**IOT CA2**

**The Smart Coaster (Alpha Release)**



Jamie Lawlor

Caitlin Maguire

Luke Hilliard

Shahzad Shabeer

**Version 1**

**22 November 2024**

**Table Of Contents**

1. [**SYSTEM ARCHITECTURE……...………………………………………………………..**.](#Hardware_Section) **3**
   1. IoT Elements………………………………………...………………..………………………..…...
   2. Web Server…………….……………………………………………….………...……………..…...
   3. Database……………………………………………………………………………………………..
   4. Flow of Data………………………………………………………………………………………….
   5. Security in transit, at rest……………………………………………………………………………
   6. Fritzing sketch……………………………………………………………………………………….
2. [**ALPHA PROTOTYPE.………….…………………….**](#Data_Section)**..........................................................**
   1. [Hardware](#Data_Gathering)…….………………………...…………………………………………………...………..
   2. [Web](#Data_Storage) Server...……………………………………………………………………………….………..
   3. Hosted on AWS……..……………………………………………………………………………..
   4. PubNub Communication…………………………………………………………………………….
   5. Database……………………………………………………………………………………………..
   6. Security……………………………………………………………………….……………………..
3. [**DEPLOYED TO AWS…….…………………………………………………….**](#Security_Section)**...................**

**3.1** [Custom](#Physical_Security) Domain..…………………………………………………………………………………….

**3.2** AWS……….………………………………………………………………………………................

1. [**PUBNUB……………………...………………………….………………………………….**](#UI_Section)
   1. Communication…………………………………………………………………………...………
2. [**SECURITY……………………...……………………….………………………………….**](#UI_Section)
   1. IoT device………..………………………………………………………………………...………
   2. Access to communication channels………………………………………………………………..
   3. Database………………………………………………………………………………………….
   4. Web Server…………………………………………………………………………………………
   5. Data In Transit……………………………………………………………………………………..
3. **System Architecture**

**A diagram of a network

Description automatically generated**

Figure System Architecture diagram created on Draw.io (Jamie)

**1.1 IoT Elements -**

**1.2 Web Server – Jamie**

**1.3 Database - Caitlin**

**MongoDB Atlas**

Using MongoDB Atlas I created a drinks collection within the Sipify database. The drinks data holds the drink name and temperature ranges. The drinks data is displayed on the front end using a post request from the server, the user will select a drink, and the name of the selected drink will come back to the database. Within the drink status collection, a new record will be created with the selected drink name.

Once the current temperature comes in from the server this will also be added to the drink status record. The current temperature will be read and compared against the minimum and maximum temperatures of the selected drink and a notification will be issued based on this. The updated drink status record will then be sent to the front end to be displayed on the barista mode page. The user will then be able to view the name of the selected drink, the current temperature and the corresponding notification.

**1.4 Flow of Data – Jamie**

**1.5 Security in Transit, at rest –**

**1.6 Fritzing sketch –**

**2. Alpha Prototype**

**2.1 Hardware – Luke**

**2.2 Web Server – Jamie**

**2.3 Hosted on AWS - Jamie**

**2.4 Pubnub communication – Jamie**

**2.5 Database – Caitlin**

**Gathering Data**

For the database aspect of the project, I started off with creating a local database using MongoDB Compass. I created a database called Sipify, with three collections inside. I gathered coffee temperature data and created a dataset based off my research for the project.

To track coffee temperatures and issue notifications based on the current temperature, we firstly needed some data to compare the temperature against. I carried out some research to find out what exactly are the ideal coffee drinking temperatures, and most importantly discovering the minimum and maximum temperatures. The maximum temperature being the hottest temperature the coffee can be served at, and the minimum being the coldest. I also got some data about tea, as we would like the Smart Coaster to be used by everyone not only coffee drinkers. I created a csv file based on the drinks data and imported this into MongoDB Compass into my Sipify database. My aim was to find the temperature ranges based on the different coffee types and get the average drinking temperatures. I found the following websites helpful at figuring out the temperature ranges for each drink type:

<https://thedrinksproject.com/how-hot-is-coffee-served/>

<https://weareliferuiner.com/perfect-temperature-hot-chocolate-the-ultimate-guide/>

<https://letsdrinktea.com/what-is-the-right-temperature-for-drinking-tea/>

**Drinks Data:**

A screenshot of a computer

Description automatically generated

**Notifications Data:**

A screenshot of a computer

Description automatically generated

**Drink Status Data Example:**

**A screenshot of a spreadsheet

Description automatically generated**

**Connecting to the Database:**

I created the drinks, notifications and drink status collection in the local database using MongoDB Compass. For the drink status I first used sample data to test the displaying of data, the idea of the drink status collection is to hold all data related to the drink the user has set on the coaster. Starting with the database I used the local database to get data displaying and functions working, after this I began working on the MongoDB Atlas.

**MongoDB Atlas:**

I created a cluster on MongoDB Atlas called sipify-mongodb, within the cluster I have three collections similarly to MongoDB Compass, I added the data for each collection

**Flask App:**

I defined each of the routes in the app.py file within the flask app, for this release we have a loading screen, drink selection page, barista mode page and a view temperatures page. Starting with the drink selection page, I created a view all drinks function to display the list of drinks for the user on the front end.

By calling this function in the drinks selection route it will display a list of all the drink names as radio buttons, allowing the user to select a drink. Before the project was fully connected, I first used an input box for the user to input the current temperature to show how the database will deal with the temperature. Once the user inputs the temperature and selects a drink, this data is sent to the database using a POST call on Jamie's server side, I then included a function to add the inputted data into the drink status collection.

On the barista mode page, the selected drink, current temperature and the notification matching the temperature are displayed.  To get the notification I have compared the inputted temperature with the minimum and maximum temperatures. Within the function I am checking whether the selected drink matches any of the drinks that is in the drinks collection, the current temperature is then compared against the drinks minimum and maximum temperature ranges. If the drink is in the colder range, it will output the notification message associated with the cold status, it is the same for the hotter range the hot status notification will be outputted. The notification is passed in to the add drink status to be displayed to the user.

* 1. **Security – Jamie/Shahzad**

**3. Deployed to AWS**

* 1. **Custom Domain – Jamie**
  2. **AWS – Jamie**

**4. Pubnub**

* 1. **Communication – Jamie**

**5. Security**

* 1. **IoT Device –**

### **Prevent direct access to Iot device (Shahzad)**

Disabling unused gpiopins

To enhance the security , I disabled unused GPIO pins in the system. This step minimizes vulnerabilities by preventing unauthorized access or manipulation of the hardware. By limiting access to only essential components, the device's overall security and stability will be significantly improved.

### **Box to place device**

I attempted to find a suitable metal box to house the Pi for improved durability and to prevent direct access. However, I was unable to get a metal enclosure in time. Instead, I sourced a plastic box as an alternative. After discussion with the team, we decided not to proceed with the plastic option

A plastic container on a counter

Description automatically generated

### **Adding Security Tests**

I implemented and executed security tests for the project to ensure the application is robust against potential threats like SQL injection and XSS attacks. These tests involved simulating harmful inputs to confirm that the app handles them safely. As a result, the app is now more secure and reliable

* 1. **Access to communication channels – Jamie**
  2. **Database – Caitlin**

**MongoDB Atlas**

I enabled the network access to allow any IP Address to connect this will allow my team members to connect, and I also added the IP address of our server. Within the database access, I created an admin user for myself and added in my team members as users with read and write access to the database. A username and password is required to gain connection to the database on MongoDB Atlas.

**Database Access on MongoDB Atlas:**

A screenshot of a computer

Description automatically generated

* 1. **Webserver – Jamie**
  2. **Data in transit -**