Aiden Bentz

Matthew Devaney

Connor Kress

Shenyu Zhou

Dr. Jason Harrington

MAD2502 Capstone Project

11 December 2023

Classpy

Project Overview and Introduction:

When planning on what courses to take in the next semester, students nowadays frequently need to browse several websites for ratings and course details and may manually write down tentative schedules on paper which can be confusing and laborious. In this capstone project, we aim to address such a challenge that resonates with many college students by attempting to optimize academic schedule planning. Our project attempted to facilitate academic schedule planning by providing a course search feature with RateMyProfessor ratings and textbook details, as well as a schedule-building simulator that calculates the distances between classes. For this, we utilized JSON files for building and bus stop information at the University of Florida, along with CSV files for course and professor ratings, obtained through web scraping. A variety of public Python packages were essential in our development process, including Playwright, BeautifulSoup, Selenium, and Requests for robust web scraping; Pandas, Typing, os, and abc for core utilities; Concurrent, asyncio, and threading for enhanced concurrency and

performance in web scraping; Tkinter and Matplotlib for interactive user interfaces and visualizations. These tools collectively supported a wide range of functionalities in our project. Additionally, all other algorithms, except for the functionalities provided by these packages, were originally developed by our team members, ensuring no external open-source repositories were used in this project. AI tools such as ChatGPT were occasionally consulted for syntactic correction and PEP-8 formatting, enhancing the clarity and effectiveness of our code.

For the distribution of responsibilities, Aiden was primarily responsible for building the scrapper for GatorEval, Connor for programming the course query and schedule builder applications, Shenyu for constructing the one.uf schedule and ratemyprofessor.com Selenium scrapper and the GUI for course query function, and Matthew for optimizing scraping algorithms. Besides, each member has contributed to each other's primary responsibilities.

Project Description:

Our capstone project, designed for students at the University of Florida, is a Python-based application with modular design that amalgamates functionalities to achieve our objectives with optimizing schedule planning. The project is structured into multiple interconnected modules, each contributing to a specific of the tool's overall effectiveness and user experience.

For a brief and holistic breakdown of the structure, the 'core' sub-module is the central anchor of the application, encapsulating the primary entities. Files "class_.py" and "course.py" define class and course structure and attributes like course code, names, professors, etc., setting the basis for querying and schedule building. "schedule.py" includes the functions and logistics

of schedule creation, enabling adding, removing, and modifying classes for a sample schedule. “textbook.py” and “textbook_collection.py” manage textbook data associated with each course with details like ISBN, title, and authors. “course_req.py” ensures that course prerequisites are met, which is auxiliary to schedule building to prevent conflicts.

In completing web scraping and data handling both during data preparation and interactively with GUI, three sub-modules were created. 1. The ‘data’ sub-module holds “find_building.py” and “locations.py” that handle the mapping of campus buildings and bus stops used for the distance calculations in schedule building. sub-module for ‘data’, ‘raw_data’ contains scraped data like “course_with_scores.csv”, JSON files of building and bus stop information, and basic configurations and constants for retrieval and use. 2. The ‘scraping’ sub-module gathers all of the web scrapers used for preparation like the ones for ratemyprofessor.com and GatorEval and the data handling algorithms that produced data files stored in ‘raw_data’; and the interactive scraping method course_query() within ‘course_data.py’. 3. The ‘parsing’ submodel converts raw data into a usable format. ‘parse_reqs.py’ and ‘parse_time.py’ parse course information and transform raw scraped data by the real-time interactive method into a structured format suitable for the core and schedule sub-modules.

In order to calculate the building distances between classes, the ‘locations’ sub-module that deals with the spatial aspect of the university campus was formulated. “building.py”, “bus_stop.py”, “coordinates.py”and “classroom.py” define the spatial entities on campus as classes with coordinates and characteristics for better comparison. “coord_distance.py” utilizes

geographical location data to compute distances between classes, targeting to optimize the practicality of the schedule builder.

The sub-module ‘utils’ supports the main application by providing additional functionalities like logical operations, UI clearing and refreshing, and Haversine formula approach of calculating geographical distances.

Inside the main module ‘classpy’ locates the user interface and integration programs. “GUI.py” offers a user-friendly interface of the interactive course_query() that can perform searches and present course information with more clarity. Functionality integration scripts ‘class_functions.py’ and ‘schedule_builder.py’ are keys to integrate various functionalities like course selection, schedule building, and distance calculation, ensuring the execution of the application features. As the central hub for the application, ‘main.py’ script integrates all sub-modules and components, orchestrating the overall functionality and user flow of the project.

The modular design confirms maintainability and scalability, allowing for future enhancements and adaptations for evolving user needs. Detailed code description and methodological explanation with examples will be provided in the following sections.

Methods:

At the initial stage of our project, we planned to create a comprehensive database that merges course information from University of Florida one.uf website with professor ratings from RateMyProfessor.com that will be used to select courses and generate schedule suggestions. To achieve this objective, we researched and decided to use BeautifulSoup as this is one of the

easiest and most popular choices for parsing static HTML content, in addition with Selenium for browser automation to access the webpage since both websites are dynamically loaded.

Beginning with “oneuf_scraping.py”, the libraries usage includes ‘csv’ to write scraped data into a CSV file; regex for string manipulation using regular expression; BeautifulSoup for parsing HTML; ‘selenium’ for web automation to interact with web page, with internal features such as ‘webdriver’, ‘By’, ‘WebDriverWait’, and ‘expected_conditions (EC)’ to control the browser and for locating elements and waiting for certain conditions on web pages.

The script starts by launching a Selenium-driver browser session to access the one.uf schedule search URL. It then repeatedly clicks on a “View More Results” button to ensure all courses are loaded onto the page. During this process, a try-except-finally structure is introduced to handle exceptions like ‘NoSuchElementException’ and ‘TimeoutException’ that indicate all courses have been loaded without halting the execution.

Once the entire list of courses is displayed, the script uses BeautifulSoup to parse the HTML content. It systematically extracts crucial course details following the process of first locating the course and then scrapping all sessions associated with that course. Data like course title, class digit, and instructor are identified based on the class tag like “div” or “p” and class name. Special cases like credits, department, final exam info, and class dates are handled by the specially design function “_find_sibling_div_by_label(session, label) -> Optional[str]” which find a specific “div” element that follows a “span” with a specified label text to extract. For meeting times and location, the text extracted is being further cleaned with regex for clarity. Moreover, for meeting type, it is managed with a if statement that if meet is online and no meeting time available, meeting type should be asynchronous; if meet is online and no location

available, meeting type will be remote. Eventually, the information scrapped is compiled into a CSV file named “course_title.csv” for future use.

The scraping of the UF Schedule of Courses was later moved to using ‘Playwright’ for both the browser automation and HTML parsing because of ease of use and speed when a user of the ClassPy module makes a query. This version was moved into the `course_query(...)` method also in the scraping sub-module. This method built off the existing course and class/session data scraping functionalities to also scrape ‘Textbook’ instances. This change means that more than a single web-page would have to be loaded to perform all of the required scraping. This meant that the asynchronous API for Playwright would have to be used in conjunction with ‘asyncio.gather(...)’ to load the dozens of required web-pages concurrently.

When scrapping the ratemyprofessor.com, the logic of “ratemyprof_scrape.py” is very similar to using beautifulsoup and selenium. Yet, in order to increase efficiency and speed, in “professors_CSV.py”, “concurrent” and its method “ThreadPoolExecutor” were employed to implement multithreading to achieve parallelism for faster scraping. Each unique professor was being searched and scrapped in ratemyprofessor.com, and the information including num_rating, rating_value, and would_take_again percentage was stored in a CSV file “professors.csv”.

This csv was then cleaned and rows with 0 ratings were dropped. Rating value and num_rating were normalized by dividing their max values. Would_take_again percentage was converted to float type and divided by 100. A score, rounded to 2 decimal place, was then calculated and assigned based on the following weight and formula “Score = 0.4*(normalized_rating_value) + 0.4*(normalized_num_ratings) + 0.2*(would_take_again)”. The score updated file was stored in “professors_cleaned.csv”.

Later, “course_titles.csv” and “professors_cleaned.csv” were merged in “merge_csv.py”. A score was added for each professor if there is a match in both name and department, considering professors with the same name while in different departments. Professors without a match were assigned with a score by imputation of the mean. The final CSV file was named “course_with_score.csv”.

Next we have “GatorEvalsScraper.py”, which uses a selenium webdriver to open the gator evals Tableau page in a Google Chrome window. Most of the info stored in the Tableau html is hidden and accessed internally, meaning traditional scraping methods were ineffective. The information is all readily available publicly, but is not provided in csv format and only available for viewing on the Tableau website itself, meaning a mass screenshot library is the easiest way to access it. The original plan for the scraper was to use the pytesseract ocr image_to_string() function to copy the data from the screenshots into a csv, but the accuracy of the pytesseract extracted data was extremely inconsistent. This is either due to the low image quality of the screenshots, complicated color schemes, or evidence of a flaw in the pytesseract ocr. Even after image simplification procedures were implemented, like converting the image to grayscale, increasing the contrast, and applying thresholding, the text recognition left much to be desired, meaning other methods were necessary.

Selenium web driver is used to simulate user input to open the Instructor_Name dropdown menu to load the required data. After the dropdown menu is open, pyautogui is used to simulate a click and drag on the dropdown scrollbar to load all instructor names. Each instructor name has a unique number assigned to it in the html based on the alphabetical order of their

name, these numbers range from 1-5223. Selenium web driver is used to click each professor in the range individually to display their gator evals data and take a screenshot of the screen.

```
# Select each teacher in drop down menu one by one and take a screenshot
for x in range(1, 5224):

    # Move mouse to hover over drop down menu scroller
    pyautogui.moveTo(x=990, y=425)

    # Drag scroller to bottom of drop down menu to load all teacher options
    pyautogui.dragRel(xOffset=0, yOffset=570, button='PRIMARY', duration=3.0)

    # Select teacher
    options = WebDriverWait(driver, 30).until(
        EC.element_to_be_clickable((By.ID, f'FI_sqlproxy.04pjtia0xer5jm1ctta0u0ec5L1n,none:INSTRUCTOR_NAME:nk16126187992227925297_15952188591581136529_{x}'))
    )
    options.click()

    # Find teacher's name in html
    element = WebDriverWait(driver, 30).until(
        EC.visibility_of_element_located((By.ID, f'FI_sqlproxy.04pjtia0xer5jm1ctta0u0ec5L1n,none:INSTRUCTOR_NAME:nk16126187992227925297_15952188591581136529_{x}'))
    )
```

It's at this point that selenium is also used to scrape the instructor's name from the element text corresponding to their number. This name is then run through the sanitize_filename() function to ensure it's in the right format to be made into a filename. This function replaces spaces with underscores and removes any punctuation. The screenshot is then saved with the sanitized filename using os to a folder on the desktop.

```
#Function to remove punctuation and whitespace from teacher's names so they are acceptable as file names
def sanitize_filename(filename):
    # Replace spaces with underscores and remove invalid characters
    sanitized = re.sub(r'[\\/*?:"<>|,]', "", filename)
    sanitized = sanitized.replace(' ', '_')
    return sanitized
```

The function then repeats this process for all {x} in the range 1-5224 until all teachers are collected. After this the crop_images_in_folder() function is used on the folder with all of the collected screenshots. This function uses pillow to open and crop the image and os to construct the file path for the source and destination folder. The images are cropped to the dimensions of the specified "crop_coordinates", which in this case is focused on the area of the gator evals data graph in the screenshot.

```python
#Images show the entire gatorevals screen so they need to be cropped to focus on the chart
def crop_images_in_folder(source_folder, dest_folder, crop_coordinates):

    # Create the destination folder if it does not exist
    if not os.path.exists(dest_folder):
        os.makedirs(dest_folder)

    # Get a list of files in the source folder
    for file_name in os.listdir(source_folder):
        # Construct the full file path
        file_path = os.path.join(source_folder, file_name)

        # Check if it's a file and not a directory and has an image extension
        if os.path.isfile(file_path) and file_path.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
            # Open the image
            with Image.open(file_path) as img:
                # Crop the image using the provided coordinates
                cropped_img = img.crop(crop_coordinates)

                # Construct the full path for the destination of the cropped image
                dest_file_path = os.path.join(dest_folder, file_name)

                # Save the cropped image
                cropped_img.save(dest_file_path)
                print(f"Cropped image saved as {dest_file_path}")

# Define the source and destination folders
source_folder = 'C:\\Users\\aiden\\OneDrive\\Desktop\\python project'
dest_folder = 'C:\\Users\\aiden\\OneDrive\\Desktop\\python project'

# Define your crop coordinates (left, upper, right, lower)
crop_coordinates = (0, 0, 1000, 825)

# Call the function to crop the images
crop_images_in_folder(source_folder, dest_folder, crop_coordinates)
```

After this, the compress_images_in_folder() function is called on the screenshot folder to decrease its size a bit. This function uses os to get a list of all of the files in the folder and construct the file path. It then uses pillow to open the image, convert it to RGB to make it easier for display, and compress the image with a quality rating of 60 out of 100. The function then uses os again to add the compressed image to the destination folder.

```python
# Compresses folder from around 500mb to 350mb
# The setting for quality, set to 60 here, can be lowered or raised if needed
def compress_images_in_folder(source_folder, dest_folder, quality=60):

    # Create the destination folder if it doesn't exist
    if not os.path.exists(dest_folder):
        os.makedirs(dest_folder)

    # List all files in the source folder
    files = os.listdir(source_folder)

    # Loop over all files in the directory
    for file_name in files:
        # Construct full file path
        file_path = os.path.join(source_folder, file_name)

        # Check if the file is an image
        if file_path.lower().endswith(('.png', '.jpg', '.jpeg')):
            with Image.open(file_path) as img:
                # Convert the image to RGB mode (this step is necessary for .png files to be saved as .jpeg)
                if img.mode in ("RGBA", "P"):
                    img = img.convert("RGB")

                # Construct the full path for the destination of the compressed image
                dest_file_path = os.path.join(dest_folder, file_name)

                # Save the image with the desired compression
                img.save(dest_file_path, quality=quality, optimize=True)
                print(f"Compressed image saved to {dest_file_path}")

# Define your source and destination folders
source_folder = 'C:\\Users\\aiden\\OneDrive\\Desktop\\python project'
dest_folder = 'C:\\Users\\aiden\\OneDrive\\Desktop\\python project'

# Call the function to compress the images
compress_images_in_folder(source_folder, dest_folder, quality=60)
```

The end goal of the “GatorEvalsScraper.py” program is to create a file folder to be interacted with by the “gator_evals_searcher.py” program. This program accepts a user inputted file folder path and a filename in the “Firstname Lastname” format and then searches through the inputted screenshot folder for a file matching the inputted name but in the “Lastname_Firstname” format. When the inputted teacher is found, pillow is used to open and show the image. If no matching image is found, the program returns a string message that no image was found.

```python
from PIL import Image
import os

def gator_evals_searcher(folder_path, input_name):

    # Rearrange the input name from "Firstname Lastname" to "Lastname_Firstname"
    name_parts = input_name.split()
    if len(name_parts) == 2:
        search_name = f"{name_parts[1]}_{name_parts[0]}".lower()
    else:
        print("Please enter the name in 'Firstname Lastname' format.")
        return False

    # Search for an image with the rearranged name
    for file in os.listdir(folder_path):
        if file.lower().startswith(search_name) and file.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
            image_path = os.path.join(folder_path, file)

            # Open and show the image
            with Image.open(image_path) as img:
                img.show()
            return True

    # If no matching image is found
    print(f"No image found for '{input_name}'.")
    return False

# Ask the user for the folder path and the person's name
folder_path = input("Enter the path of the folder: ")
input_name = input("Enter the name as 'Firstname Lastname': ")

# Search for and display the image
gator_evals_searcher(folder_path, input_name)
```

The GUI.py" is designed to offer an intuitive and interactive platform for users to query and view detailed course information. The script is built with the “tkinter” library, which is one of the easiest GUI toolkits and is sufficient for basic GUIs.

Initially, it reads course score data from the “course_with_score.csv” using “pandas”, converting this data into a dictionary that links class digits to their scores. The “CourseQueryGUI” class centralizes the GUI’s setup and configures various input widgets such as entry fields for user inputs to be used for performing queries with course_query() function, which was defined in “course_data.py”. The call of the query function integrated with the threading package for managing background tasks efficiently ensures that the application remains responsive and quick, even while handling resource-intensive data retrieval operations.

```
def on_search_click():
    """Runs a course query on a new thread."""
    thread = threading.Thread(target=threaded_course_query)
    thread.start()
```

The results of a search query are displayed in a “treeview” widget, outlining essential course details like code, title, and credits in tabular display. The sessions associated with the course are showcased in a listbox with session-specific information like instructor, class digit, and score, obtained in the earlier defined dictionary. Upon selecting and clicking on a session in the listbox, there opens a new window with comprehensive course details like course description and prerequisites. To ensure that the window opens with information specific to the selected session, the script utilizes a nested function named on_session_select(event). This function employs the curselection() method, inherent to tkinter, to accurately identify and retrieve the index of the chosen session.

```
def on_session_select(event):
    selected_index = lb.curselection()[0]
    selected_session = sessions[selected_index]
```

Results and Discussion:

In the evaluation of our project, we concentrated on assessing the effectiveness and usability of our application against our initial objective: aiding students in academic schedule planning. The project's final output comprised several crucial components, each integral to the application's overall functionality.

A key feature is the schedule builder, designed to enable users to simulate their academic schedules. Despite the absence of a dedicated graphical user interface, this component remains functional and user-friendly, providing basic yet effective visualization capabilities. This design choice ensured that the tool remained accessible and practical for users.

Another significant achievement is the development of a GUI for course querying. Its intuitive design, enhanced by the inclusion of session scores and detailed session information, greatly simplified the course selection process. This feature allowed for a more informed decision-making experience for users, aligning well with our project's goals.

Supporting these main functionalities, we created several web scrapers to extract data from one.uf schedules, RateMyProfessor, and Gatoreval. These scrapers were crucial in collecting data necessary for building the application's algorithms and ensuring comprehensive information availability.

Overall, our application effectively met the set initiative, marking the project as a success. While there are areas for improvement, the project lays a solid foundation for future development. It not only achieves its primary goal of assisting students in schedule planning but also establishes a robust framework for potential enhancements.

Conclusion and Future Work:

Overall, the final application has limitedly fulfilled the objective of our project. By amalgamating data from various sources including RateMyProfessor and one.uf course search, we've crafted an interactive course search GUI that allows users to search for courses and receive insights from professor ratings. Moreover, the addition of a schedule builder makes this project even more useful in assisting students with effective academic planning.

Despite some progress, there's ample room for growth and refinement. Looking forward, several enhancements could be made to strengthen the performance and elevate the utility and user experience for this application.

1. The current GUI, while functional, could be made more professional and aesthetically pleasing. By incorporating advanced filtering options, for example for the scoring weights and adopting more complex libraries like PyQt/PySide, the user interface could be transformed into a more engaging and efficient portal for course advising.
2. A graphical interface specifically for the schedule builder would be a valid addition. This would allow users to visually and comfortably organize their courses, thereby enhancing the ease and effectiveness of the planning process.
3. Incorporating data from the one.uf degree audit system would offer a personalized course recommendation system. By scraping students' completed courses and analyzing remaining degree requirements, the tool could suggest courses that align with their academic progression, ensuring a smooth course selection process and being more like a real advisor.

4. Considering more preferences like meeting time into account for course suggestions would tremendously increase the practicality of our tool. Offering users choices with how to weight each parameter in scoring each session would also advance the degree of personalization and user experience.
5. The introduction of an automated scheduling feature would significantly upgrade our application. Utilizing algorithms and models to generate the most suitable schedule based on above personalized parameters - such as class times and degree requirements - would greatly assist users in making decisions about courses.
6. The current version did not include one planned aspect of textbook scraping which would be to automatically search for available online pdf versions of each textbook. (In fact, the caching for this is already set up as you can see in the console when running the ‘course_query’ function.)

In conclusion, the present iteration of the project has established a foundational framework for aiding students in their academic journey. The envisaged future directions, however, hold the potential to transcend the project's current scope, elevating it from a mere assignment to an actual meaningful tool for academic advising.

Reflection:

Through this project, our team has gained valuable insights into group programming and project management, glimpsing into the professional programming environment. This experience has taught us the importance of communication, adaptability, and consistency, which are crucial

for a career in technology. We have learned strategies to resolve emerging challenges as the project evolves.

Reflecting on these lessons, we recognize their immense applicability to our future endeavors. Python libraries and programming languages will change, and what we have learned about those packages may eventually become obsolete. However, the ability to work cohesively as a team, adapt to new tools, and manage complex projects effectively are competencies that endure and extend. This experience has been a significant step in preparing us for upcoming academic and professional challenges, equipping us with what is needed to be successful developers.

References:

"Beautiful Soup Documentation¶." *Beautiful Soup Documentation - Beautiful Soup 4.4.0 Documentation*, beautiful-soup-4.readthedocs.io/en/latest/. Accessed 11 Dec. 2023.

"Chatgpt." *ChatGPT*, openai.com/chatgpt. Accessed 11 Dec. 2023.

"Find and Rate Your Professor or School." *Rate My Professors*, www.ratemyprofessors.com/search/professors/1100?q=%2A. Accessed 10 Dec. 2023.

"Schedule of Courses." *ONE.UF*, one.uf.edu/soc/?category=%22CWSP%22&term=%222241%22&prog-level=%22UGRD%22. Accessed 11 Dec. 2023.

Public.Tableau.Com, University of Florida, public.tableau.com/app/profile/john.j6434/viz/GatorEvalsSpring2023ThreeYears/Dashboard1. Accessed 11 Dec. 2023.

"The Selenium Browser Automation Project." *Selenium*, www.selenium.dev/documentation/. Accessed 11 Dec. 2023.