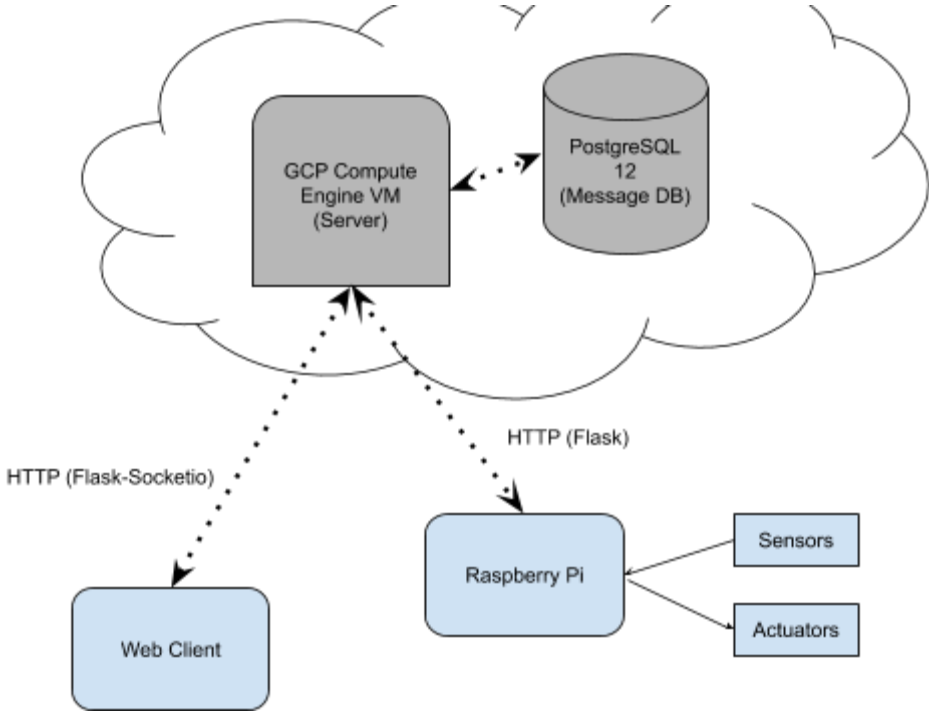
Final Project Writeup

By Lorand Cheng

Summary:

This project is a morse code translator and messenger system that can be used to communicate morse code messages from the raspberry pi to a web client. The web client user can then respond through their web interface and send messages back to the raspberry pi, as well as view the message history in the database. Communication happens through a GCP Compute Engine VM, which talks to a CloudSQL Postgres 12 database.

Architecture:



Components:

- **Raspberry pi:** The raspberry pi detects button presses in various combinations of lengths, and decides to take actions based on the result of the combination. A state machine is used to determine what action to take given the inputs, with states 0: button not pressed, 1: button pressed, and 2: receiving message. The raspberry pi uses the grovepi for sensing and actuating, and it uses Flask to communicate with the server via HTTP. The rpi.py file uses several other classes,

including a message handler for handling HTTP actions, an LCD handler for updating the display, a message notifier to receive incoming messages, and a morseCode translating library that uses a binary tree to decode individual letters. Those other classes employ various tools like timezone conversion, json parsing, timer interrupts, etc. in order to complete their tasks.

- **Web client:** When a client connects on port 4200 of the web server, they are served a simple interface that allows users to send messages as well as view the message history. This interface is written using Bootstrap, JQuery, and SocketIO. Fields are “command”, “name”, and “message”. If the command is “send”, messages can be sent with the name and message fields, and when the command is “history” the other fields are ignored and a request for the entire message history is sent to the server
- **Cloud server:** The cloud server is in charge of handling requests and returning requested data, as well as updating and querying the database for the necessary resources. It uses Flask and SocketIO to handle HTTP transactions and to notify connected clients of new messages. The cloud server uses a message manager class to update and query the database, using psycopg2. The config info for the database is also stored in a separate database.ini file that is hidden from the public repo to hide database credentials.
- **Cloud database:** A simple relational database with only one table containing the entire message history. Columns in the database are: id, sender, message, timestamp. Only connects to the cloud server directly.

Other Notes and Reflection:

While the current system works decently well, it is far from complete and is a pretty rough patchwork of technologies. For example, using a custom notifier class for the rpi while using socketio for the web clients is clunky. I chose HTTP over MQTT for this project because I did not intend it to be a many-to-many network, but HTTP has the downside that it is not bidirectional and requires additional tools to implement notifications. In addition, there are not many security measures taken other than hiding the database credentials, so this does not even come close to a production implementation. Lastly, if I had to clean up one area of this project it would be the rpi.py file and how the state machine is handled. The way I split up the functions between the state machine, explicitly defined functions, and the LCD handler is extremely messy and could have used a lot more planning in order to craft a more organized strategy.