Project Deliverable G: Prototype 2 and Customer Feedback

Mariyam Sheikh, Amélie Chénier, Michael Braimah,
Margaret Kravchenko, Luke Meintjes

November 10, 2024

Introduction:

Proper documentation is crucial during the prototyping phase. This document will outline the second prototype of the smart glasses assistance website, covering a key subsystem and includes a model. It introduces the location subsystem It also includes documentation of the testing conducted on the prototype. Feedback collected from this testing will be analyzed and used to guide potential adjustments to the target specifications or the design of the website. Lastly, the document presents a test plan for the third prototype.

1.

This prototype was made to test how the location subsystem would work on the website. It allows us to receive feedback on this subsystem and improve it for the future product. The program allows the user to enter a location, a radius around that location and the type of place they are looking for (restaurant, park, school, …). The program then prints the different places which fit inside the radius and the description of the place. Along with the name of the place, it also prints the adress and the location of the place. To complete these functions, it uses a Google Places API. The user would use this setting on the website.

Part of this code was written with Chat GPT, where the prompt was "Write a code for a Google Places API key with get location", 09/11/2024.

```python
import requests
import json

# Replace with your own Google API Key (with access to both Places and Directions APIs)
API_KEY = 'AIzaSyAfSFTVoMOKW39Wsw_92Wr7jn2E-JhCZ3I'

# Define the endpoint for Places search (Nearby Search)
places_endpoint = "https://maps.googleapis.com/maps/api/place/nearbysearch/json"

# Define the endpoint for Place Details API
place_details_endpoint = "https://maps.googleapis.com/maps/api/place/details/json"

# Define the endpoint for Directions API
directions_endpoint = "https://maps.googleapis.com/maps/api/directions/json"

# Parameters for the API request (Nearby Search)
print("Enter a location: ")
location = input()
print("Enter a radius in meters: ")  # Example: Latitude, Longitude for San Francisco
radius = input() # Search within 200 meters
print("Enter the type of place you are looking for (restaurant, school, park...): ")
type_of_place = input()  # Type of place to search for (e.g., restaurant, park, etc.)

# Your starting point (could be your current location or a fixed point)
start_location = "location"  # San Francisco coordinates (use current location here if needed)

# Create the URL with parameters for Nearby Search
params = {
    'location': location,
    'radius': radius,
    'type': type_of_place,
    'key': API_KEY
}

# Make the request to the Google Places API for nearby places
response = requests.get(places_endpoint, params=params)

```

```python
# Check if the request was successful
if response.status_code == 200:
    data = response.json()

    # Check if the response contains 'results'
    if 'results' in data:
        print(f"Found {len(data['results'])} {type_of_place}s near {location}:\n")

        # For each place in the search results, get more details using the Place Details API
        for place in data['results']:
            name = place.get('name', 'No name available')
            address = place.get('vicinity', 'No address available')
            place_id = place.get('place_id', None)

            print(f"- {name}, Address: {address}")

            # Now, use the Place Details API to get detailed information
            if place_id:
                details_params = {
                    'place_id': place_id,
                    'key': API_KEY
                }

                details_response = requests.get(place_details_endpoint, params=details_params)

                if details_response.status_code == 200:
                    details_data = details_response.json()

                    # Check if the response contains 'result'
                    if 'result' in details_data:
                        result = details_data['result']
                        location = result.get('geometry', {}).get('location', {})
                        latitude = location.get('lat', 'N/A')
                        longitude = location.get('lng', 'N/A')

                        print(f"  -> Latitude: {latitude}, Longitude: {longitude}")
```

```python
                        # Get directions from the start location to this place
                        print(f"  -> Getting directions from {start_location} to {name}...\n")

                        # Now call the Directions API to get directions from start_location to the destination
                        directions_params = {
                            'origin': start_location,
                            'destination': f"{latitude},{longitude}",
                            'key': API_KEY,
                            'mode': 'walking'  # You can change this to 'walking', 'bicycling', or 'transit'
                        }

                        directions_response = requests.get(directions_endpoint, params=directions_params)

                        if directions_response.status_code == 200:
                            directions_data = directions_response.json()

                            # Check if the directions request was successful
                            if 'routes' in directions_data and len(directions_data['routes']) > 0:
                                route = directions_data['routes'][0]
                                legs = route.get('legs', [])

                                if len(legs) > 0:
                                    leg = legs[0]  # Assuming we're just getting the first route
                                    distance = leg['distance']['text']
                                    duration = leg['duration']['text']
                                    print(f"  -> Distance: {distance}")
                                    print(f"  -> Estimated travel time: {duration}\n")
                            else:
                                print("  -> No route found.")
                        else:
                            print(f"  -> Error getting directions: {directions_response.status_code}")
                    else:
                        print("  -> No detailed location information available.")
                else:
                    print(f"  -> Error retrieving details for {name}. Status Code: {details_response.status_code}")
            else:
                print("  -> No Place ID available.")
```

```
    else:
        print("No results found.")
else:
    print(f"Error: {response.status_code}, {response.text}")

```

Link to prototype:
GNG1103_Places_Location_Directions_API-Deliverable G.py

2.

For this subsystem, it is a focused type focusing on the location subset where the user can input the initial location and the radius in meters, and then the type of place they are looking for, such as a restaurant, park, or school.

For the key characteristic comprehensibility, the user, when asked the location, without looking at the code, it's hard to tell whether the location is the starting location that they're starting or whether it is the location they are looking for. What places could be defined as a restaurant, and what type of school (elementary, secondary, university). The parameters need to be defined a bit more.

With the outputs being latitude, amount of distance that will need to be traveled, and how long it will take there. A lot of functions are defined, such as the mode of transport like bicycling, walking, and transit. Which makes this program very integrateable to the camera subsystem and then the audio subsystem to notify the user potentially. More clear boundaries are needed in order to understand the user's limits with this API.

Terminal that the user entered

45.2529,-75.4142

Enter a radius in meters:

3000

Enter the type of place you are looking for (restaurant, school, park,...):

Park

Found 1 parks near 45.2529,-75.4142:

- Kenmore Bicentennial Park, Address: Ottawa

  -> Latitude: 45.2333186, Longitude: -75.4163177

-> Getting directions from location to Kenmore Bicentennial Park...

-> No route found.

Functionality of the prototype:

We need to specify to the user whether this is a case-sensitive case and how they should input the location through the postal code or through the common name of the location.

For the type of place, the user could be potentially confused on whether it is a specific park or just a general location to input, like restaurants.

Prone to error for the user, as it is hard to switch back and fix it if the location was not desired or if there was a spelling error in one of the parameters.

We need to change the code so it allows the user to enter a location in text instead of in coordinates since it is a very unconventional method.

3.

For this prototype the same principles apply to the testing of it, we will only test for the functional requirements and not the constraints and functional requirements since it isn't physical glasses that are being tested but instead the functions of it. So again we will apply the same tests as the previous prototype:

- Functionality - Test to see if prototype works
- Adaptability - Test to see if user can adapt to when the prototype changes
- Number of uses - Test to see how many clicks it takes to make a change
- Number of features - Check the amount of options are available for change
- Easy to use and understand - Survey users of prototype

Test:

Test type	Functionality	Adaptability	Number of uses	Number features	Easy to understand and use
Test results	Yes	Yes	4	3	No

Analysis:

For the first test the functionality of the prototype does technically function but the only problem is that the inputs need to be extremely specific for the API to function correctly. For the next test it is extremely adaptable with the code being able to adapt instantly and changes to make for the location data as well. For the number of uses there were four input values that needed to be put in to change it. Next for the number features there are

three: The location inputs, the verification of search results and the directions. Lastly is the ease of use and understandability, for this prototype specifically it isn't very user friendly with it not being intuitive for the average user on how it works.

4.

Mechanical Engineering Student:

- The prototype combines multiple Google APIs- places, place details and directions- making it robust for various location-based queries. This enables the user to search for places and obtain detailed directions and travel estimates.

Computer Engineering student:

- "The clear prompts for location, radius, and type of place make the input process straightforward. I like that users can specify the type of establishment they're looking for (e.g. restaurant, park), which aligns well with user needs."
- "I like that the prototype provides users with comprehensive details, such as latitude, longitude, distance, and estimated travel time. This makes it a valuable tool for location-based assistance and planning."

Software Engineering Student:

- "The functions and parameters seem like they can be easily extended, making it adaptable for future integration with different subsystems too!"

Computer Science student:

- The code includes error handling for API responses, ensuring that users get feedback if something goes wrong, which is crucial for user experience in a real-world application.

Mechanical Engineering Student:

- "The prototype is well-structured, providing a strong foundation to add improvements in clarity and usability, such as predefined suggestions for location format or input validation to prevent errors."

5.

No updates needed.

6.

Prototyping Test Plan

Testing Functionality

Method: Testing the code with a lot of inputs.
Duration: 2 days
Timing: These tests will be conducted upon completion of the prototype.

Testing Adaptability:
Method: Testing the code with a wide range of inputs.
Duration: 1 day
Timing: Upon Completion of the prototype.

Testing Understandability and Difficulty of Use:
Method: Survey
Duration: 1 day
Timing: Upon completion of the prototype.

Conclusion:

During the prototyping phase of the Smart Glasses Assistance App, we gave customers an opportunity to explore our ideas and provide valuable feedback. Although this feedback didn't change the target specifications or the app's design, it remains important for future prototypes.

Trello Update: