Mauricio Pulido

I had to learn a good amount of how web scraping works. I learned how to look for specific tags and classes in an html page and then using those to get the information I need by using Beautifulsoup and selenium. These two can work together but sometimes one is better than the other. I had to learn what exception errors that selenium would throw if the tags and or the classes I was looking for did not exist. At first it was a lot to take in. The good thing is that both have a lot of documentation. This made it easy to figure out what exactly I was getting wrong or what the functions I was using returned. For example a BeautifulSoup function will return a Nonetype if what I am looking for is not found but on the other hand, selenium will return nosuchelementexception error which will cause the program to stop. To cover this issue I needed to import this error from the selenium library to be able to use try and except. This allowed me to continue to see if there are other errors besides this one and gave me a way to figure out the patterns the website we are using has. I also had to learn some of how PyQt 5 worked. I helped figure out some issues with the GUI. The part that gave me the most trouble was working with the buttons. They sometimes would be pressed when the program first ran and I helped figure that with lambda it was able to make it so the button would not start until it was clicked. I also learned how to deal with the PyQt DateTimeEdit widget. This one had a lot of possible things to do and it was tricky at first but we were able to get it working. It was definitely a challenge figuring out all the things needed for this project.

I helped come up with some requirements such as having google calendar connected so the events could be stored on the users calendar which means they would be able to see the events they want to go to without the need of having to reopen the desktop app. This would

allow us to store the events a user would want. Another requirement I helped with was the sort events requirement. This requirement would allow the user to filter through a list based on the type of event it is.

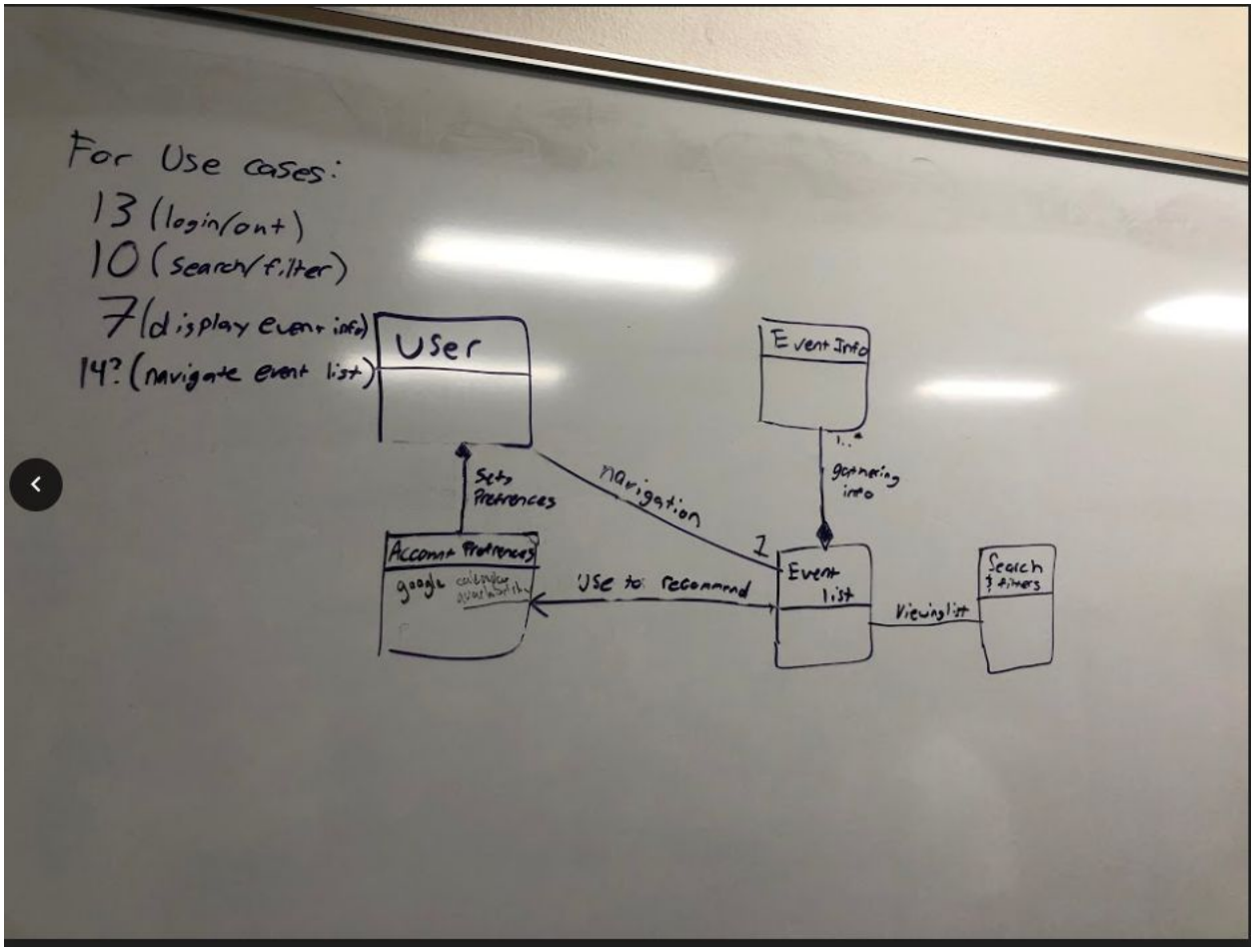
We split up the use cases and I worked on three of them. I worked on the recommended event list , Favorite events, and export calendar. Right now I am not sure if these four will be developed. The favorite events was completed and it would allow a user to favorite an event and there would be a page for the list of favorite events. From their the user would be able to click on the event if they wanted to. The user also has the option to right the event of the favorite page to remove the event from there.

I helped out with figuring out the UML of how the classes would be connected and why they would be connected the way we decided on. We knew the basic format we wanted to know but we weren't sure if how everything would fit all together. We came up with the Event Manager would be the main class that would handle creating the main list of events and displaying them onto the UI. Then User Would only see what events the manager would have and it would be filtered by what the user wanted. The account preference would be looked at by the event manager if the user wanted a list of recommended events. The main information of the events would not be shown until the user decide what event they would like to see.

The main part I have been working on are the web scraper functions that will be used by the Event Manager class which will have the main event list after using the first web scraper. The first web scraper is to find a bunch of events and have them but into a list the the Event Manager will have. This way we can just sort through that list rather than having to run the scraper everytime we need something. The second web scraper is for individual events. The link to the

event will be used and gather the rest of the event's information that was not available when the first scraper was ran. The second scraper will also only run when the user clicks on a specific event. This is done because the scraping takes a bit and not every event will be chosen. I also used a variable in the event class named Filled as a flag to figure out if an event had already been run with the second web scraper. If it was not filled it would run the web scraper portion and save everything into the correct variables in the event object. Then it would set its Filled variable to “filled” so that if the user selects the same event event again it would not need to run the scraper since it had all the data already.

UML for how the classes would interact with each other.



1. Recommend Event:

1.1. User clicks on the recommend button
1.2. System shows event based on past events/preferences
1.3. User searches the events
1.4. User can then click on an event
1.5. Possibility to remove a recommended event

2. Favorite event:
   2.1. Users searches for an event
   2.2. System shows a list of events
   2.3. User clicks on an event
   2.4. User can then favorite an event
   2.5. Favorite event shows up on a favorites page
3. Export calendar:
   3.1. User selects the calendar
   3.2. System shows the current month with plans
   3.3. User able to choose which month to download
      3.3.1. User could select more than one month
   3.4. User clicks to download the month
   3.5. System exports the selected month as a pdf

These are the use cases I did.

Links

https://selenium-python.readthedocs.io/

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

https://stackoverflow.com/questions/35819538/using-lambda-expression-to-connect-slots-in-pyqt

https://doc.qt.io/qtforpython/PySide2/QtWidgets/QDateTimeEdit.html

Code I worked on

```
options = Options()
options.add_argument('--ignore-certificate-errors')
options.add_argument('--incognito')
options.add_argument('--headless')
driver = webdriver.Chrome(executable_path=chromedriver, options=options)
driver.get("https://www.sandiego.org/explore/events.aspx")
LoadMore = driver.find_element_by_xpath('//button[@class="submit-button__secondary load-more"]')
LoadMore.click()
time.sleep(0.3)
LoadMore.click()
time.sleep(0.3)
soup = BeautifulSoup(driver.page_source,'lxml')
#print(soup)
eventlist = []
for links in soup.find_all('section',class_="result"):
    #print(links.find('div', class_="result__tag").get_text())
    E = Event(links.find('a',class_="result__title-link").get_text(), links.find('div',class_="result__dates").get_text(),
          "No Location","Not Know","Not Know", "not Know",links.find('div',class_="result__tag").get_text(),"Not Known",
           links.find('a',class_="result__cta-link").get('href'),"Not Filled","Not Contact",
              "No DateTime")
    FullEventList.append(E)
```

Finding the events and storing them into the main list to use.

```
    try:
        ShowMore = driver.find_element_by_xpath('//div[@class="links"]').click()
        time.sleep(1)
        try:
            LoadMore= driver.find_element_by_class_name('off').click()
            time.sleep(1)
        except ElementClickInterceptedException:
            pass
    except ElementClickInterceptedException:
        pass
    EventDescription = driver.find_element_by_class_name('header-component__content')
    # print(EventDescription.text)
    CompletedEvent.description = EventDescription.text
    popup.append(EventDescription.text)
except NoSuchElementException:
        #print("No Other Description found")
        popup.append("No description found")
```

This is when a certain event was chosen. This was hard because not all events had the same pattern which is why I needed to do try and except to catch errors that might happen.