Prototype III and Customer Feedback

E Cubed Group (P2)

Deliverable H	Rares Constantin Serban	Emma Belal
November 24th	Louis-Philippe Keith	

Abstract

This deliverable details our final prototype and the updates made to it from the previous two. The prototype test results are positive and in line with our target specifications. We have received our final set of comments and feedback from our fellow peers and are considering them for final touches and future advancements in the product.

Table Contents

Introduction

This deliverable will detail our final prototype, prototype three. Prototype three is a comprehensive high-fidelity model. We will detail an overall map of prototype three and its features. The features that are implemented here are a user interface (UI), facial recognition processing, user registration and update, access logs, location tracking and notification transmission. From our test plan we have gathered results which will emphasize the completeness of our product. In addition to prototype testing, we have also gathered user feedback. This feedback will help us refine the final product and prepare for Design Day.

Prototype Three

Prototype three is a consequence of prototype one and prototype two. It is the closet version we have to a fully-fledged product. The focus was to develop a UI, implementing location tracking, and integrating one Shabodi API into the product. The only API we can implement is bandwidth; we would like to implement latency, jitter, and location from them. Since we are unable to use those three APIs, we found an alternative location to use in our application. This is so we can fully develop our product and have the best version possible to demonstrate on design day.

Location tracking has been implemented in our product, via phone interaction with our Flask Server. This is an alternative to the Shabodi location tracking API. Location tracking improves our application since it adds additional information for the administrator. Location tracking was also a requirement we had gathered from the first client meeting.

The most significant update we have made is adding a UI. The development of an interface takes our product from being purely backend to a functional utilitarian product. The UI allows people without coding experience to be able to use the product. This allows enterprises to integrate web applications into their companies without having to train their staff in python coding. Additionally, a web application saves companies from having to download and update software frequently. Currently, our web application is private for testing purposes.

The implementation of Shabodi's APIs is also a requirement from the first client meeting. Since the only API available currently is bandwidth. We have chosen to implement it. This implementation is to demonstrate to the client that we can use their product. Ideally, we would like to implement more practical APIs like jitter and latency. These would improve the performance of our product and make it more seamless for the client.

User Interface

The user interface is the integration of our two previous prototypes. This allows the administrator to use the product without any knowledge of coding. The addition of a UI also fulfils the client's need to have an application. Our interface is a web application, we chose this to avoid having system integration errors and increase the flexibility of the product.

Features of the interface include:

- Login page for both administrators and employees
- Administrator’s employee registration form
- Administrator's employee location request page
- Log files of all access attempts for the week
- Door access pin prompt
- Facial recognition verification system

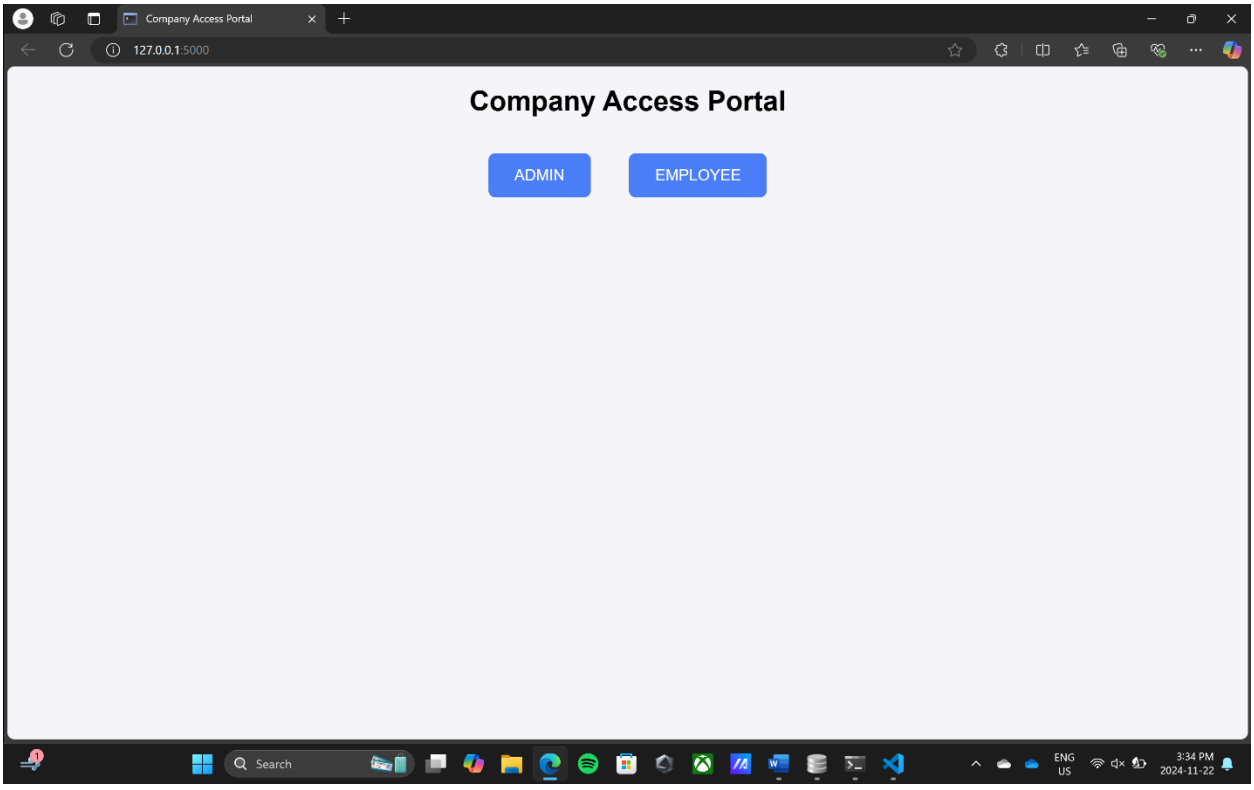

Figure 1: Home Page of UI

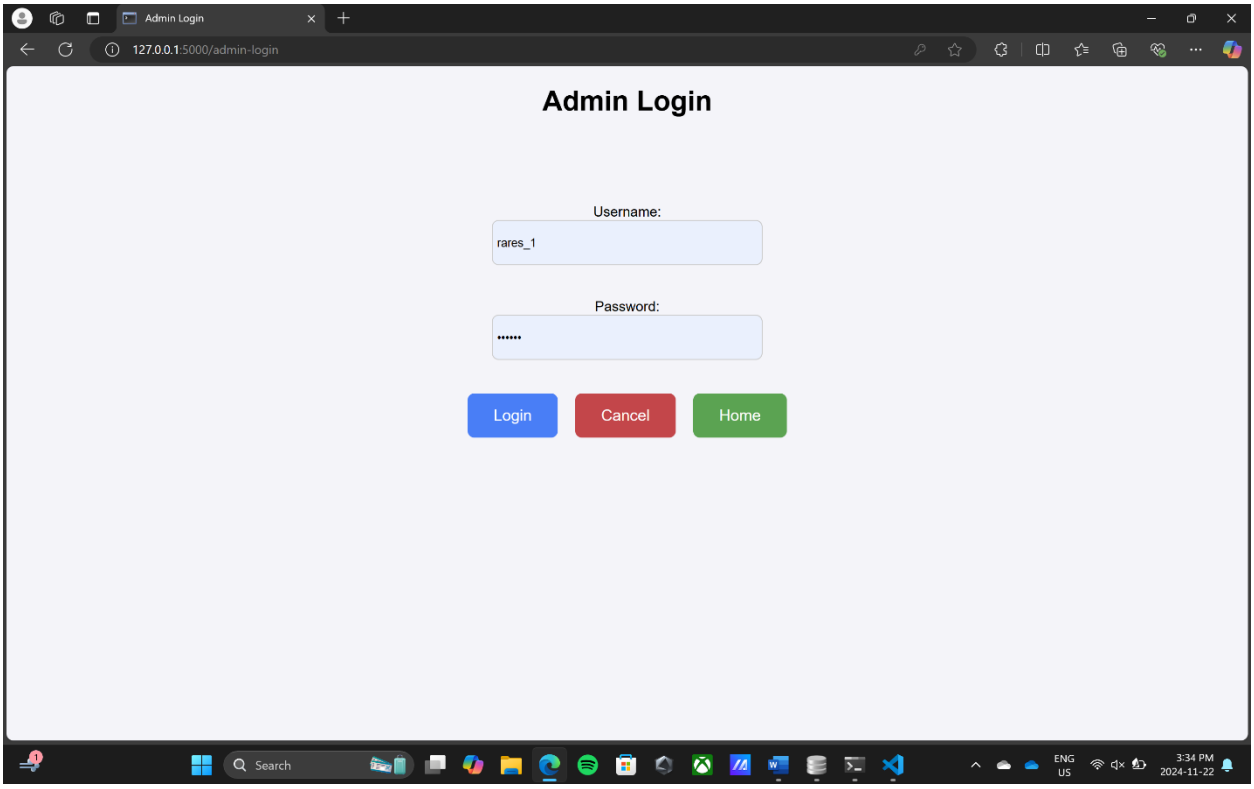

Figure 2: Administration Login in Page

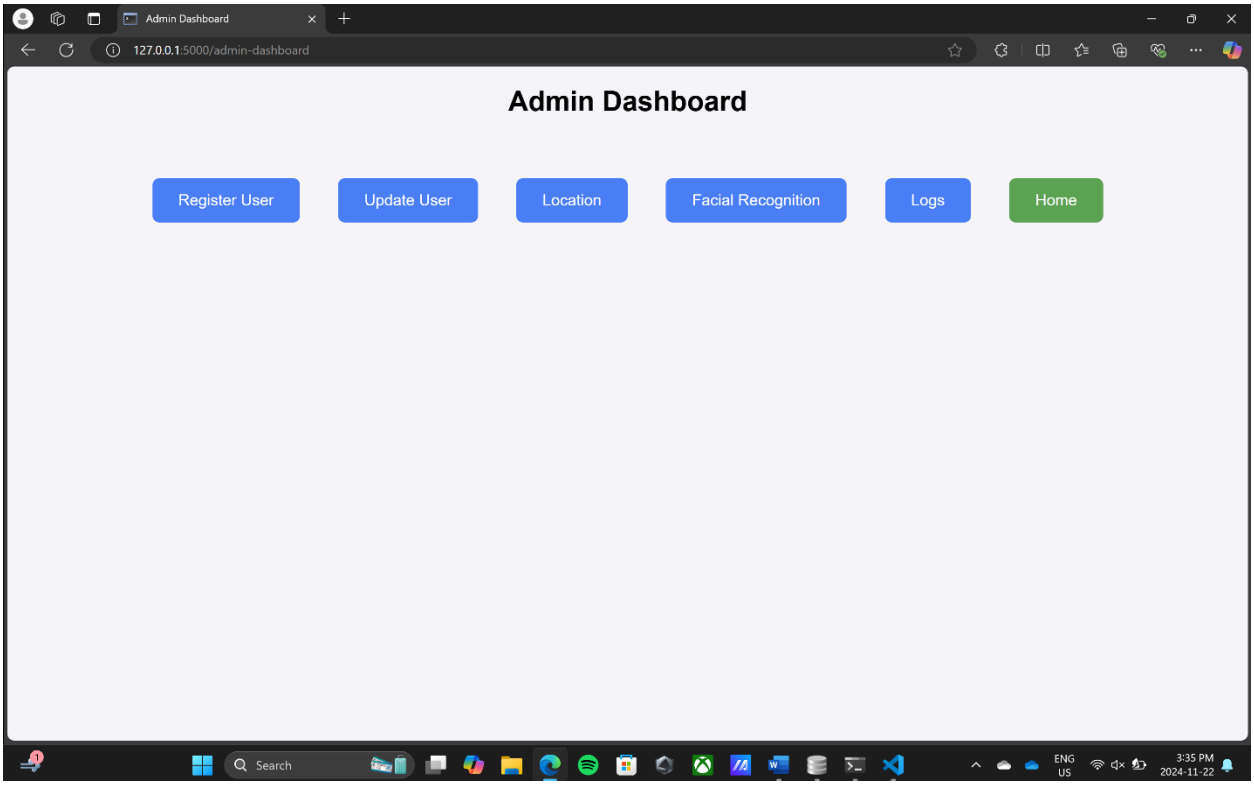

Figure 3: Administration Dashboard

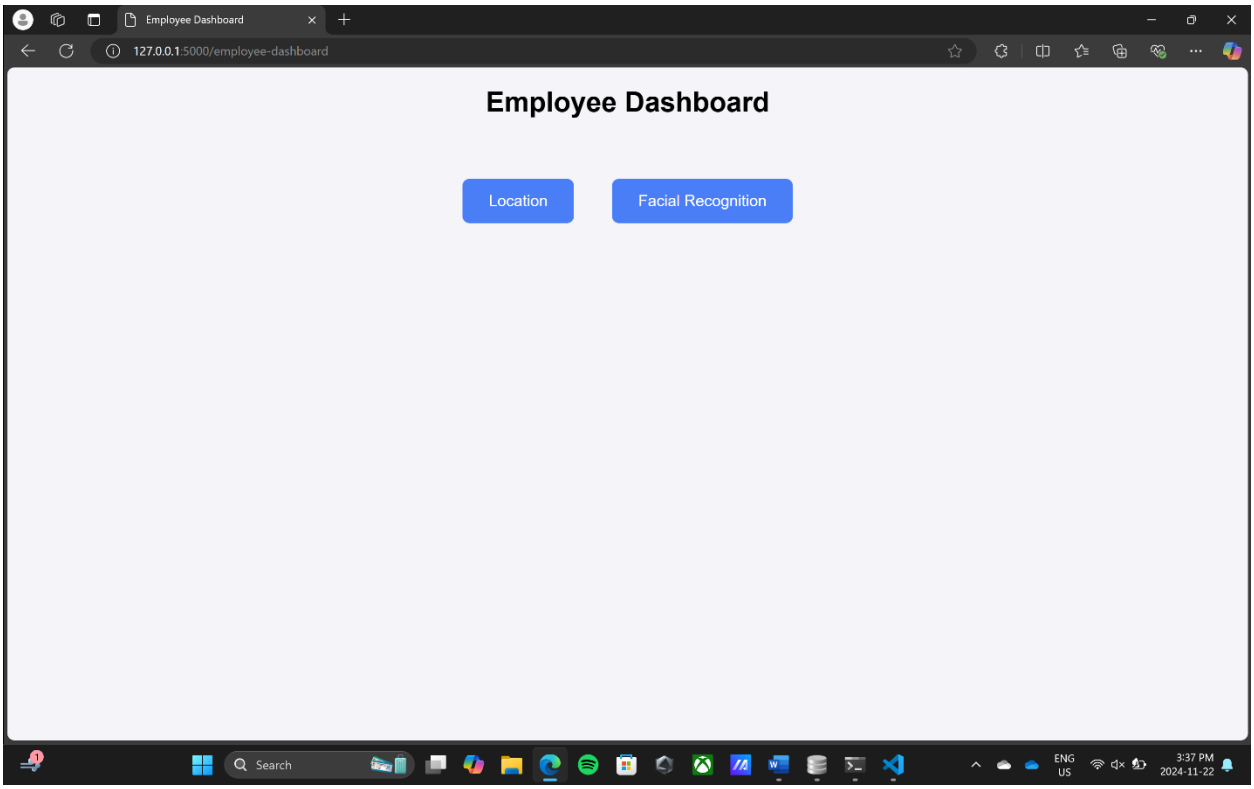

Figure 4: Employee Dashboard

Employee Registration

Employee Registration is another update from our previous prototypes. This allows the administrator to register and unregister employees. This is an easier system compared to what we had previously. Originally the administrator would need to enter the user's information manually into a directory file that was in python. This would not be practical, or time efficient. It would also require the staff to be trained in python coding.

In addition to this the administration can update the employee information from a specified portal. This makes it easier than having to unregister and register employees when information such as email changes.

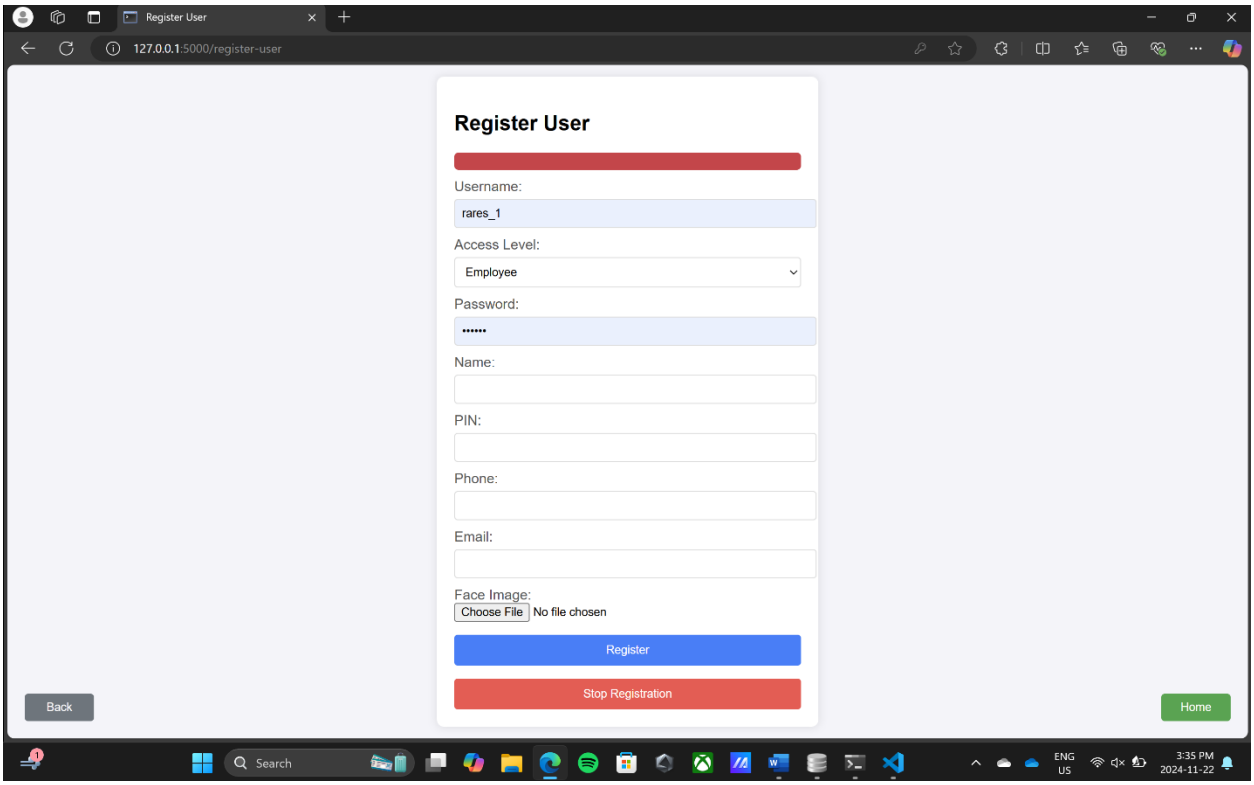

Figure 5: Registration Interface with example username and password

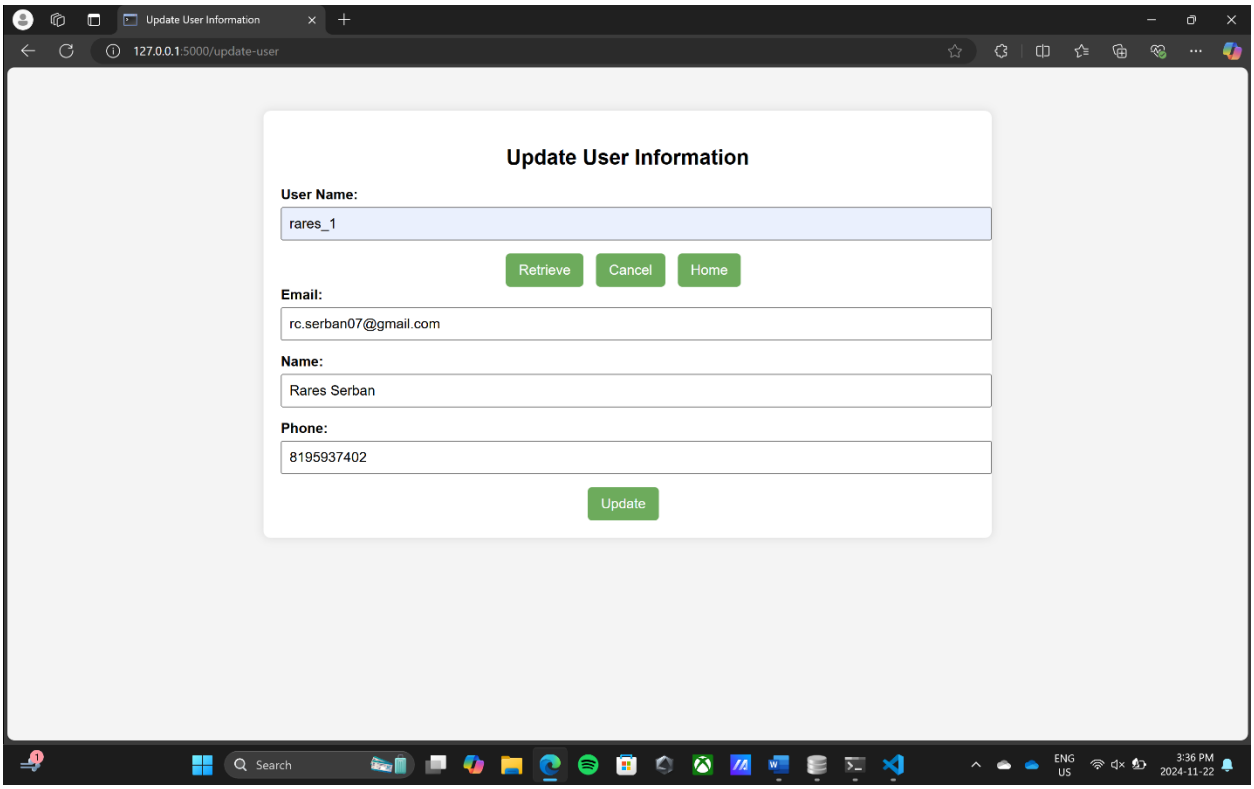

Figure 6: Updating User Interface with example information

Location Tracking

Location tracking is established by a wireless connection between our iPhones and the Flask Server. This method allows for high accuracy location retrieval regardless of any external factors (e.g. cell towers).

The following steps show how to set the system up:

1. Open the Shortcuts application on your iPhone.
2. On the top right corner, press the “+” button to create a new Shortcut.
3. In the Search Actions toolbar, type “Get current location” and click on it.
4. Click again on “Search Actions” and look for “Get Details of Locations”.
5. From the newly added rectangle box, click on the “Detail” case and under “Variables”, select “Latitude”.
6. Repeat step 5 but select “Longitude” instead of “Latitude”. If the second box contains anything other than “current location”, press on it and change it accordingly.

7. Click again on “Search Actions” and find “Get Device Details”. Make sure that the action gets the “Device Name” (your device identifier).
8. Click again on “Search Actions” and find “Get Contents of URL”.
9. Clear all variables in the box and enter *http://<your IP address>/<port>/update_location*. If you don’t know your local IP address, open your command prompt and type “ipconfig”. The information on the left of the IPv4 field is your local IP address. Typically, your port is 5000.
10. Next to the entered information, click on the little arrow. Change the Method to “POST”. Make sure the Request Body is “JSON”.
11. Click on “Add new Field”. The first two fields are “Number”. Instead of “Key”, type “lat” for the first one and “lon” for the second. Instead of the inscribed number, select the blue “Latitude” icon for the first one and “Longitude” for the second. For the third, select “Text”, name it “device_id” and select the blue “Device Name” icon.
12. Optional: Under “Search Actions”, look for “Show Notification”, and write the notification you wish to see when the Shortcut is successfully executed.

Now you must set up your Flask server. N.B. This procedure includes two categories of users (admin and employee):

1. Install SQLite database on your desktop.
2. Create two separate Python files. You can name the first “admin_locations.py” and the second one “employee_location.py”.
3. On “admin_locations”, you can copy the code you find in the annex. With this, you will create an SQLite database with 5 fields: “id”, “device_id”, “lat”, “lon” and “timestamp”. After running the code and accessing the IP address given to you in the terminal you should see a page saying, “Waiting for location data”.
4. Run the Shortcut on your iPhone and reload your page. You should now see an interpreted map (by Folium) and a pin indicating your current location.
5. If you run multiple Shortcuts all directed to the same IP address, you should see all the locations of all the devices.
6. On “employee_location”, you can copy the second code you find in the annex. This code doesn’t involve the database.
7. Execute step 4 again. You should now see your location on the map.

Additional information:

- To automate the Shortcut process, you can set up an automation action from the automation tab in your application. For example, you can say that when you get a message from "_person_" execute the shortcut. This way, an admin can get the location of all the employees without requesting their permission or any other action from their side.
- To avoid entering the IP address in the shortcut every time you change Network connection, you can create a tunnel for your specific port. This way, you can send the location from anywhere if you have some kind of Network connection. If you want to do that, follow these steps:
    1) Install "ngrok"
    2) Go to you command prompt and type "ngrok http <your port number>"
    3) If you're on the free trial version, don't close the command prompt and note that you can only create a tunnel for one port at a time.
    4) Next to the "Forwarding" field, you should see something resembling this: https://a7c5-137-122-64-245.ngrok-free.app
    5) Type this address instead of the IP address you currently have in the Shortcut, while keeping the route ("update_location"). It should look like this: https://a7c5-137-122-64-245.ngrok-free.app/update_location
    6) Now you should be able to send your location from anywhere you currently are, if you're connected to the internet.
- If you want to run these codes together in parallel with your main application, at the same time, you must run the files on different ports. For example, run the main application on port 5000, employee_location on port 5001 and admin_locations on 5002. Don't forget to change the information in the Shortcut as well if you need (if you don't use the tunnel)!

```
#admin_locations code example


from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
import folium
from datetime import datetime
import bandwidth
```

```python
app = Flask(__name__)

# Configure SQLite database
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///locations.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db = SQLAlchemy(app)

# Define the DeviceLocation model for storing device data
class DeviceLocation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    device_id = db.Column(db.String(50), unique=True, nullable=False)  # Unique
device identifier
    lat = db.Column(db.Float, nullable=False)  # Latitude
    lon = db.Column(db.Float, nullable=False)  # Longitude
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)  # Last updated
timestamp

# Initialize the database
with app.app_context():
    db.create_all()

@app.route("/update_location", methods=["POST"])
def update_location():
    data = request.json
    print("Raw request data:", request.data)
    print("Parsed JSON data:", data)

    if not data:
        print("No JSON data received.")
        return {"status": "error", "message": "No JSON data received."}, 400

    device_id = int(data.get("device_id"))

    api_token = bandwidth.get_access_token()
    bandwidth.invocation(api_token, device_id)

    lat = data.get("lat")
    lon = data.get("lon")

    if not all([device_id, lat, lon]):
        missing = [k for k in ["device_id", "lat", "lon"] if not data.get(k)]
        print(f"Missing fields: {missing}")
        return {"status": "error", "message": f"Missing fields: {',
'.join(missing)}"}, 400
```

```
    try:
        # Check if the device already exists
        existing_device =
DeviceLocation.query.filter_by(device_id=device_id).first()
        if existing_device:
            # Update the existing device's location
            print(f"Updating location for device_id: {device_id}")
            existing_device.lat = float(lat)
            existing_device.lon = float(lon)
            existing_device.timestamp = datetime.utcnow()
        else:
            # Add a new device location
            print(f"Inserting new device_id: {device_id}")
            new_device = DeviceLocation(
                device_id=device_id, lat=float(lat), lon=float(lon)
            )
            db.session.add(new_device)
        db.session.commit()
    except Exception as e:
        print(f"Error inserting/updating data: {e}")
        return {"status": "error", "message": "Failed to insert or update
data."}, 500

    return {"status": "success", "message": "Location updated."}, 200

@app.route("/")
def index():
    """
    Display a map with all device locations.
    """
    devices = DeviceLocation.query.all()
    if not devices:
        return "No device locations available."

    # Calculate the map's center based on all devices' locations
    avg_lat = sum([device.lat for device in devices]) / len(devices)
    avg_lon = sum([device.lon for device in devices]) / len(devices)
    map_object = folium.Map(location=[avg_lat, avg_lon], zoom_start=10)

    # Add markers for all devices
    for device in devices:
        tooltip = f"Device: {device.device_id}\nLast Updated:
{device.timestamp.strftime('%Y-%m-%d %H:%M:%S')}"
        folium.Marker(
```

```
            [device.lat, device.lon],
            tooltip=tooltip,
            icon=folium.Icon(color="blue", icon="info-sign")
        ).add_to(map_object)

    # Render the map in an HTML template
    html_map = map_object._repr_html_()
    return render_template('device_locations.html', html_map=html_map)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

```
#employee_location


import folium
from flask import Flask, render_template, request

import bandwidth

app = Flask(__name__)

# Store the latest location (default: None)
latest_location = {"lat": None, "lon": None}

@app.route("/update_location", methods=["POST"])
def update_location():
    """
    Receive real-time GPS data (latitude and longitude) from the iPhone.
    """
    global latest_location
    data = request.json

    device_id = int(data.get("device_id"))

    api_token = bandwidth.get_access_token()
```

```
    bandwidth.invocation(api_token, device_id)

    if "lat" in data and "lon" in data:
        latest_location["lat"] = data["lat"]
        latest_location["lon"] = data["lon"]
        return {"status": "success"}, 200
    return {"status": "error", "message": "Invalid data"}, 400

@app.route("/")
def index():
    """
    Display the latest location on an interactive map.
    """
    if not latest_location["lat"] or not latest_location["lon"]:
        return "Waiting for location data..."

    # Create a map centered around the latest location
    map_object = folium.Map(location=[latest_location["lat"],
latest_location["lon"]], zoom_start=15)
    folium.Marker(
        [latest_location["lat"], latest_location["lon"]],
        tooltip="Current Location",
        icon=folium.Icon(color="green", icon="info-sign")
    ).add_to(map_object)

    # Render the map in an HTML template
    html_map = map_object._repr_html_()
    return render_template('device_locations.html', html_map=html_map)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5001, debug=True)
```

Integration of Shabodi API

Unlike the two previous prototypes we were able to implement the bandwidth. Bandwidth was chosen since it was the only API we got to operate. Bandwidth is not being directly used in our prototype. The purpose of bandwidth is to give the companies the ability to restrict their cameras to specific areas. This will help with performance, considering not all cameras need to operate at once. Ideally, we would like to add latency and jitter APIs. These two APIs would help with our prototype performance.

```python
import http.client
import json

def get_access_token():
    conn = http.client.HTTPConnection("192.168.3.18", 31002)
    payload = json.dumps({
        "client_id": "0b9972c1-a76f-4c44-9ee4-9f6990811434",
        "client_secret": "g1InNk1-kQRueYHj3DmvNALQK8pGw_U_dlpL38A6Rqc"
    })
    headers = {
        'Content-Type': 'application/json'
    }
    conn.request("POST", "/security/v1/token", payload, headers)
    res = conn.getresponse()
    data = res.read()
    token_data = json.loads(data.decode("utf-8"))
    access_token = token_data.get("access_token")
    conn.close()
    return access_token

def invocation(access_token, device_id):
    conn = http.client.HTTPConnection("192.168.3.18", 7999)
    payload = json.dumps({
        "device": {
            "deviceId": device_id
        },
        "maxBitRate": 400,
        "direction": "uplink",
        "duration": 10000
    })
    headers = {
        'Content-type': 'application/json',
        'Authorization': f'Bearer {access_token}'
    }
    conn.request("POST", "/qos/v1/bandwidth", payload, headers)
    res = conn.getresponse()
    data = res.read()
    print(data.decode("utf-8"))
    conn.close()

# Main execution
if __name__ == '__main__':
    access_token = get_access_token()
    if access_token:
        invocation(access_token)
```

```
    else:
        print("Failed to retrieve access token.")
```

Prototyping Test Plan

Table 1: Remaining Schedule until Design Day

Week November 18th	Week November 25th
Finish the coding of the final product and include at least 1 API. Final product (code) must work consistently well	Finish the PowerPoint presentation and get prepared for the pitch. Code is ready to demonstrate to everyone.

Prototyping Results

We tested two pervious systems to ensure that they are still functioning as expected post UI creation. The facial recognition system and notification were tested in parallel by running multiple trials. A person would present their face after entering the pin information for the door. If the input information and user's face would match the door would 'unlock' and the admin would not receive a notification. This would count as a correct event. Similarly, if the information did not match and the administration received a notification it would also be a correct event. On occasion the facial recognition system would take longer than 5 seconds to process, which we considered a failure.

For location accuracy we did a trial over a period of time and locations. The administrator would request to have the location of a registered employee's phone number. We would receive the coordinates and measure how far they were from the phone's location. The average test results were below 20 m^2.

Testing the registration system, we added 2 people to the directory to see if the information could be saved. To test if this was correct, we had the users enter their pins into the 'door' and then see if they were granted access after the facial recognition system started. This worked multiple times for the 2 employees.

Testing the ability to update user information we had the two registered employees enter in a different phone number or email to see if they were still able to receive 'access granted' notifications. This worked on the 5 trials we did.

Table 2: Testing Results for Prototype 3

Subsystem	**Accuracy**	**Number of attempts**
Facial recognition	95%	20
Location tracking	Accurate to 20 $m^2 \approx$ 4-5 m radius	20
Notification	100%	20
Log-In System	100%	20
Registration - New	100%	2
Registration - Update	100%	5

User and Client Feedback

We asked family members and other people in our lab section. They have all motioned that our prototype seems very easy to use, and it is very utilitarian. The most common complaint about our software is that there is no “help” button to show new users how to use our software. Since our team will be there to explain how to use the software it does not matter if we don't have a help button since we will be there to teach them. We will also be creating a user manual for whomever has questions. The following table shows the comments we collected.

Table 3: Comments and Feedback

Person commented	**Positive feedback**	**Things needed improvement**
Classmate 1	I really appreciate how clear and easy to use it is.	I would like to see the GPS locations moving in real time. I would also like the help button.
Classmate 2	I really enjoy seeing the different use cases. I never thought of making a login for users and administrator.	If I used it alone it would be hard to understand

Conclusion

To conclude, prototype three passed all our tests with great success and it resembles and functions a lot like our final product. Prototype three was able to implement location, Face ID, and notification, all in a user-friendly web application. We have received many positive comments about our 3rd prototype and feedback we could address easily. For our final product we will try our best to implement one more of Shabodi's APIs, as well as make the whole application more presentable.

References

“Update HTML file with app routes” prompt. ChatGPT, OpenAI, 23 Nov. 2024, chat.openai.com/chat.