ECE 569A: Water Tap System

By:

Payton Murdoch, V00904677, pmurdoch@uvic.ca

Peter Sun, V00847800, peiyuan@uvic.ca

Yunfei (Alex) Tian, V00923409, yunfeitian@uvic.ca

Heebatullah, V00990440, hbeg@uvic.ca

Oche Eko, V01033252, ocheeko@uvic.ca

Project Background

In October of 2023, approximately 73% of Canada was declared to be in drought-like conditions [1]. With such drought conditions becoming increasingly common as we are beginning to experience more severe side effects of global warming, with this in mind it is important to note that according to the UN limited water resources have affected approximately four billion people. As such it is essential for the average consumer to track water usage habits within their household [2]. As such implementing a system with embedded sensors which can detect when water is running and inform the user is immensely beneficial. Furthermore, this system can inform users about drought-like and water-shortage conditions as well. While implementing the actual sensors is costly and rather difficult to configure, for this proposal, we can simulate the system to show investors the feasibility of this IoT device.

Technology and Market Analysis

Canada in general is within the top ten countries concerning freshwater resources [3]. As such preserving this resource is essential for sustaining Canadians for future generations. Given recent advancements in sensor technology water flow sensors can accurately depict even the rate at which water is flowing from a pipe or faucet. Coupled with infrared sensors to determine water flowing out of the tap system we can accurately measure the rate of water being lost [2]. Given these facts, water monitoring can be easily employed in water tap systems by companies like BC Hydro using incentives in an attempt to convince users to reduce water consumption. Otherwise, the average environmentally conscientious consumer can employ these devices themselves in an attempt to watch their habits.

System Design and Architecture

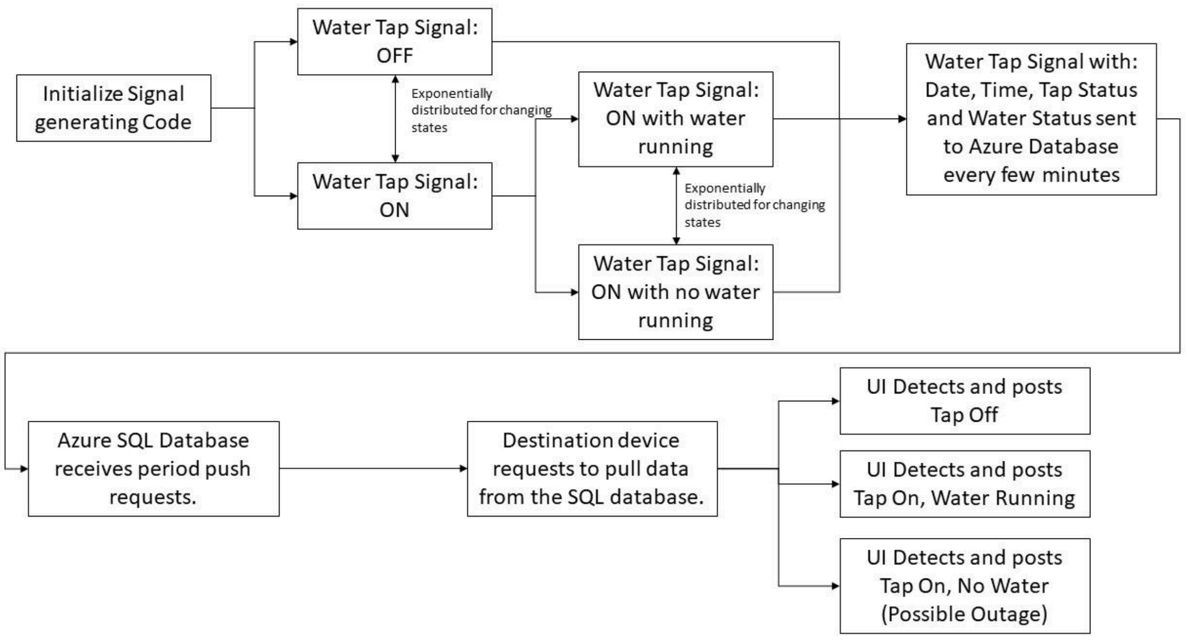
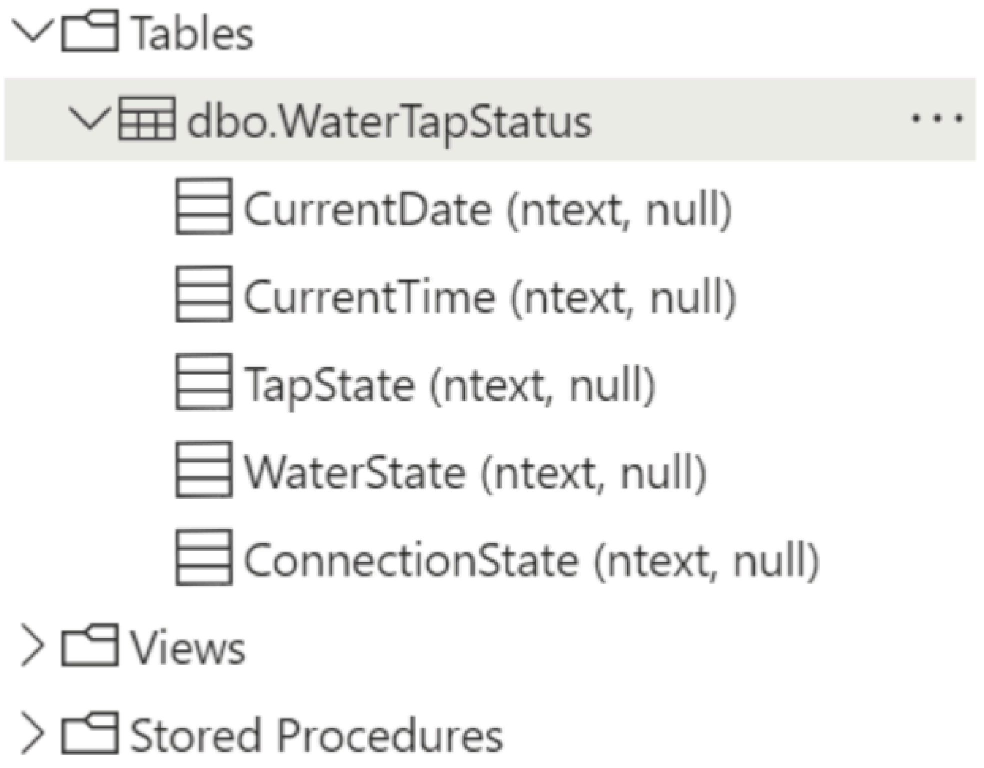

Initialize Signal generating Code
Water Tap Signal: OFF
Water Tap Signal: ON
Exponentially distributed for changing states
Water Tap Signal: ON with water running
Water Tap Signal: ON with no water running
Exponentially distributed for changing states
Water Tap Signal with: Date, Time, Tap Status and Water Status sent to Azure Database every few minutes
Azure SQL Database receives period push requests.
Destination device requests to pull data from the SQL database.
UI Detects and posts Tap Off
UI Detects and posts Tap On, Water Running
UI Detects and posts Tap On, No Water (Possible Outage)

Figure 1. System Design and Architecture

In Figure 1 we have our proposed system design which we have worked on for the semester. While it is relatively simplistic we divided the work into five major sections. First is the code about generating the data which uses probabilities to randomly change states, thus testing the various possible states. Next, we have the data upload, storage and download phases, as IoT devices employ cloud services for scalable storage, therefore we decided to employ the use of an Azure Cloud server with an SQL Database [4]. Therefore, with every set of data being generated, we learned how to upload and download seamlessly to the Azure database. In the storage phase, we needed to employ the free version of Azure provided by Microsoft so that we had a reliable system in place [5]. The final phase is the UI which is meant to extract the raw data from the database and using logic infer the current status of the Water Tap System and display this to the user.

System Implementation approach

As depicted above, our team decided to simulate the system using probabilities to generate random data signals. We split the workload over 5 individual tasks. For data generation, data upload and data download we utilized Python coding as this diverse language is equipped with numerous libraries to streamline the process. The most notable libraries for data generation were NumPy and the Python Random module which aided in computing the probabilities for switching between states within the model [6][7]. Furthermore, adding a library like DateTime allows for an accurate depiction of the time a signal is transmitted [8]. Additionally, the most important library for the uploading and downloading coding components was Python ODBC which streamlines accessing the databases as seen in the Code.txt file attached with the report [9]. As discussed previously, utilizing the free SQL services provided by Microsoft Azure, we created a table with the entries described in Figure 2.



```
Tables
  dbo.WaterTapStatus
    CurrentDate (ntext, null)
    CurrentTime (ntext, null)
    TapState (ntext, null)
    WaterState (ntext, null)
    ConnectionState (ntext, null)
Views
Stored Procedures
```

Figure 2: SQL tables and entries for Water Tap Status.

Finally, we designed the UI implementation in Figma to represent the possible resulting states and layout that will occur at the user end of the device which will be shown in the testing results section [10].

Data safety and security considerations

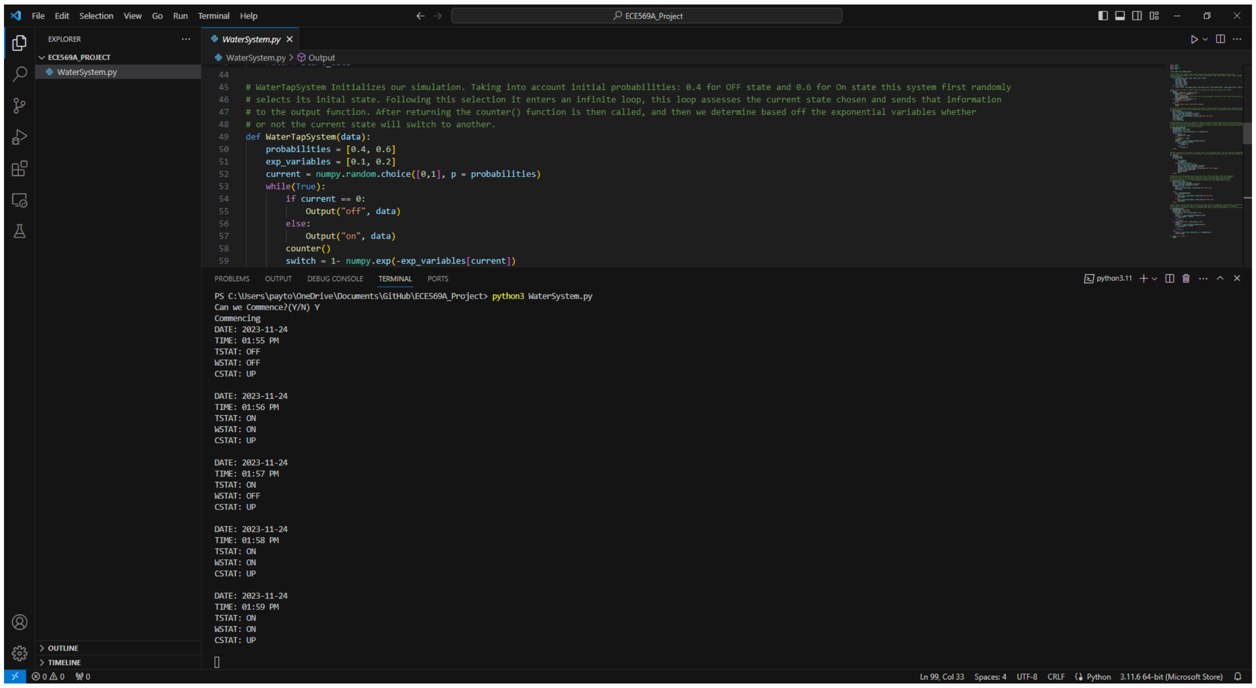
While our system does not implement any data security methods, it is important to note that in a commercial environment, it is important to ensure data security and safety. In a practical model, additional information such as usernames, locations and other personally identifiable information would be present in the upload of data to the cloud servers. This would be to locate the end-user device which should be configured with a specific key or username in the system. The most pertinent security method would be encryption for data at rest and in transit. Azure services come with server- or client-side encryption models to protect information in transit [11]. However, we employed no encryption feature for data transmission over the ports connecting to the cloud server. As such if any man-in-the-middle attack occurred there would be raw information readily accessible without any difficulty. Fortunately, there are many techniques which we can employ to encrypt data in transit. With recent advancements in AI, we can even consider methods which are outside the realm of conventional encryption methods which encrypt all data regardless of confidentiality. One could employ AI models to situationally determine which data is confidential and thus require encryption using a common key by training the model with Machine Learning methods [12].

Testing Plan

Our testing plan design was relatively simple. Given that our model was pretty straightforward requiring only simple coding and a properly configured Azure SQL database server, we only needed to develop a plan for the fully configured system. Unfortunately, fully incorporating the user interface into the system was out of the bounds for the task of this team as we had no front-end development experience. Given this fact, the UI is a series of screenshots demonstrating the possible states for the system and was not involved in the testing plan. Therefore, to demonstrate the feasibility of our system where the end-user device is around 2 kilometres away we established a testing plan where the team member responsible for data uploading is at least two kilometres away from the team member responsible for data downloading.

Testing Result

For data generation and uploading we configured the WaterTapSystem.py file to output a log representing the information that will be uploaded to the SQL database. Unfortunately, the actual data used in the real-time testing was not captured, however, in Figure 3 there is an example of the data generation logs with the actual data upload disabled as this is simply a proof of concept and the Azure server has since been shut down.

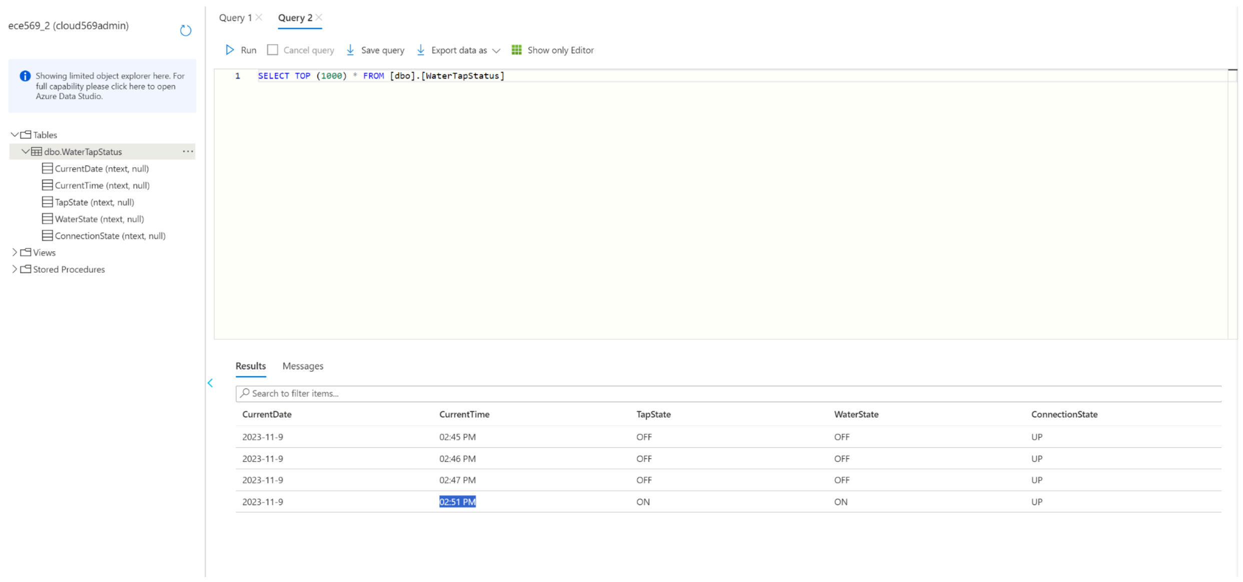


```python
# WaterTapSystem Initializes our simulation. Taking into account initial probabilities: 0.4 for OFF state and 0.6 for On state this system first randomly
# selects its inital state. Following this selection it enters an infinite loop, this loop assesses the current state chosen and sends that information
# to the output function. After returning the counter() function is then called, and then we determine based off the exponential variables whether
# or not the current state will switch to another.
def WaterTapSystem(data):
    probabilities = [0.4, 0.6]
    exp_variables = [0.1, 0.2]
    current = numpy.random.choice([0,1], p = probabilities)
    while(True):
        if current == 0:
            Output("off", data)
        else:
            Output("on", data)
        counter()
        switch = 1- numpy.exp(-exp_variables[current])
```

```
PS C:\Users\payto\OneDrive\Documents\GitHub\ECE569A_Project> python3 WaterSystem.py
Can we Commence?(Y/N) Y
Commencing
DATE: 2023-11-24
TIME: 01:55 PM
TSTAT: OFF
WSTAT: OFF
CSTAT: UP

DATE: 2023-11-24
TIME: 01:56 PM
TSTAT: ON
WSTAT: ON
CSTAT: UP

DATE: 2023-11-24
TIME: 01:57 PM
TSTAT: ON
WSTAT: OFF
CSTAT: UP

DATE: 2023-11-24
TIME: 01:58 PM
TSTAT: ON
WSTAT: ON
CSTAT: UP

DATE: 2023-11-24
TIME: 01:59 PM
TSTAT: ON
WSTAT: ON
CSTAT: UP
```

Figure 3: Example of Data Generation

Figure 4 represents the SQL database tables after the data has been uploaded from the user's computer. The user uploading data allowed the system to run for 6 minutes which it uploaded to the database 4 times.

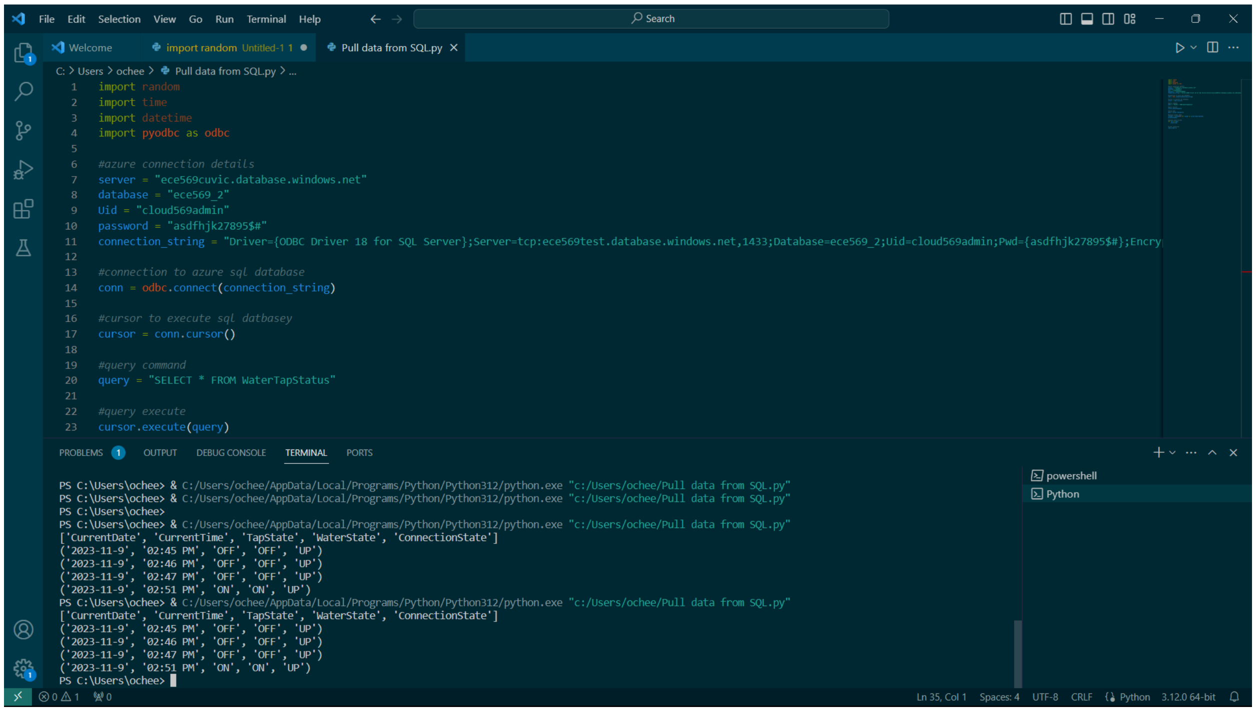


```sql
SELECT TOP (1000) * FROM [dbo].[WaterTapStatus]
```

CurrentDate	CurrentTime	TapState	WaterState	ConnectionState
2023-11-9	02:45 PM	OFF	OFF	UP
2023-11-9	02:46 PM	OFF	OFF	UP
2023-11-9	02:47 PM	OFF	OFF	UP
2023-11-9	02:51 PM	ON	ON	UP

Figure 4: Actual Data after upload in Azure SQL Database.

Figure 5 represents the same testing data being pulled from the Azure server SQL Database to the group member representing the end user. As such given that we did not configure the member who designed the Python file for pulling data simply outputted the data to the command line without making inferences for the system state.



```python
import random
import time
import datetime
import pyodbc as odbc

#azure connection details
server = "ece569cuvic.database.windows.net"
database = "ece569_2"
Uid = "cloud569admin"
password = "asdfhjk27895$#"
connection_string = "Driver={ODBC Driver 18 for SQL Server};Server=tcp:ece569test.database.windows.net,1433;Database=ece569_2;Uid=cloud569admin;Pwd={asdfhjk27895$#};Encry

#connection to azure sql database
conn = odbc.connect(connection_string)

#cursor to execute sql datbasey
cursor = conn.cursor()

#query command
query = "SELECT * FROM WaterTapStatus"

#query execute
cursor.execute(query)
```

```
PS C:\Users\ochee> & C:/Users/ochee/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ochee/Pull data from SQL.py"
PS C:\Users\ochee> & C:/Users/ochee/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ochee/Pull data from SQL.py"
PS C:\Users\ochee>
PS C:\Users\ochee> & C:/Users/ochee/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ochee/Pull data from SQL.py"
['CurrentDate', 'CurrentTime', 'TapState', 'WaterState', 'ConnectionState']
('2023-11-9', '02:45 PM', 'OFF', 'OFF', 'UP')
('2023-11-9', '02:46 PM', 'OFF', 'OFF', 'UP')
('2023-11-9', '02:47 PM', 'OFF', 'OFF', 'UP')
('2023-11-9', '02:51 PM', 'ON', 'ON', 'UP')
PS C:\Users\ochee> & C:/Users/ochee/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ochee/Pull data from SQL.py"
['CurrentDate', 'CurrentTime', 'TapState', 'WaterState', 'ConnectionState']
('2023-11-9', '02:45 PM', 'OFF', 'OFF', 'UP')
('2023-11-9', '02:46 PM', 'OFF', 'OFF', 'UP')
('2023-11-9', '02:47 PM', 'OFF', 'OFF', 'UP')
('2023-11-9', '02:51 PM', 'ON', 'ON', 'UP')
PS C:\Users\ochee>
```

Figure 5: Pulled Data

Figures 6 through 9 show the resulting data represented in our designed User Interface showing the date, time and the current status of the tap with the settings being “Tap off” or “Tap on”. Furthermore, in Figure 10 we have the “Power off” state which indicates that the system is not connected or that an emergency has happened and the system has somehow been disconnected.

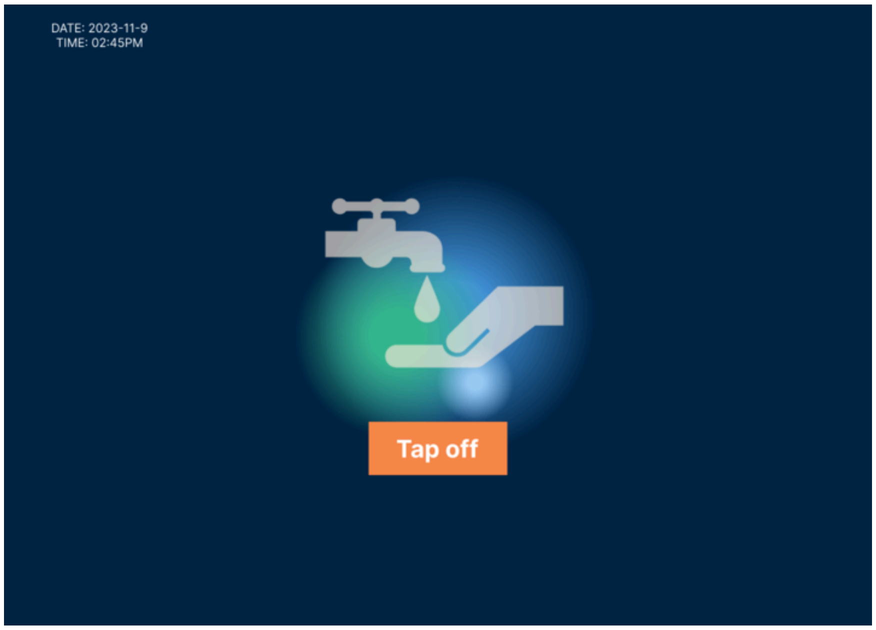


Figure 6: UI representation of Pull 1

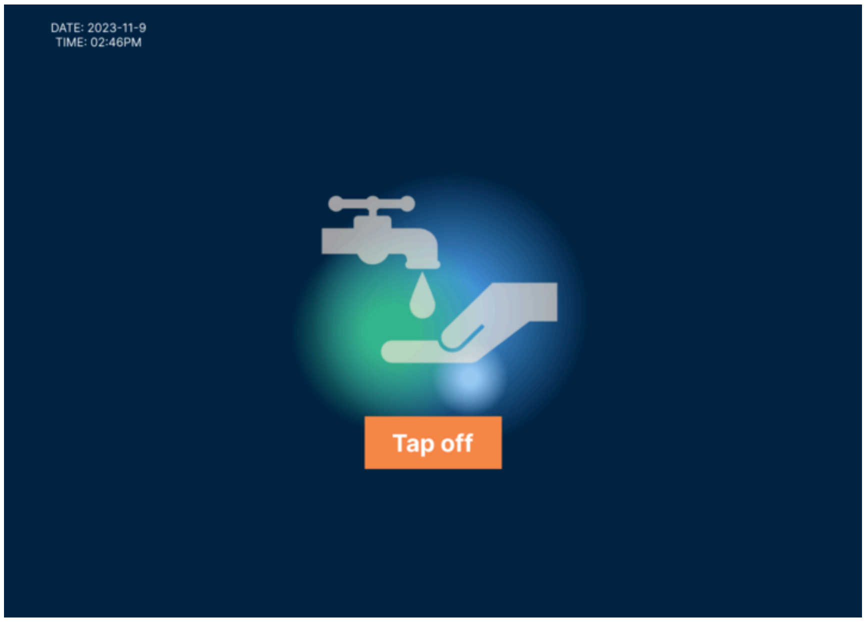


Figure 7: UI representation of Pull 2

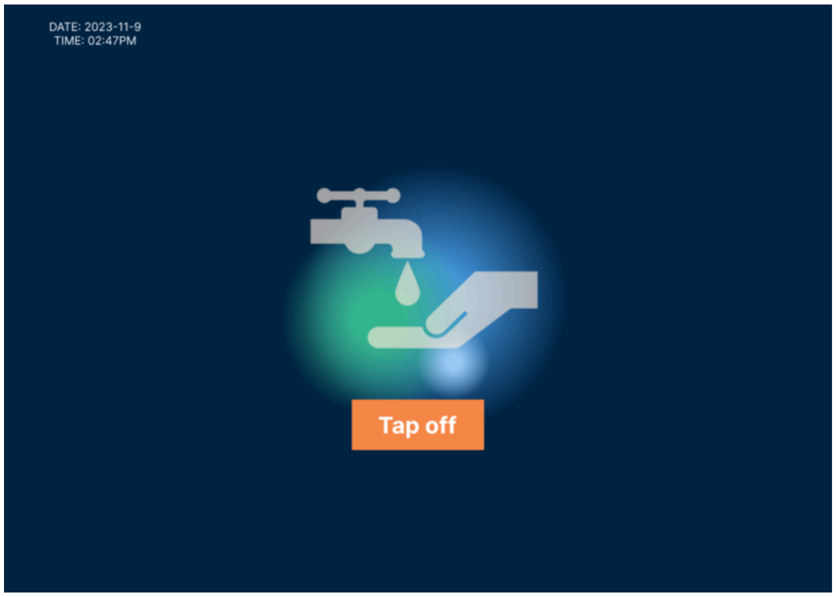


Figure 8: UI representation of Pull 3

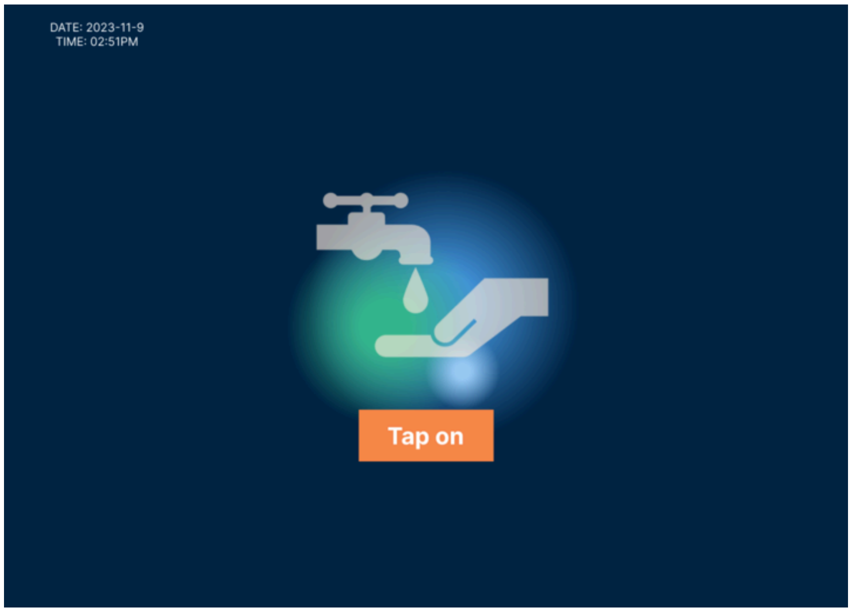


Figure 9: UI representation of Pull 4

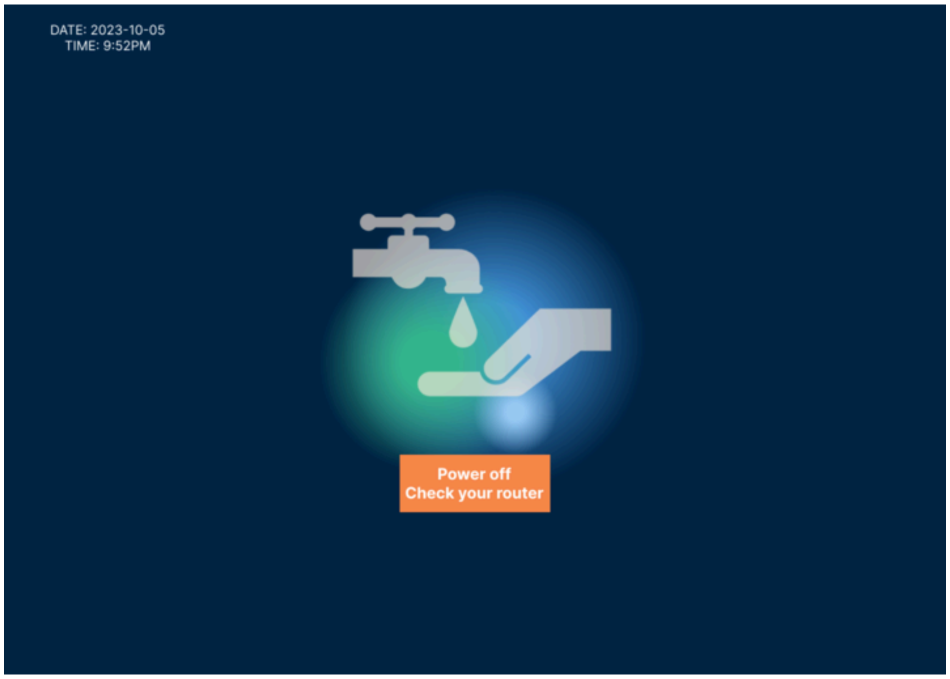
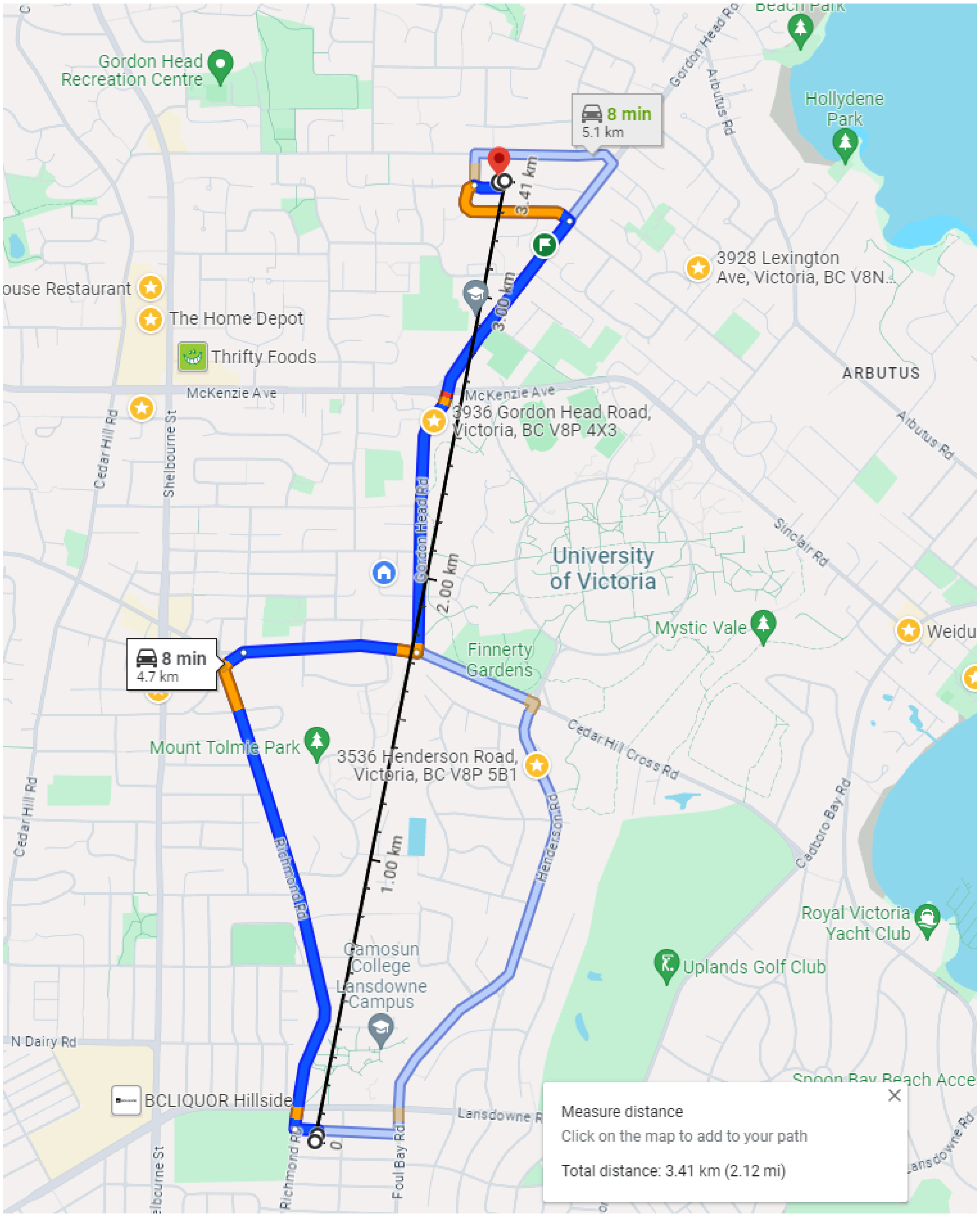


Figure 10: UI representation of Emergency.

Our final Figure 11 shows us the exact position of the group members pushing and pulling the data from the cloud server. Data pushing was done at 1875 Watson Street, Victoria BC, Canada and data pulling was done at 1858 Elmhurst Place, Victoria BC, Canada. As such utilizing Google Maps we can see that the approximate distance between the locations measured by the application is 3.41 km [13]. With this in mind and the aforementioned screenshots from the testing, we showed empirically that this system can transmit the water tap data over 2 kilometres at least.



Measure distance
Click on the map to add to your path
Total distance: 3.41 km (2.12 mi)

Figure 11: Distance between the point of Upload and Download.

Project Planning and Timeline

Our team began this project swiftly after the professor elaborated on the parameters for it. By September 29th, we had already decided on the system we would build and the distribution of tasks. Given that using physical sensors was not required for this project we wanted to make sure that we did not spend any money. As such we only needed to use 2 computers with code made to simulate the device and the free services provided by the Azure Cloud Servers. I (Payton) was responsible for creating the code meant to generate the data using probability coupled with the random number-generating features in Python libraries. Yunfei (Alex) was responsible for adding the data uploading features to the aforementioned data generation code. Peter set up and configured the SQL database for this assignment. Oche was responsible for data downloading from the SQL database. Finally, Heeba was responsible for creating the UI for displaying the information to the end-user. Throughout our weekly meetings, we decided to complete our prospective sections by the first or second week of November to allow for lots of time to complete the reports. I completed the Data generation portion on October 27th, Data upload and configuration of the Azure SQL server was completed by November 9th and finally, Data download and UI prototyping was finished by November 11th and thus we could complete our reports.

Summary

Given the state of the world with rising droughts and water shortages, the need for wirelessly connected water detection systems is becoming increasingly necessary. Average people or even companies can use these systems commercially to measure user consumption of water in faucets and taps. While our system only simulates possible water tap states, we effectively utilized cloud services to transmit small packets directly to the database which can then be sent to end-users to monitor the real-time system. We have tested our proposed system over a two-kilometer range proving that it is viable and able to meet the requirements of the project description.

References

[1] “Government of Canada,” Language selection - Agriculture and Agri-Food Canada / Sélection de la langue - Agriculture et Agroalimentaire Canada, https://agriculture.canada.ca/en/agricultural-production/weather/canadian-drought-monitor/current-drought-conditions#a1 (accessed Nov. 25, 2023).

[2] S. Basu, A. Ahmed, H. Pareek and P. K. Sharma, "Autonomous Water Flow Control And Monitoring System," *2022 Interdisciplinary Research in Technology and Management (IRTM)*, Kolkata, India, 2022, pp. 1-4, doi: 10.1109/IRTM54583.2022.9791534.

[3] J. Misachi, “Which country has the most freshwater?” WorldAtlas, https://www.worldatlas.com/articles/countries-with-the-most-freshwater-resources.html (accessed Nov. 25, 2023).

[4] “Azure SQL Database – managed cloud database service: Microsoft Azure,” – Managed Cloud Database Service | Microsoft Azure, https://azure.microsoft.com/en-ca/products/azure-sql/database/ (accessed Nov. 25, 2023).

[5] “Free services Microsoft Azure,” Free Services Microsoft Azure, https://azure.microsoft.com/en-us/pricing/free-services (accessed Nov. 25, 2023).

[6] NumPy, https://numpy.org/ (accessed Nov. 25, 2023).

[7] “Random - generate pseudo-random numbers,” Python documentation, https://docs.python.org/3/library/random.html (accessed Nov. 25, 2023).

[8] “DateTime - basic date and time types,” Python documentation, https://docs.python.org/3/library/datetime.html (accessed Nov. 25, 2023).

[9] “Pyodbc,” PyPI, https://pypi.org/project/pyodbc/ (accessed Nov. 25, 2023).

[10] “The Collaborative Interface Design Tool,” Figma, https://www.figma.com/ (accessed Nov. 25, 2023).

[11] Msmbaldwin, “Azure encryption overview,” Microsoft Learn, https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview (accessed Nov. 25, 2023).

[12] M. Ghouse, M. J. Nene and V. C., "Data Leakage Prevention for Data in Transit using Artificial Intelligence and Encryption Techniques," *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*, Mumbai, India, 2019, pp. 1-6, doi: 10.1109/ICAC347590.2019.9036839.

[13] “Google Maps,” Google maps, https://www.google.com/maps (accessed Nov. 25, 2023).