WEB SCRAPER UTILIZES GOOGLE STREET VIEW IMAGES TO POWER A UNIVERSITY TOUR

Peiyuan Sun[1] and Yu Sun[2]

[1]Webb School of California, Claremont, CA 91711
[2]California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

Due to the outbreak of the Covid-19 pandemic, college tours are no longer available, so many students have lost the opportunity to see their dream school's campus. To solve this problem, we developed a product called "Virtourgo," a university virtual tour website that uses Google Street View images gathered from a web scraper allowing students to see what college campuses are like even when tours are unavailable during the pandemic. The project consists of 3/4 parts: the web scraper script, the GitHub server, the Google Domains DNS Server, and the HTML files. Some challenges we met include scraping repeated pictures and letting the HTML dropdown menu jump to the correct location. We solved these by implementing Python and Javascript functions that specifically target such challenges. Finally, after experimenting with all the functions of the web scraper and website, we confirmed that it works as expected and can scrape and deliver tours of any university campus or public buildings we want.

KEYWORDS

Web scraping, virtual tour, cloud computing

1. INTRODUCTION

Due to the Covid-19 pandemic, in-person campus tours are no longer possible. This means that a lot of seniors in high school will miss the opportunity to see their dream schools' campuses, which is significant in helping them form an idea of what college is like. Currently, there lacks a persuasive virtual tour platform on the internet that can deliver to students and families a comprehensive overview of what school campuses really look like. In recent years, more than two million American high school seniors that graduate each year enroll in a college or university, which is about 66% of each class. Although it is hard to predict the percentage of these two million students that tried to get a virtual tour, it is a reasonable deduction that many of them wanted at least some guide to know what kind of school they are applying to. In the future, the number will only rise, as both the total number of high school students and high school graduation rates are projected to increase. We predict that even after the pandemic, when colleges and universities reopen their campuses to the public, there will still be a need for virtual tours, as not every applying senior will be able to take a tour at every school that they are interested in. This leaves a strong demand for a platform hosting virtual tours of universities. The same applies to other public organizations or buildings. Not only do schools need virtual tours, public buildings as big as sports stadiums and airports, or as small as restaurants and stores, might also find themselves one day in need of developing a virtual tour to increase publicity or improve customer experience.

Some existing methods have already been used to address this problem. Some universities and public places have had outside companies develop virtual tours in the form of 360-degree images,

which allows the user to switch between images and navigate around the place [11, 12, 13]. However, this requires people to go around the place to take pictures, which is inefficient and often takes a lot of time and effort to complete. Also, it is hard to get permissions for such tours, especially during the pandemic, when schools are extra cautious about the potential spread of virus. Therefore, if images were not made prior to 2020, tours using this method would be extremely hard to develop during the pandemic.

In fact, immersive tours in the form of 360-degree pictures or, if possible, virtual reality, will produce the best results because they will let people have better long-term memory results [14]. One important goal of virtual tours is to make people remember the university, so immersive tours are better than 2D pictures at letting people experience the school visually; therefore, they are more preferable. However, because the technology of virtual reality is still not mature, there are obstacles in solely using virtual reality for campus tours [15]. For instance, a major challenge is that virtual reality is not yet popular enough so that everyone has a pair of goggles, which are often expensive. In addition, people likely will not buy a pair of goggles just to go on a school virtual tour. As a result, because 2D pictures cannot provide an immersive experience, and virtual reality has not been popularized enough for everyone, using 360-degree images seems to be the best approach. It is also better to develop a website than a phone application, because computer screens are larger and can give a better view, and people don't have to download an application in order to access the virtual tour.

In this paper, we follow the same line of research to provide immersive experiences in the form of 360 images. Our goal is to create a service that can generate simple tours within a short period of time and do not require people to travel to the campus during the pandemic. Our method is inspired by web scraping and Google Street View [1, 2]. Google Street View is a function developed by the Google Earth [3, 4] team to provide users with the opportunity to explore the world in the form of 360-degree images, which can be taken with special panoramic cameras able to capture the surrounding environment; an example of a panoramic camera would be Ricoh Theta [9,10]. Because there are many users of Google Street View all around the world [22], and many researchers also use it for environmental protection and social development [23, 24], Google's team constantly spend a lot of effort ensuring a clean and quality environment of the street view platform; the quality of pictures on Google Street View, as a result, serves as a baseline for our service as well. First, we developed a Python scraping script that finds 60 locations near a given university and an available 360-degree picture around each location [5, 6]. Web scraper is a popular method that gathers information from the internet for analytical or personal use. Second, after deleting the repeated pictures, we output the scraped information into a json file. Finally, we developed a website that pulls the information from the json file and shows the tour. Google has many users; some of them own 360 cameras and have taken and uploaded the pictures of a school campus before. This effectively solves the problem of needing people to go to the school and take pictures, as we can take advantage of those that are already available on the internet through a web scraper.

In two application scenarios, we not only demonstrated the effectiveness of our scraping script, but also proved the usefulness of our website in an actual environment. First, we generated a list of the top 100 schools in the United States, as well as every registered university or college in California. We then ran our python scraping script and waited for it to iterate through every school on the list. When each school was completed, the terminal would print the name of the school and its index in the list (the first school is 1, the second is 2, etc.) After the scraping part was done, two processing functions went through the results, checking for repeated pictures and changing the descriptions. We successfully acquired a file named "data.json" that contains all the scraped information. The scraping script worked just the way we expected it to. Secondly, we

copied the data.json file and put in the same directory as our website html file [7]. We pushed the information to GitHub, our website's server [8], and after a while, the data showed up on our domain "https://www.virtourgo.com." After we typed in the name of a random school in our list of schools and clicked the button to start the virtual tour, we were led to separate tour page, where we could not only see the street view panorama up and working, but also a places list dropdown menu that shows all the available buildings at the school, two descriptions that tell people the name of each building and the next one, and a button to go to the next building on the list. The website also worked just as we intended.

The rest of this paper is organized thus: Section 2 provides insight on the challenges we experienced during the development process of the web scraper and website; Section 3 focuses on the details and gives a guide for how we designed the different components, as well as how we solved the problems mentioned in Section 2; Section 4 gives an overall evaluation of our final product; Section 5 presents the related works done in this field. Finally, Section 6 allows for concluding remarks and future possibilities for the project.

2. CHALLENGES

In order to build a university virtual tour website that uses Google Street View images gathered from a web scraper, a few challenges have been identified as follows.

2.1. Challenge 1: The scraping script does not tell what the next place is

The scraping script generates a json file, which consists of two descriptions for each location. Description1 is a sentence that states the name of the current location, while description2 is a sentence that either tells the viewer what the next location is, or concludes the tour if it is the last stop. However, when the python script tries to find the available 360 pictures, it finds the locations near the targeted school one by one, which means that it cannot get the name of the next location and put it in "description2." In order to fix the problem, we implemented a function called "processing(inp)" (see Figure 1) for when the scraping part finishes running. The processing function will iterate through every location in the data dictionary, changing the "description2" of each location with the name of the next location before outputting the data as a json file.

```
# Add the descriptions
def processing(inp):
  data = inp
  for school in data["data"]:
    for i in range(len(school["buildings"])):
      if i+1 == len(school["buildings"]):
        school['buildings'][i]['description2'] = "You have reached the last stop of the tour. Hope you enjoyed. Have a wonderful day!"
      else:
        school['buildings'][i]['description2'] = "Your next stop is "+ school['buildings'][i+1]['title']  + '.'
  return data
```

Figure 1. "processing (inp)"

2.2. Challenge 2: The Scraping Script will Always Produce Duplicate Locations

Our scraping script is designed to find available 360 pictures around several locations in each school. However, because there might not be that many 360 pictures on Google Street View, every school will more or less have locations with the same pictures, depending on the school and its available pictures. If a school has abundant images (meaning that people in the past have taken a lot of 360 pictures and uploaded them), then it will have fewer places with the same images.

Obviously, users will not want to see the same image twice for a school, so in order to avoid this problem we implemented another processing function called "check_repeat(inp)" (see Figure 2). The function will iterate through every location with every school scraped, truncate the latitude and longitude to four decimal places (this is because although many locations show different latitude and longitudes in the later decimal places, they still represent the same image), and delete the location if the image already appears before the data gets outputted as a json file. Meanwhile, the function will also print out the name and the number of locations deleted in the terminal. Although this will not eradicate the problem entirely since sometimes the latitude and longitude will still show the same image, it will effectively solve the problem in 80% of the instances.

```
def check_repeat(inp):
  data = inp
  for school in data["data"]:
    li = []
    index_factor = 0
    for i in range(len(school["buildings"])-1):
      lat = float('%.4f'%school['buildings'][i-index_factor]['lat'])
      lng = float('%.4f'%school['buildings'][i-index_factor]['lng'])
      school['buildings'][i-index_factor]['lat'] = lat
      school['buildings'][i-index_factor]['lng'] = lng
      if {lat:lng} not in li:
        li.append({lat:lng})
      else:
        school['buildings'].pop(i-index_factor)
        index_factor += 1
    print(school['school_name'], index_factor)
  return data
```

Figure 2. "check_repeat (inp)"

2.3. Challenge 3: The "List Of Places" Menu Cannot Jump to the Correct Location

In the touring page, each school will have its own "List of Places" menu in the navigation bar that shows a list of all the places available on the site. This is intended for users to see all the locations available and jump to the one they want. However, since we did not make a new html page for every location, it is difficult to let them jump to the correct location of the correct school, especially when there are a lot of locations stored in the database. Therefore, we developed a JavaScript function, "goToNextPlaceWithIndex(gotoIndex)" (see Figure 3). This function will take the index of the location that appears on the List of Places menu, find the data stored with the index, and start a street view panorama with that location.

```
function goToNextPlaceWithIndex(gotoIndex) {
  if (index < allData[schoolIndex]["buildings"].length) {
    index = gotoIndex;
    $("#title").text(allData[schoolIndex]["buildings"][index].title);
    $("#description1").text(allData[schoolIndex]["buildings"][index].description1);
    $("#description2").text(allData[schoolIndex]["buildings"][index].description2);
    $("#heading").text('Virtourgo | ' + allData[schoolIndex]["buildings"][index].title);
    var panorama = new google.maps.StreetViewPanorama(
      document.getElementById('pano'), {
        position: {lat: allData[schoolIndex]["buildings"][index].lat, lng: allData
        [schoolIndex]["buildings"][index].lng},
        pov: {
          heading: allData[schoolIndex]["buildings"][index].heading,
          pitch: allData[schoolIndex]["buildings"][index].pitch,
        }
      });
    map.setStreetView(panorama);
  }
}
```

Figure 3. "goToNextPlaceWithIndex(gotoIndex)"

3. SOLUTION

Our finished product – Virtourgo – is a website that displays virtual tours for well-known universities in the world. It implements a web scraper to gather information from the internet and compile it into a virtual tour. The scraping process is done in the background, so individual users will not need to see or change anything to access the tour. The scraper uses Selenium in Python to search 1) buildings on the campus of the desired school, 2) available Google Street View pictures around each building, and 3) each picture's latitude and longitude. The information scraped is then re-evaluated by another script, which goes through each location to make sure they are not repeated. To conclude the scraping process, we stored the information scraped in a json file within a GitHub repository. Like any other website, users can access the Virtourgo by entering the URL, which in this case is virtourgo.com, in a web browser. The DNS Server we chose is Google Domains, and the server is GitHub. The users will automatically be directed to a page named "index.html," the main page of the website. On the homepage, there is a search box with auto-correct function. Users will enter the name of their desired school and click the "Start Virtual Tour" button to go to the tour page that shows the scraped locations one at a time. On the tour page, most of the screen will be the Google Street View images embedded on the page, and below those will be some descriptions about the place, as well as what the next stop will be. When the tour is concluded, users will be prompted to return to the home page, where they can feel free to start another tour. In sum, there are three main components of our system (see boxes in Figure 4):

- a Python Web Scraper that gathers information about all the locations
- a GitHub repository that acts as the server, hosting the website and storing the scraped information
- a DNS Server by Google Domains that processes the request

The following sections will describe the implementation process for each of the three components, as well as the website itself, in detail.

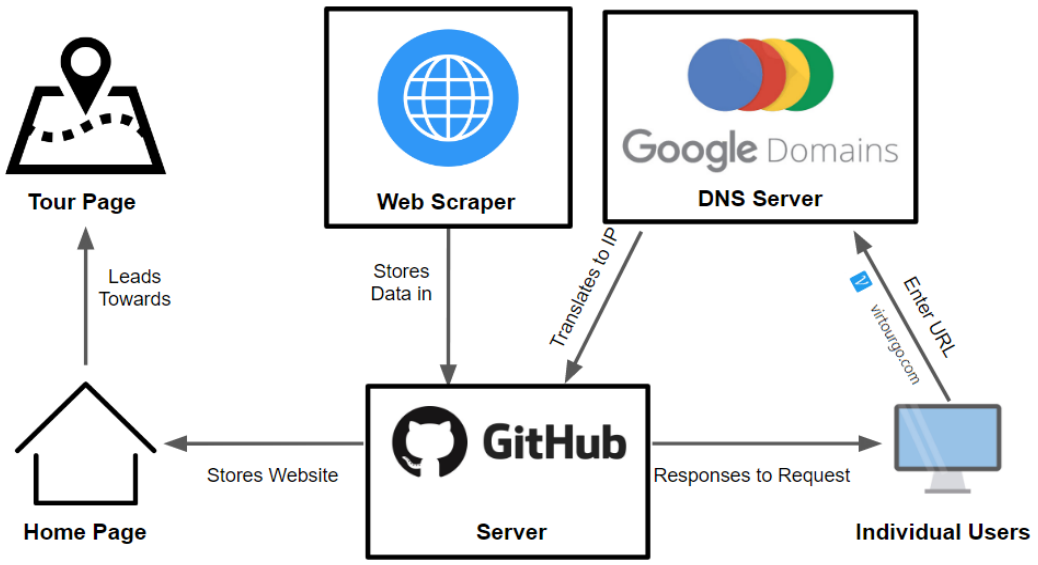

Figure 4. Three main components of our system

Web Scraper

For the Web Scraper part, our goal is to find the latitude and longitude of places of interest at and around the given school. To ensure efficiency, we would also like to scrape many schools at once, given a list of all the schools we want. First, we need to make sure that the name of the school we have is the same one that appears on Google Maps. We can do this by finding the place with the closest name to the one we inputted. This is to prevent unexpected writing errors when typing in or copying the name of the school. Then, we can start searching for the actual locations around the place we want. For each place, we will find the closest 60 locations around it; for each location, we will gather the name of the place and the latitude and longitude of the closest available image. The code to find places around each school is shown in Figure 5. To be able to search many schools at a time, we implemented another function, which simply iterates through a list of different schools to find the information above for each one. This way we don't have to run the code for each school. Finally, we will output the data we scraped into a json file, which we will use in the actual website. The json file, which we name as "data.json," is designed specifically for the website and has all of the same variable names. This allows us to simply copy the "data.json" file from the Python scraper directory and directly paste it in the website repository without needing to change the its formality.

```
def findPlaces(loc=(location["lat"],location["lng"]), pagetoken = None):
  lat, lng = loc
  url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json?location={lat},{lng}&rankby=distance&key={APIKEY}{pagetoken}
  ".format(lat = lat, lng = lng,APIKEY = APIKEY, pagetoken = "&pagetoken="+pagetoken if pagetoken else "")
  response = requests.get(url)
  res = json.loads(response.text)

  for result in res["results"]:
    building_name = result["name"]
    lat = result["geometry"]["location"]["lat"]
    lng = result["geometry"]["location"]["lng"]
    buildings.append(
      {
        "title": building_name ,
        "description1": "This is the " + building_name + '.',
        "description2": "",
        "lat": lat,
        "lng": lng,
        "heading": 34,
        "pitch": 10,
      })
  pagetoken = res.get("next_page_token",None)

  return pagetoken
```

Figure 5. Code to find places around each school

Of course, because not every location has its own 360 picture on Google Street View available, many locations will show the same picture, so we have to use another function to rule out locations with the same images. Because we chose to find the locations first, and then find 360 pictures according to the locations, duplicate pictures are unpreventable. An alternative scraping method would be to find the pictures first, then find the name of the building closest to the picture. This would eradicate duplicate pictures. However, Google Street View currently does not support searching surrounding buildings' names with a 360 picture. It only functions the other way around, that is, to search the closest 360 pictures of a building. Therefore, we have to deal with the duplicate pictures elsewhere with another function.

Google Domains DNS Server

The DNS Server used for the website is Google Domains, which hosts our domain. First, we entered the webpage to build a domain through "https://domains.google.com/registrar/search." Then, we entered the name that we wanted the domain to be: "virtourgo." Next, we chose the

ending of my domain, which is ".com" in our case. Finally, we needed to pay an annual fee of $12 for Google to host our domain. After we paid for the domain and logged in to the google account that bought it, we navigated to "My Domains," and selected the domain we just registered, and clicked on "DNS." There are a couple of things we need to put in before it starts working (see Figure 6).

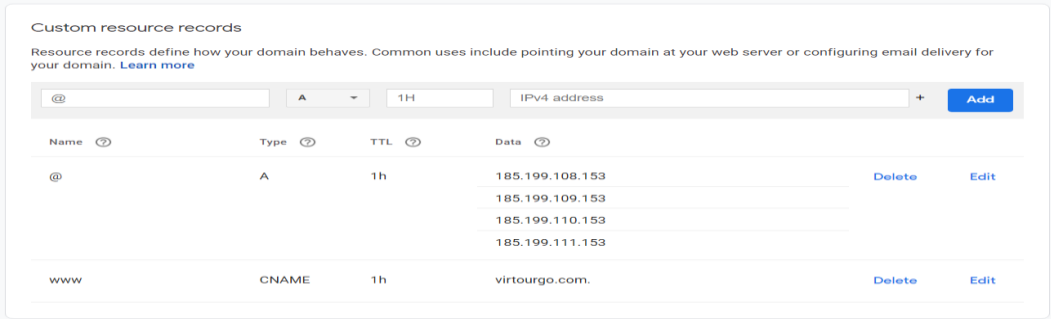

Figure 6. DNS records

In the box above, we entered "185.199.108.153" into the "IPv4 address" section, and then clicked the + button to the right of it. Next, we entered the rest three of the IPv4 addresses, "185.199.109.153," "185.199.110.153," and "185.199.111.153." These are the IP addresses that belong to GitHub. Then we clicked "Add" after finishing. Finally, in the dropdown menu that says "A," we clicked to find a button called "CNAME," and in the "Domain name" section, we put in the domain we just registered, "virtourgo.com."

GitHub Server

All the html programs of the website, as well as the scraped information, are stored in a GitHub repository, which also serves as the server of the website. To do this, we first created a GitHub repository, and pushed the html and json program files on the repository. Then, we went to the "settings" page of the repository, which can be found at the top of the repository menu, and found a section called "GitHub Pages" (see Figure 7). In this section, we entered the domain registered before in the "custom domain" part and clicked "save." In fact, users can host their websites on GitHub without owning their own domains, so if you don't mind the domain showing your username and "github.com," you could skip the "Google Domains and DNS Server" part. But because we want our website to show our custom domain, we needed to add this in the GitHub repository settings. Finally, we checked the box that says "enforce HTTPS," which stands for "Hypertext Transfer Protocol Secure." This gave our website secure communication by encrypting the data requests automatically.

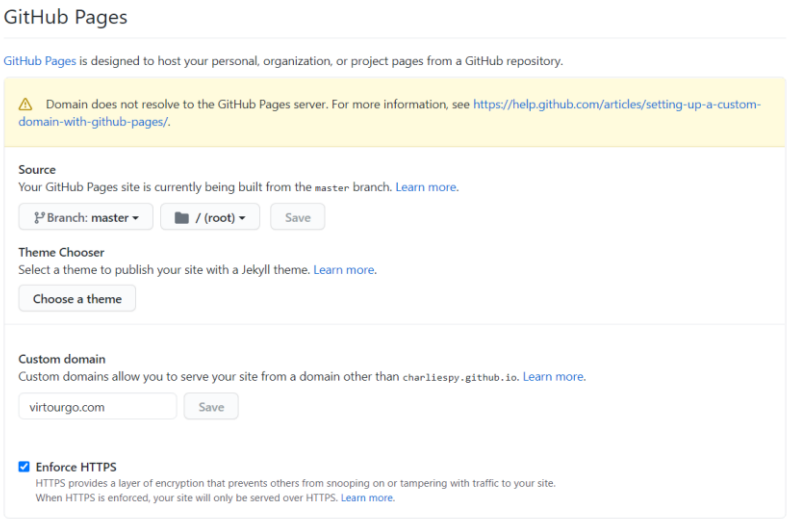

Figure 7. "GitHub Pages"

Website Development

There are three main technical parts of the website development part. The first one is the search box, the second one is the function to find the desired school, and the third one is the embedding of the Google Maps Street View window into our website. For the search box, we want to add an auto-complete function for it, so users will not need to enter the precise name that appears on our website. We implemented the autocomplete function using Jquery. We created a "#schoolInput" tag (see Figure 8) and used the tag when defining our search box by saying (id="schoolInput"). The source "schools" is a var that we created with all the names of our school list, and the delay is the time in milliseconds the search box will wait before giving its suggestions. For instance, in our case, users will see the search box automatically suggesting the name of the school according to what the user has already typed in after 600 milliseconds of idle time.

```
$( "#schoolInput" ).autocomplete({
  source: schools,
  delay: 600
});
```

Figure 8. Autocomplete function using Jquery

For our search box to be able to jump to the correct school, we created another function called "#startVirtualTour" (see Figure 9). In this function, we retrieve and call the name of the school from the previous function "#schoolInput," and find the index of that school's name. Then, we navigate to the index of that specific school with another function "goToNewPage," which leads us to the actual tour page.

```
$("#startVirtualTour").click(function(){
  var school = $("#schoolInput").val();
  var indexForTheSchool = school_mapping[school];
  dataToParse = allData[indexForTheSchool];

  function goToNewPage() {
      url = 'new_view.html?index=' + indexForTheSchool;
      document.location.href = url;
  }

  goToNewPage();
});
```

Figure 9. "#startVirtualTour"

For the Google Maps Street View function, there are a couple of stats we have to enter in order for the street view panorama to initiate (see Figure 10): the position, which includes the latitude and longitude of the location, and the pov, which includes the heading and the pitch. The latitude and longitude determine the exact position the place is located on the map, so that Google Street View can find it in its system. The heading signifies the position the users are looking at (e.g., whether north or south), and the pitch determines the angle of the pov (whether looking straight ahead or pointing at the sky). We then accessed the information scraped in the previous section in a function by id "pano," and entered those numbers. Notably, besides linking the JavaScript file in our html file as usual, we also inserted the following script with our google api in the html file (we replaced our actual API key with "your_api" for security reasons):
<script async defer src="https://maps.googleapis.com/maps/api/js?key=your_api&callback=initialize"></script>

```
var panorama = new google.maps.StreetViewPanorama(
  document.getElementById('pano'), {
    position: {lat: allData[schoolIndex]["buildings"][index].lat, lng: allData[schoolIndex]["buildings"][index].lng},
    pov: {
      heading: allData[schoolIndex]["buildings"][index].heading,
      pitch: allData[schoolIndex]["buildings"][index].pitch,
    }
  });
map.setStreetView(panorama);
```

Figure 10. Stats to enter for street view panorama

4. EXPERIMENT

To evaluate the major components described above, we designed three experiments to see if they work as expected. The first experiment aimed to test the autocomplete search box and the jump to school function. On the website, we typed in the first few letters of some school we scraped. If our targeted school appeared in the autocomplete list, we then clicked the "Start Virtual Tour" button to see if we are navigated to the correct school. We picked 20 schools randomly from the list, and if all the schools passed the aforementioned test, we considered the whole system to fulfill our expectations.

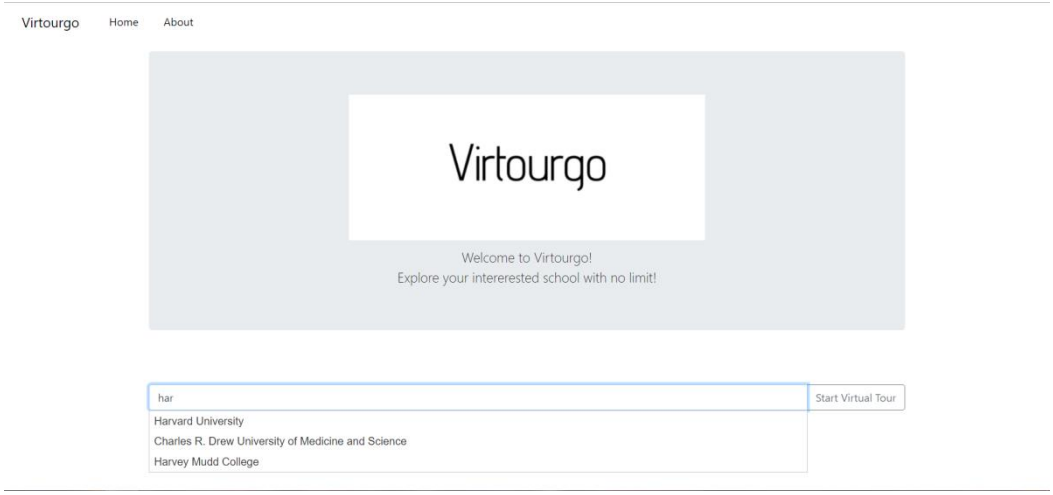

Figure 11. Typing fist three letters, "Har…"

We tried the test with 20 schools randomly selected from the school list, one of which is Harvard University. We typed in the first three letters of the name "Harvard" (see Figure 11), and the autocomplete function successfully provides us with all the schools whose names include "har." In our database now, there are three schools with "har" in its name, "Harvard University," "Charles R. Drew University of Medicine and Science," and "Harvey Mudd College." After we clicked and selected our targeted school Harvard University, we clicked the "Start Virtual Tour" button at the right of the search box. The page then changed to the main tour page (see Figure 12), which shows the first stop of Harvard University.

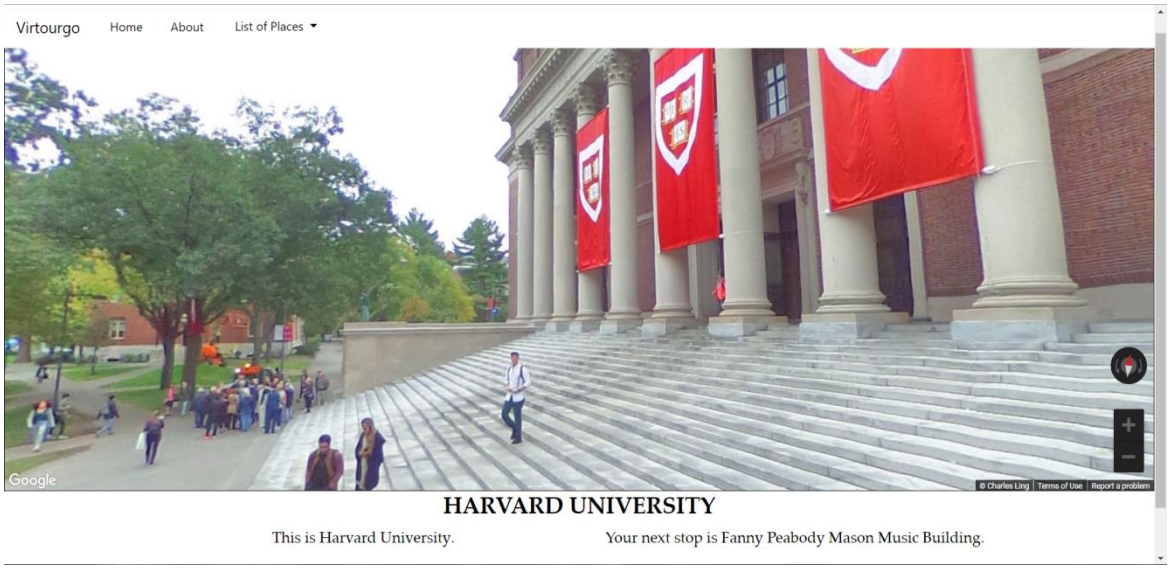

Figure 12. The main tour page for Harvard, which shows the first stop of Harvard University

We repeated the process for another 19 randomly selected schools and all got the results we expected - the autocomplete function showed us the school's name, and the start tour button led us to the tour page with the correct school's images.

Experiment 2

The second experiment was designed to test the scraping function. We randomly generated 20 schools and put them in a list named "schools." Then we called for the scraping function and the processing function (see Figure 13).

```
schools = [
    'Andrews University',
    'University of Maryland, College Park',
    'University of West Los Angeles',
    'California State University San Marcos',
    'Starr King School for Ministry',
    'University of Pittsburgh',
    'Claremont Graduate University',
    'Vanderbilt University',
    'University of Illinois, Chicago (UIC)',
    'Providence Christian College',
    'San Jose State University',
    'California State University, Los Angeles',
    'Golden Gate University',
    'Epic Bible College',
    'Fielding Graduate University',
    'Biola University',
    'University of Southern California',
    'Alliant International University',
    'Illinois Institute of Technology',
    'The Ohio State University'
]

# Add the descriptions
def processing(inp):
  data = inp
  for school in data["data"]:
    for i in range(len(school["buildings"])):
      if i+1 == len(school["buildings"]):
        school['buildings'][i]['description2'] = "You have reached the last stop of the tour. Hope you enjoyed. Have a wonderful day!"
      else:
        school['buildings'][i]['description2'] = "Your next stop is "+ school['buildings'][i+1]['title']  + '.'
  return data

result = generateSchoolList(schools)
final = processing(result)
with open('data.json', 'w') as outfile:
  json.dump(final, outfile)
```

Figure 13. Scraping and processing function

The processing function's purpose is to fix the problem with the school's names. During the scraping process, we could not get the name of the next location, so we cannot tell users where the next stop is. Therefore, the processing function aims to go into every location and change the part where it says what the next stop is, which is in a key called "description2." After the scraping is done, we then uploaded it into a json file called "data.json."

After the scraping is done (after each school's scraping process is completed, the terminal will print the index and the name of the school), we can open the output file "data.json" and see the results (see Figure 14).

data.json


```
{
  "timestamp": 1617508974.146392,
  "data": [
    {
      "school_name": "Andrews University",
      "address": "8975 Old 31, Berrien Springs, MI 49104, United States",
      "location": {
        "lat": 41.9646127,
        "lng": -86.35946609999999
      },
      "buildings": [
        {
          "title": "Andrews University",
          "description1": "This is the Andrews University.",
          "description2": "Your next stop is AUSA.",
          "lat": 41.9646127,
          "lng": -86.35946609999999,
          "heading": 34,
          "pitch": 10
        },
        {
          "title": "AUSA",
          "description1": "This is the AUSA.",
          "description2": "Your next stop is Andrews University Dining Services.",
          "lat": 41.96432169999999,
          "lng": -86.3600211,
          "heading": 34,
          "pitch": 10
```


Figure 14. Open output file "data.json"

The file is constructed with many dictionaries in json format. In the main dictionary, there is a timestamp, which represents the time the scraping action was performed, as well as a list called

"data." This list is compiled of 20 dictionaries that each point to their respective school. In each school dictionary, there includes a data point for the school's name, the school's address, the school's location in terms of latitude and longitude, and a list of buildings. In the list of buildings, there are 60 different dictionaries that each contain the information for a particular building. The information includes: the name of the building marked as "title," a sentence describing the name of this building called "description1," a sentence that tells the users where the next stop is stored in "description2," the latitude and longitude of the 360 picture, and the heading and pitch (also called "pov" in the html file of the website) of the location.

Experiment 3

Besides the processing function, we also wanted to test the function we designed to prevent duplicate locations. The third experiment is designed to test the effectiveness of the "check_repeat(inp)" function mentioned in the "Challenges" section.

First, we ran the scraping script of the "schools" list mentioned above, which consists of 20 randomly generated schools, without activating the "check_repeat(inp)" function; we named that json file "unprocessed." Then, we ran the scraping script, but this time with the function, and named its corresponding file "completed." Next, we went through each school in the school list of both the "unprocessed" and "completed" file with our website and recorded the number of duplicate locations in each file. Finally, we compiled the number of duplicate locations of each file into an excel table (see Figure 15).

	Total locations	Average duplicate locations	Percent duplicate
Unprocessed	60	31	51.60%
Completed	38	9	23.70%
		Percent change: -71.0% / -54.1%	

Figure 15. Number of duplicate locations in each file

In the "unprocessed" json file, each school had an average of 60 total locations, which corresponds with our expectations, because we scraped 60 locations for each school. In these 60 locations, 31 of them are duplicate results. In contrast, in the "completed" file, each school only had an average of 38 locations and 9 duplicate ones, showing a 71% decrease in the number of duplicate locations. Our check function successfully ruled out an average of 22 duplicate locations. Furthermore, in the "unprocessed" file, the rate at which duplicate locations occur is 51.6%, while in the "completed" file, the rate is only 23.7%. This shows a 54.1% decrease in the average duplicate rate of schools in the "completed" file. Although 23.7% of duplicate locations will still more or less affect users' experience, it is already much better than 51.6%. In the future, we hope to develop more advanced functions to check for repeated 360 pictures, and we hope to lower the duplicate rate to less than only 5%.

The experiment results show that the website brings up the correct school and the scraping script overall met our expectations.

5. RELATED WORK

Andri, C., et al. created a virtual tour for Management and Science University (MSU) with augmented reality and virtual reality in 2019 and recorded that the majority of the users were happy with the product [16]. Their work is more detailed and entails more time with each specific

information. Our work is a more general script that can cover the tours of many schools at the same time. Perdana, D., et al. organized a virtual tour with 360-degree pictures using 3DVista for 3 buildings of Telkom University (Tel-U) in Indonesia [17]. Their 360 pictures were made using stitched-together 2D images, while the source of our 360-degree pictures were scraped from Google Street View. Similar to the augmented 360-degree panoramic safety training done by Eiris, R. et al., their works are more interactive than ours, but require more precise planning of the tour route [18]. Li, X. et al. developed a program to use 360-degree panoramic pictures from Google Street View to map urban landscapes and quantify environmental features [19]. Both of our works use Google Street View as the source of 360 pictures, but instead of analyzing them, our project tries to develop a tour with the pictures, while they try to analyze the city. Thennakoon, M. et al. advanced a tour mobile application, using web scraping to compile information from Wikipedia for tourists [20]. Both our applications are web-based products, but their work requests information from Wikipedia, while ours is from Google Street View. Widiyaningtyas, T. et al. also aimed at creating a virtual tour for school campus with ORB image stitching [21]. Their work is made possible by image stitching, which involves taking several 2D pictures and combining them into one panoramic image, while ours does not require us to take pictures ourselves.

6. CONCLUSION AND FUTURE WORK

In this project, we successfully developed a python web scraper that scrapes images from Google Street View to compile a tour for a given list of universities. We also created and launched a website that makes use of the data scraped from a json file and can show the school the user wants based on a search using a search box.

The entire project can be split up into 3 or 4 parts: the scraping script, the GitHub server, the Google Domains DNS Server, and the HTML website program. We discussed the implementation process of all of them in Section 3. In section 4, we did a series of experiments and proved that each part can function and the website can work as planned and deliver a user's tour smoothly. The experiments also showed that we have solved the challenges effectively.
During the Covid-19 pandemic, many students are locked in their homes, unable to attend an actual university tour. The purpose of this project was to provide a solution to this problem by developing a virtual tour using 360-degree images. Through the experiments, we have proven that users can achieve this goal effectively on our website, and that the scraping script can create tours for many schools efficiently. In fact, the tour can not only be used for universities, but for other public buildings or gathering places. It is even possible to scrape other information, such as detailed descriptions, people's comments, or 2D images of each school.

Notably, the product of our research can be applied in many scenarios other than university tours. Because our web scraper has the ability to scrape 360 pictures of universities, it can also be used in other places such as football stadiums, public amenities, as well as other landmarks of the city, where there are a fair amount of pictures available on Google Street View, and users would like to know the surrounding environment of these places. In such case, our web scraper can be used directly for searching these new places with no change needed to the scraping script; we can also make a new website in a short amount of time, as we only have to change the homepage of the website while keeping the functions that power the main tour page.

However, there are still limitations to the system. For instance, although the function that checks repeats works in the majority of cases, it is still unable to detect every repeated location with just latitude and longitude data. This requires human effort to physically go through the scraping script and delete repeated locations. Also, the quality of the 360-degree images cannot be

ensured. Everybody can upload 360 pictures to the internet, and since Google Street View has been around for more than 10 years, some pictures were taken many years ago and therefore have lower image resolution and quality. Lastly, because the website uses Google Domains to power its functions, users who cannot access Google services, such as those in China, will not be able to view the website unless with the help of other proxy applications.

In the future, we can try to find another way to analyze and delete the repeated images, and also find ways to analyze the resolution of the images in order to maximize the tour quality. Improving the look of the website UI can also give users a better experience with the tour.

REFERENCES

[1] Anguelov, Dragomir, et al. "Google street view: Capturing the world at street level." Computer 43.6 (2010): 32-38.

[2] Rundle, Andrew G., et al. "Using Google Street View to audit neighborhood environments." American journal of preventive medicine 40.1 (2011): 94-100.

[3] Gorelick, Noel, et al. "Google Earth Engine: Planetary-scale geospatial analysis for everyone." Remote sensing of Environment 202 (2017): 18-27.

[4] Patterson, Todd C. "Google Earth as a (not just) geography education tool." Journal of Geography 106.4 (2007): 145-152.

[5] Mitchell, Ryan. Web scraping with Python: Collecting more data from the modern web. " O'Reilly Media, Inc.", 2018.

[6] Lawson, Richard. Web scraping with Python. Packt Publishing Ltd, 2015.

[7] Marrs, Tom. JSON at work: practical data integration for the web. " O'Reilly Media, Inc.", 2017.

[8] Dabbish, Laura, et al. "Social coding in GitHub: transparency and collaboration in an open software repository." Proceedings of the ACM 2012 conference on computer supported cooperative work. 2012.

[9] Aghayari, S., et al. "Geometric calibration of full spherical panoramic Ricoh-Theta camera." ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences IV-1/W1 (2017) 4 (2017): 237-245.

[10] Kavanagh, Sam, et al. "Creating 360 educational video: A case study." Proceedings of the 28th Australian conference on computer-human interaction. 2016.

[11] Napolitano, Rebecca K., George Scherer, and Branko Glisic. "Virtual tours and informational modeling for conservation of cultural heritage sites." Journal of Cultural Heritage 29 (2018): 123-129.

[12] Ashmore, Beth, and Jill E. Grogg. "Library virtual tours: A case study." Research Strategies 20.1-2 (2004): 77-88.

[13] Kabassi, Katerina, et al. "Evaluating museum virtual tours: the case study of Italy." Information 10.11 (2019): 351.

[14] Parsons, Thomas D., and Albert A. Rizzo. "Initial validation of a virtual environment for assessment of memory functioning: virtual reality cognitive performance assessment test." CyberPsychology & Behavior 11.1 (2008): 17-25.

[15] Burdea, Grigore, and Philippe Coiffet. "Virtual reality technology." (2003): 663-664.

[16] Andri, Chairil, Mohammed Hazim Alkawaz, and A. Bibo Sallow. "Adoption of mobile augmented reality as a campus tour application." Int. J. Eng. Technol. 7 (2018): 64-69.

[17] Perdana, Doan, Arif Indra Irawan, and Rendy Munadi. "Implementation of a web based campus virtual tour for introducing Telkom university building." International Journal if Simulation—Systems, Science & Technology 20.1 (2019): 1-6.

[18] Eiris, Ricardo, Masoud Gheisari, and Behzad Esmaeili. "PARS: Using augmented 360-degree panoramas of reality for construction safety training." International journal of environmental research and public health 15.11 (2018): 2452.

[19] Li, Xiaojiang, Carlo Ratti, and Ian Seiferling. "Mapping urban landscapes along streets using google street view." International cartographic conference. Springer, Cham, 2017.

[20] Thennakoon, M. S. B. W. T. M. P. S. B., et al. "TOURGURU: tour guide mobile application for tourists." 2019 International Conference on Advancements in Computing (ICAC). IEEE, 2019.
[21] T. Widiyaningtyas, D. D. Prasetya and A. P. Wibawa, "Web-based Campus Virtual Tour Application using ORB Image Stitching," 2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 2018, pp. 46-49, doi: 10.1109/EECSI.2018.8752709.
[22] Fry, D., Mooney, S.J., Rodríguez, D.A. et al. Assessing Google Street View Image Availability in Latin American Cities. J Urban Health 97, 552–560 (2020).
[23] Keralis, J.M., Javanmardi, M., Khanna, S. et al. Health and the built environment in United States cities: measuring associations using Google Street View-derived indicators of the built environment. BMC Public Health 20, 215 (2020).
[24] Li X. Examining the spatial distribution and temporal change of the green view index in New York City using Google Street View images and deep learning. Environment and Planning B: Urban Analytics and City Science. October 2020. doi:10.1177/2399808320962511