Kevin Chen

Arthur Hua

Jared Solis

Sourav Shib

YouTube link: https://youtu.be/3w\_H-NOgEUQ

Goal:

Implement a website that dynamically displays data regarding the occupancy of different study buildings so that students can use it to determine where to study and how crowded a building would be on any given day. We built this website with the BU student in mind as there have been many a time when we walk across campus to our favorite spot to find it regretfully full of people. By giving these students access to various information about study spots around campus, it saves them the hassle of travel and can potentially give them new study spots to use.

Front End:

Necessary Imports: Flask

* Flask was used to implement the front end of the website by integrating Python and HTML to allow the user to create a dynamic website. Flask uses several website templates in order to set up the website’s framework. Flask was chosen with the intent to use RestAPI in order to obtain the data for the WiFi signals around BU campus. Flask offers API support while Django does not.

Programming Language(s) Used: HTML, Python

Back End:

Necessary Imports: classes, openpyxl, requests, os, location

* Classes were used in order to hold our data for each BU building and its corresponding floors with major study areas. The two classes that were used were a building and a floor class. Floor is instantiated first and then put into the Building class to account for buildings with different amounts of floors.
* openpyxl was used to read in an excel file into the back end of our code. In order to instantiate our dummy data in the program with initializing each variable in the main, openpyxl was used as it is able to read and write to Excel files. Once the variables were moved, they were used within the project as data for each BU building.
* Requests were used in order to obtain information about the user using the IP. This was used to request from an IP API that gave the IP of the user of the website. This is then used in order to obtain the latitude and longitude of the user for potential calculations concerning distance between the user and the buildings. It was found later that The API would sometimes return a Null Value and cause the Code to crash. As a result, the framework to get the location remained but was not used
* (Scrapped) was originally going to use SQL and Rest API to access and implement code from the BU databases, but upon meeting with our Contact from the IT department it was deemed too ambitious.
* OS was used to create the file paths to import images into the website

Programming Language(s) Used: Python

Coding Challenges:

(for challenges #1 - #3 see the lines 12-53 in main.py)

Challenge #1: Read in Data from the Excel Sheet

The solution: Using the load\_workbook() function we read in a book object from openpyxl and create a rows iterator that reads in the rows of the xlsx file. Then we create a for loop after skipping the header line that reads each line into a python array. We have now created a list of lists that each represent a floor.

Challenge #2: Sort said data into the different buildings:

The solution: the data is now floor by floor, but we need them separated by building. Luckily all floors in the building will have the same building name in the first index and are entered next to each other. So to sort the list we iterate through the list checking the first index of the sublist for the name. If the sublist name is the same as the last. We add it to a new group of lists that represent a building. If the name is not the same that means, there are no more floors for that building and we can then add the group list that represents the building to the Overall Sorted list and reset the group list.

Challenge #3: dynamically allocated this sorted list into Building and floor Objects

The solution: We now have a Sorted List that for each index has a list of lists that each hold floor information for a building. We first index through the Sorted List and while doing that index through the smaller list turning it into a list of floor objects rather than a list of lists. Then when that is completed we use the data for the 0th index in the lists of lists and the newly created list of lists to create the Building object. Here we use allvars[] and the varname portion of the data for the floors to dynamically name the Building Objects and then put them into a building list.

Challenge #4: Sort through and find the 5 Buildings that are the least full

The solution(see Location.py): create a function that takes a building list. Create a list of indeces from that list. And then create a separate list that stores the current building’s population divided by the capacity and multiply by ten to avoid digits being lost in an overflow error. Then zip the two together and use the sorted function to sort the information. Then return an array of only the first element in the zipped list.

How We Could Improve This Project

* Looking back on the project, there were many things that we enjoyed implementing and some things that we could add in the future. During our design process, we were playing around with the idea of using the current location of the user through an API that uses the IP of the user to obtain the latitude and longitude of the user. Because we are only focusing on BU study buildings, we can set static latitudes and longitudes and calculate the distances between the user and the buildings to find the closest buildings to the user. Unfortunately, during implementation, we went through some errors as the API seemed to be returning incorrect values.
* Another implementation we wanted to include was real-time data. We were given access to a RestAPI that included the WiFi usage within BU buildings and we hoped to use this to give an estimate to the amount of people within the building. We were unable to due to being swamped with work and due to none of us having any experience with Restful APIs.