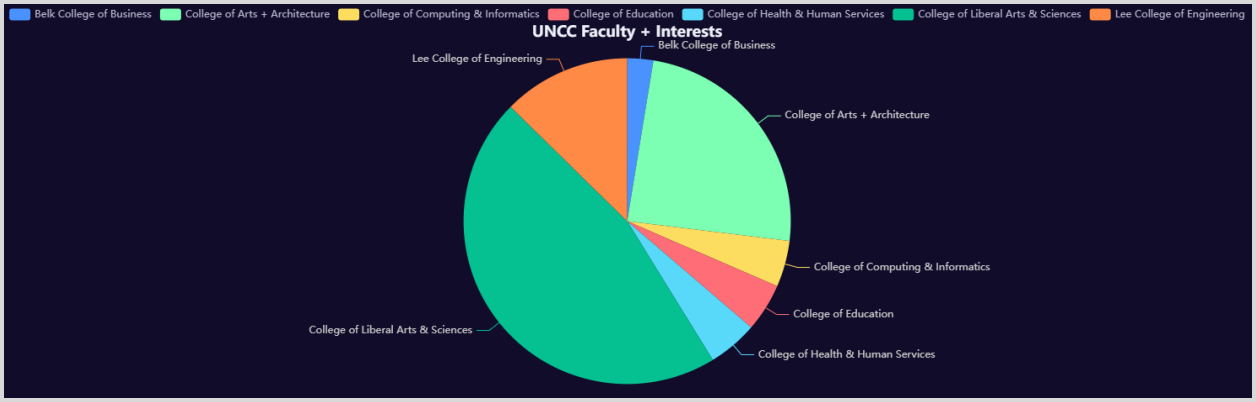
1. *Backend: Python implementation of data crawling (Sprint 2 and Sprint 3)*

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.

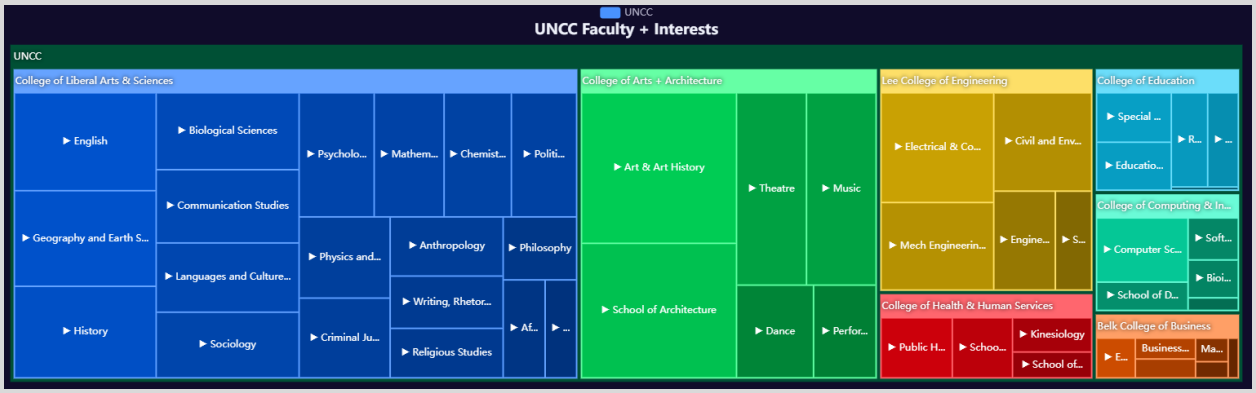
Display 1: Pie Chart



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

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

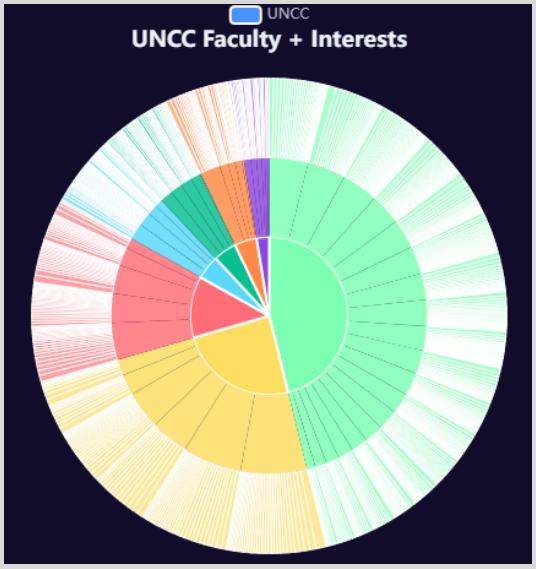
Display 2: Tree Map



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: {}
  }
]
```

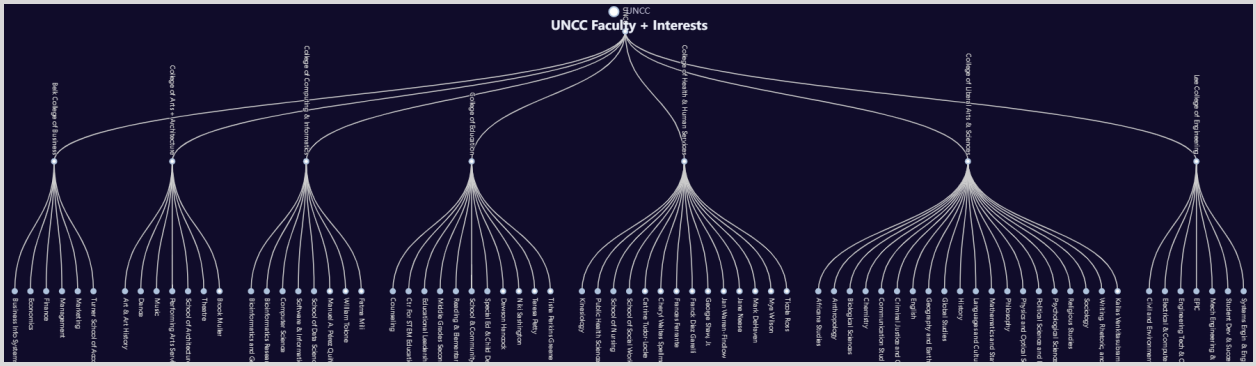
Display 3: Sunburst



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:



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");
```

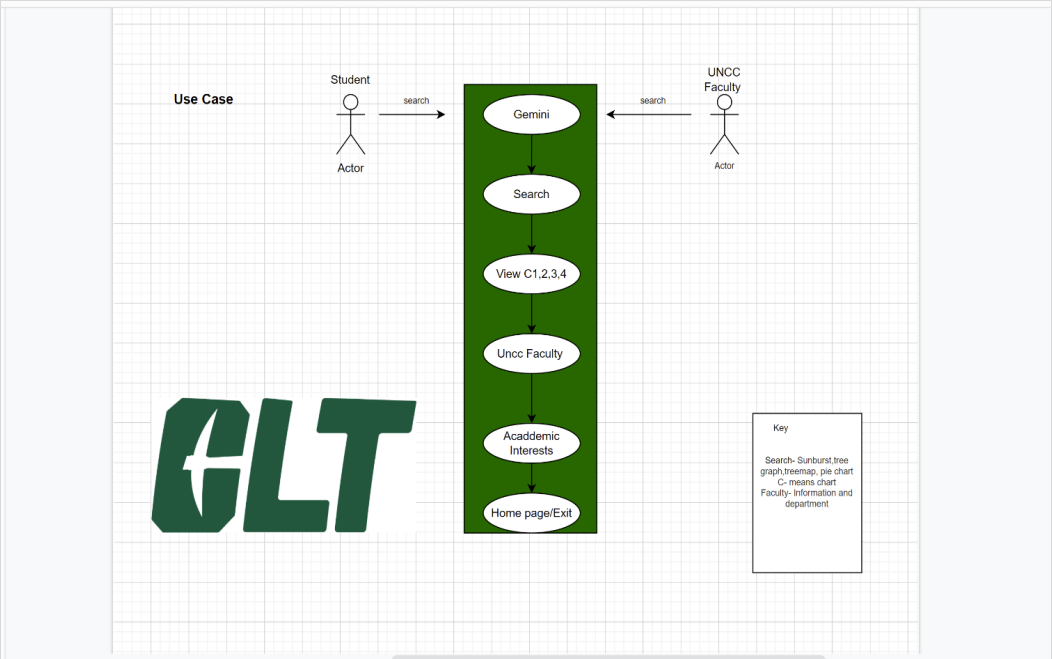
Back End

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

```python
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

```python
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

```
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

```
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

```python
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

```python
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

```python
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

```python
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

```python
# 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