CPEN 391
Module 3 Final Report
The Animal Guardian (TAG)

Team 10

Andrew Shieh
Aayush Bisht
Elbert Ng
Shade Wong
Vincent Sastra

TABLE OF CONTENTS

1. INTRODUCTION

Illegal wildlife trade has escalated dramatically over the last decade. Human population growth, increasing wealth and access to wildlife and improved global transport links have all played a part. Conservation interventions have historically focused on regulation by introducing new and stronger legislation and trade controls. However, regulation alone is not sufficient to combat illicit wildlife trade. Living in an era where technology is advancing at an unprecedented rate, we can take advantage of the current technology by utilizing it to protect species from poachers while tackling the root causes of illegal trade.

2. PRODUCT DESCRIPTION AND TARGET MARKETS

The Animal Guardian (TAG) features a Raspberry Pi 4 computer with multiple sensors attached, and encased within a durable waterproof case. TAG can be attached to a subject of various sizes to monitor its live location, ambient temperature, and heart rate. It is accompanied with a user friendly web application that gives users access to the raw/processed sensors data collected. In addition, the web app provides the features: (1) Animal Activity Prediction, (2) Wildfire Detection, (3) Geofencing Alert and (4) Human Presence Detection Notification. TAG is also equipped with a camera to capture surrounding images for human presence prediction purpose to notify users timely, hence minimizing the poaching risk.

TAG embarks to ease the pressure of animal monitoring in a wildlife preservation sanctuary. It provides ease of location tracking and abnormal behaviour alerts along with human presence detection which makes the process of at risk animals monitoring undemanding.

TAG can also be used for domestic pet monitoring for owners that have to leave their beloved ones unattended regularly. The activity prediction and human presence detection features allow the pet owner to better understand the pet's state, hence ensuring its safety at all times.

3. PRODUCT REQUIREMENTS AND SPECIFICATIONS

The requirements for the TAG based on the needs of target markets are:

1. Waterproof casing to ensure the product is resilient to different weather conditions
2. Portable and long-lasting battery supply to minimize the replacement of batteries, hence the interruption of animal activity
3. Robust service that works in an environment without WiFi
4. Real-time data processing and alert system to notify users about potential risks promptly
5. Real-time location monitoring service with geofencing capabilities that support both short and long-range tracking

6. Wildfire detection system that notifies users when the ambient temperature went beyond certain threshold
7. Animal activity monitoring system that provides users with the information of an animal daily activities
8. Responsive application interface that supports different screen sizes, eg. desktop, laptop, tablet, mobile

The specifications for the TAG are:

1. Custom 3D-printed casing using weatherproof material, PP GF30 and with simple design model to reduce the chances of water leak through *
2. Rechargeable portable battery pack that can last at least 6 months long without power supply *
3. LoRaWAN connectivity with integrated antenna and compatibility with LoRaWAN networks worldwide to produce a Wi-Fi-independent tag *
4. Websocket server deployment on the cloud to facilitate the real-time communication between DynamoDB database and the web application
5. High-precision NEO-7m GPS module integration that supports accurate indoor positioning for household pet monitoring coupled with user-defined geofence through the web application
6. DHT-11 Temperature and Humidity sensor for ambient temperature data collection and real-time notification alert system on web application when received temperature exceeds 49°C
7. ADXL 345 3-axis accelerometer data and pulse sensor data collection to construct animal activity prediction decision tree that classifies each set of data into one of the four categories: sleeping, sitting, walking, abnormal
8. Multiple versions of web application interface designs based on different commonly used device screen sizes, eg. mobile and desktop, using Bootstrap and CSS

** Specifications that would have be implemented if resources were available*

4. DESIGN CONSTRAINTS

The HPS/FPGA computer system was used in our design, due to the ease of integration of NIOS II system with the components on RFS daughter board and the mandatory use of ARM A9 HPS on the DE1-SoC board specified in the course project requirements document. However, the use of HPS restricts the performance of the hardware-accelerated task in our system. Since the dot product accelerator was implemented on the HPS side, when the NIOS II system receives data via Bluetooth on the FPGA side, an additional step is required to transfer data from FPGA side memory to HPS side memory to trigger the hardware accelerator. Similarly, an additional step is required to transfer the result from HPS to FPGA, so that it can be sent to the cloud database for storage via the Wi-Fi component implemented in the NIOS II system.
Another constraint faced during the development process, which delayed the progress of the hardware team by a week, was the abnormal behaviour of the Raspberry Pi 4, where it would

work correctly when tested within the Vancouver area, but would not save images correctly when powered up outside of the Vancouver area. The Pi showed constant red on power up, but would not work with the sensors properly. We suspected that this issue was related to the power supply to the Raspberry Pi 4 computer. As a solution, we did the rest of project testing and demonstration in Vancouver with the appropriate power supply.

For the machine learning component, manual hyperparameter tuning approach was used for our hyperparameter optimization which incurred heavy penalties on our project timeline. This was necessary to improve the accuracy of our prediction model.

The animal activity prediction model was deployed and ran on the cloud. This design choice was made to offload the computational intensive task from the Raspberry Pi computer, due to its hardware limitation. Another reason for picking this approach was to preserve the modularity of the project, as the main task of Raspberry Pi should solely be sensors data collection. However, by doing so the cold start time for a new sensor data entry creation process in DynamoDB table is relatively long, which takes approximately 15 to 30 seconds. Despite that, we decided that this is still a better solution, as the considerable delay only occurs when the function is cold and our function performs as normal when warm.

5. DESIGN APPROACH

The design approach used in our project closely followed the engineering design process: (1) Project ideas generation, (2) Target markets identification, (3) Product requirements identification, (4) High-level designs generation, (5) Prototype implementation, and (6) Verification and validation.

A. Initial Conceptual Design

The challenges faced by wildlife conservation organizations and/or domestic pet owners in monitoring their animals were first identified: *(i)* Illegal poaching activities, *(ii)* Human-animal conflict, *(iii)* Aggressive animal behaviour tracking, and *(iv)* Environmental issues, such as wildfire. Then, a list of possible solutions that address each challenge was produced: *(i)* Location tracking, *(ii)* Activity and heartbeat monitoring, *(iii)* Human presence detection, *(v)* Geofencing alert, and *(vi)* Wildfire detection alert. Followed by an extensive research on sensors, cloud services, frontend tech stack, and FPGA system design. Finally, the implementation of the prototype was broken down into five subsystems: *(i)* Web Application, *(ii)* Serverless Backend (AWS), *(iii)* Machine Learning, *(iv)* Raspberry Pi 4, and *(v)* HPS/FPGA Computer System, to preserve the modularity of the project and easier task distribution within group members.

B. Main Functionalities

1. **Location tracking**
    - Real-time GPS coordinates are displayed for every registered tag on a map
2. **Activity and heartbeat monitoring**

- Animal heartbeat rate over time line graph is displayed
- Animal daily activities are displayed in a pie chart, eg. 10% walking, 20% resting, 65% sleeping, 5% abnormal behaviour
- Real-time notification is sent to the user for any detected abnormal behaviour

3. **Geofencing alert**
   - Real-time notification is sent to the user when animal is moving out of defined area
   - Users are able to define geofence and risky zone boundaries for every registered tag with multiple GPS coordinates that form an enclosed area when connected with adjacent coordinates
4. **Human presence detection**
   - Real-time notification is sent to user when unauthorised personnel is detected near the animal tag
5. **Wildfire detection alert**
   - Ambient temperature over time line graph is displayed
   - Real-time notification is sent to user when the detected temperature goes beyond 50°C

C. High-Level Design

The following diagram provides a mapping of the software and hardware components required for the main functionalities implementation on our web application described in the previous section. Each block represents a subsystem implemented by one team member, and the arrows indicate the data flow between each subsystem.

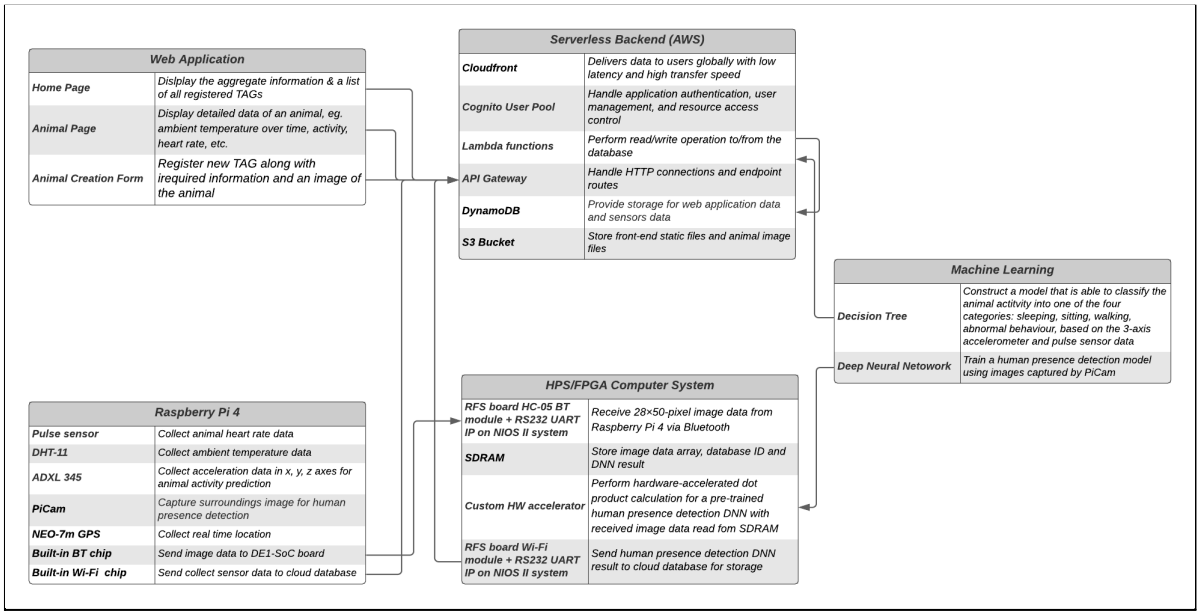

Diagram 5.1 Subsystem-Component - Functionality Relationship Diagram

The following diagram provides an in-depth design for the AWS architecture, where each individual components are separate AWS services. The implementation description can be found in *Section 6. B.*

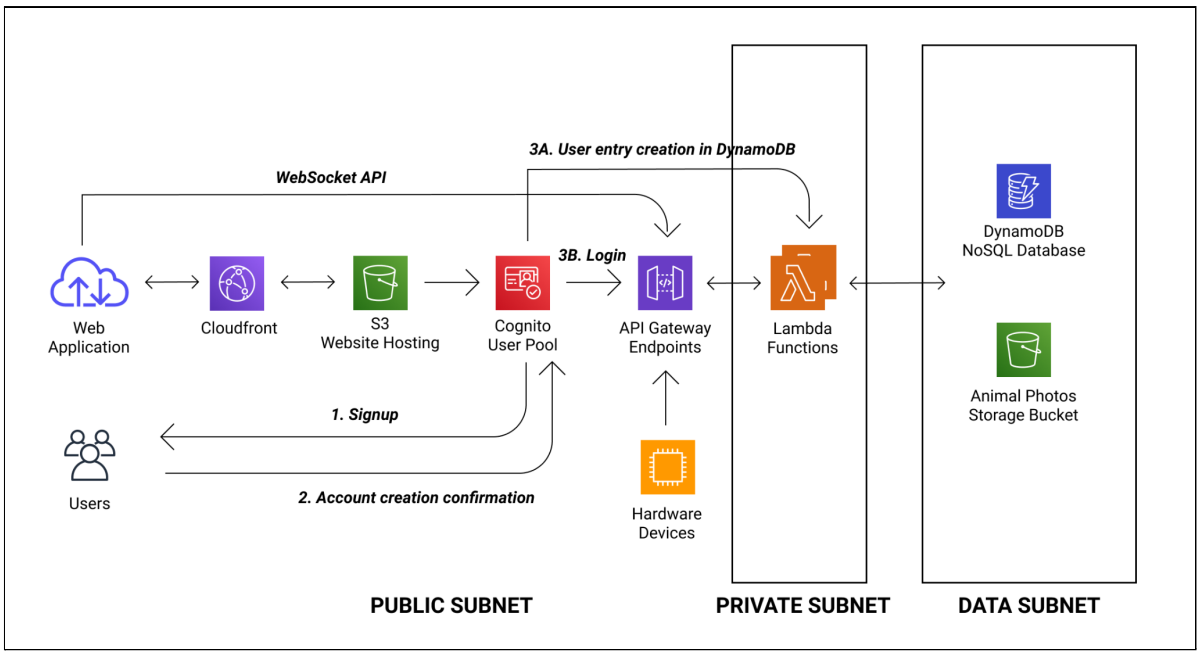

Diagram 5.2 AWS-based Serverless Backend Architecture

The following diagram provides an in-depth design for the HPS/FPGA system architecture. The implementation details can be found in *Section 6.E.*

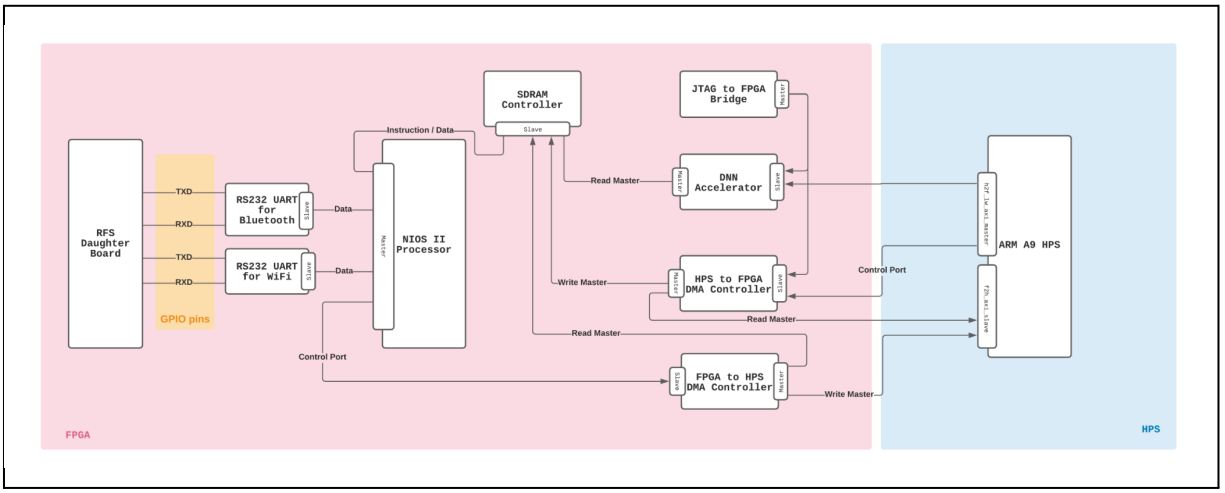

Diagram 5.3 HPS/FPGA System Architecture

D. Interfaces

1. Web Application - Serverless Backend (AWS) Interface

Data (JSON format) transfer between the web application and the serverless backend is established using the HTTP protocol and the interface should provide the following functionalities:

a. Get Particular TAG Information by Username and TAG ID

All animal-related information can be obtained from the DynamoDB through an API Gateway endpoint that queries the DynamoDB table.

b. Real-time Notifications and Data Update

A web socket connection that updates the web application with live data and notifications continuously.

c. **Update Geofence Coordinates**
Web application can write new or update existing geofence coordinates for a particular user and TAG pair in DynamoDB via an API Gateway endpoint.

d. **Register New TAG(s)**
Web application can create a new entry in the DynamoDB table to store the new TAG information and store the animal image in S3 bucket through an API Gateway endpoint.

2. **Raspberry Pi 4 - Serverless Backend (AWS) Interface**
Raspberry Pi 4 can create a new entry in the DynamoDB table to store a new set of sensor readings for a particular TAG by accessing the private database using AWS SDK.

3. **Raspberry Pi 4 - HPS/FPGA Computer System Interface**
Raspberry Pi 4 can send image data to the HPS/FPGA system by using Bluetooth Serial communication that utilizes RFCOMM protocol. NIOS II system can receive the image data successfully via a RS232 UART serial port.

4. **HPS/FPGA Computer System - Serverless Backend (AWS) Interface**
NIOS II system application can update an existing entry in DynamoDB with a new value of “poach” through an API Gateway endpoint using HTTP protocol.

5. **Machine Learning - Serverless Backend (AWS) Interface**
When a request with new sensor data is received at the API Gateway endpoint, a Lambda function that predicts the animal behaviour using a model stored in S3 bucket should be triggered.

6. **Machine Learning - HPS/FPGA Computer System Interface**
Human presence detection DNN parameters obtained from the trained model can be stored in the HPS/FPGA system application memory as constants on application start.

6. IMPLEMENTATION

A. Web Application

TAG web application was created with JavaScript and React which features 3 main pages:

- **Home Page**
The landing page where the user can see the aggregate information and a list of all registered animal TAGs.
- **Animal Page**
Detailed view of a specific animal TAG. This page can only be accessed through clicking an icon in the home page because the page requires the pet information to be passed as arguments.

- **Animal Creation Form**
  The animal registration page where users can register their animal TAG along with a picture of the animal. (*See Fig 6.3*)

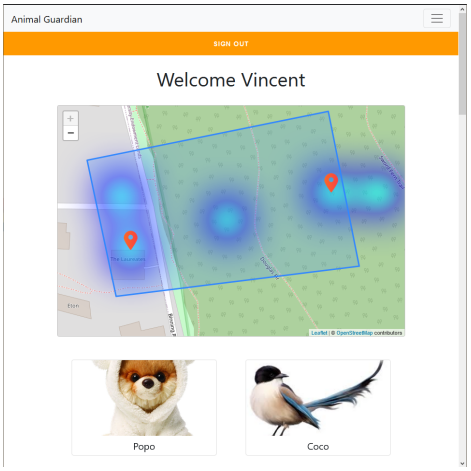

Fig 6.1 Home Page

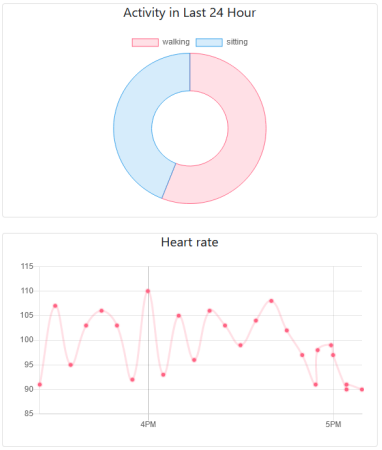

Fig 6.2 Graphs for sensor data

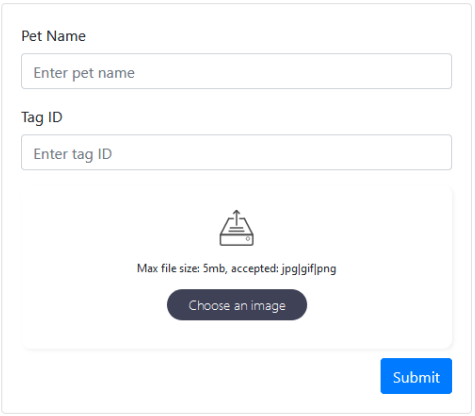

Fig 6.3 Pet Creation Form

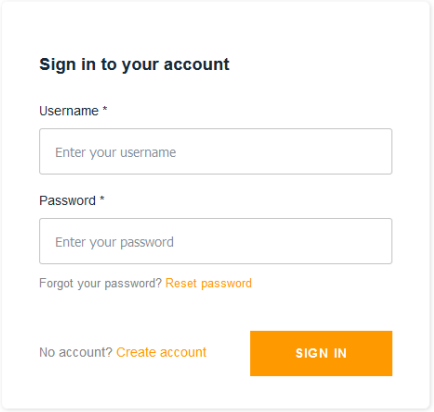

Fig 6.4 Login Page

The main components for the web application are as follows:

1. **Maps**
   The animal's location and geofence are displayed on a map widget. The main functionalities were implemented using the Leaflet libraries due to its ease of use for animal markers and map's background design implementation. An additional Leaflet Heat library was used to display the animal's location history with a heatmap. Finally, an additional implementation was added on the map using react and JavaScript to create the geofence editor.

2. **Graphs**
   The graphs were created using the Chart JavaScript library. The library provides built-in support for time-series data which was used to create accurate x-axis in the device's local time. (*See Fig 6.2)*

3. **User Login & Account Creation**
   The user login credentials were stored and managed by the Cognito User Pool. AWS Amplify default template was used for the login page and account creation page (*See* Fig 6.4.), where the users are automatically redirected to the login page if they were not logged in. User information can be fetched by the web application using the AWS Amplify SDK.

4. **State-management**
   All registered TAG(s) information of the current login user was fetched on first navigation to the home page and stored in a React State Hook of the application. A websocket connection was also established with the backend to listen for new data points and alerts. Upon receiving a new data point, the new data was appended to the old one, where the change then propagated to its child components.

5. **Navigation**
   The user can navigate between pages by using the navigation bar in the top of the web application. By using React Router, navigation loads only the new page resources but not the shared resources such as the AWS Amplify component and the navigation bar itself.

More image of the front-end website can be found in **Appendix F**

B. Serverless Backend (AWS)

The serverless framework was used to create and maintain the AWS CloudFormation stack which contains all the AWS resources required by our project. The serverless backend component implementations are as follow: (See Diagram 5.2)

1. **Website Hosting with Cloudfront**
   The front-end application was packaged in one set of static files downloaded by the user's browser at first URL access. Those static files were hosted on S3 and exposed through CloudFront, a content delivery network (CDN) service that delivers data to users globally with low latency and high transfer speed.

2. **Resource Access Endpoints**
   API Gateway was used to handle the HTTP connections and endpoint routes, synchronously triggering a Lambda function that fetches/pushes data from/to the database for each route. The websocket connection, image upload to S3 and animal activity prediction were also implemented using Lambda functions.

3. **Data Storage**
   A total of four DynamoDB tables were used in this project. See Table 6.1 for details of each DynamoDB table. On the other hand, S3 bucket was used for storing large image assets uploaded by users and one of our machine learning models to be used by our Lambda function.

Name	Status	Partition key	Sort key
notification	Active	username (String)	time (Number)
sensor-reading	Active	tagId (String)	readingId (String)
user-tag	Active	PK (String)	SK (String)
websocket-connection	Active	connectionId (String)	-

Table	Purpose
notification	Store all the past alerts sent to the web application by the backend
sensor-reading	Store all the sensor data uploaded by hardware devices
user-tag	Store user information alongside their pets' information
websocket-connection	Keep track of active websocket connections with our web app

Table 6.1 DynamoDB tables used

4. **User Pool and Authentication**
   Cognito User Pool was used in our application for authentication, user management, resource access control and external identity provider integration. It also provides flexibility to dispatch events and trigger Lambda functions, where we can use to customize the application authentication flow. One example of this, is our createUser Lambda function which is triggered when a new user is registered in our user pool. The function will then create a new user entry on the user-animal table, containing the user information needed for the web app.

5. **Real-time Updates for Web Application**
   We used WebSocket API provided in API Gateway to setup websocket connection with the frontend web app, used to provide real-time updates on sensor data. We also processed the new incoming sensor data to possibly generate alerts (wildfire alerts, poaching alerts), which would be sent together with the new sensor data.

6. **Hosting Machine Learning Model**
   When creating a new sensor data, before the data is stored in the table we passed in relevant data to our animal activity machine learning model in S3, to predict the current animal activity. The result was then appended to our sensor data and stored in a DynamoDB table.

C. Machine Learning

1. **Deep Neural network - Human Presence Detection**
   The human presence detection feature used camera information from the TAG to determine if there were any humans nearby. This feature was implemented using a combination of sensors data collection and machine learning training.

The neural network was implemented using the Python TensorFlow library, which uses *keras.Sequential* class object as the model. There were 8 layers in the neural network, with 1 flatten layer as the input layer, 3 hidden dense layers with 900 nodes each , 3 dropout layers with 50% dropout rate and 1 output layer with 2 nodes. The hidden layers use the ReLu activation function while the output layer uses the softmax activation function. The learning rate was tuned first, then came up with a LR of 0.004 with the adam optimizer that takes the cross entropy loss. We also added dropout layers and l2 regularizers just to prevent the model from overfitting. This model was trained through the "fit" function over 40 epochs and at a batch size 64. After training the model, the weights and biases of the model were then saved into a file for the later use in FPGA.

2. **Decision Tree - Animal Activities Prediction**
   The animal behaviour prediction feature predicts the animal's current behaviour by using the heart rate sensors and accelerometer sensors and a Decision Tree machine learning model.

   The model was implemented using the Python Sklearn library, where an object of class *DecisionTreeClassifier* in *Sklearn.tree* was used to represent the model. The training of the model was pretty simple, where we splitted the data into training and testing data into an 80/20 split, and called the "fit" function to train the model. The model was also given an entropy criterion to make the splitting more accurate and to make the tree less deep, which reduces computational costs. The model will then be pickled and sent to the database through S3 Bucket.

D. Raspberry Pi 4

1. **Sensors:** To collect data required by the TAG, the Raspberry Pi was equipped with the following:
   - **Pulse Sensor**: The LED on the front side of the sensor is placed over a vein, the LED emits light which falls on the vein directly and measures blood flow to monitor heartbeat. The pulse sensor can not be read out digitally since the Pi has no integrated analog IO pins, therefore we use a MCP3008 analog-to-digital converter to read out analog signals on the Raspberry Pi from the Pulse sensor.
   - **DHT-11 Temperature & Humidity sensor**: Measures temperature with a surface mounted NTC temperature sensor (thermistor) built into the unit. Temperature accuracy: ±2% °C.
   - **ADXL 345 Accelerometer**: 3-axis sensor interfaced using an I2C communication, measures static and dynamic acceleration in x,y,z axes.
   - **PiCam**: Uses piCamera library to capture images of surroundings in jpg format.
   - **GPS Neo-7m**: Highly sensitive, low power GPS that outputs precise position updates at 10Hz with 0.6 meters accuracy.

2. **Bluetooth:** Captured image data is sent to the De1-SoC HPS system over serial communication following the RFCOMM protocol. The receiving end uses a HC-05 BT module included in the RFS daughter board.

3. **Wi-Fi:** Everything else is sent to the AWS API Gateway through the built-in RaspberryPi wifi component. This sensor data is included in a HTTP POST request where the tag id will be passed on the URL and the data will be placed in the message body in JSON format.

Using the Raspberry Pi is a proof of concept and it is ideal to use a custom ASIC chip in the final product to minimize costs.

E. HPS/FPGA Computer System

This subsystem provides the human presence detection feature by performing the following tasks: *(See Diagram 5.3)*

1. Receive 28×50-pixel image data from Raspberry Pi 4 via Bluetooth
2. Perform hardware-accelerated dot product calculation for a pre-trained DNN with received image data
3. Send human presence detection DNN result to cloud database for storage via Wi-Fi

in an application that runs on a custom HPS/FPGA system on DE1-SoC board, which included the use of a ARM A9 HPS and a NIOS II softcore processor.

Tasks 1 and 3 were implemented on the FPGA system, which consists of the following main components:

1. NIOS II processor
2. JTAG UART
3. SDRAM controller
4. 2 × RS232 UART IP components
5. FPGA-to-HPS DMA Controller

The RS232 UART serial communication port pins were brought out to the GPIO pins on the DE1-SoC board, to connect with the Bluetooth and Wi-Fi modules on the RFS board. With successful serial communication between the HC-05 module and FPGA system, 28×50-pixel image data was received from the connected Raspberry Pi computer and stored in SDRAM. After the receival of 1400 pixels and 1 database ID from Raspberry Pi, a write operation was performed on the FPGA-to-HPS DMA controller's control port slave address, to copy the database ID received earlier to the DNN accelerator address in HPS, which in order triggers the hardware-accelerated task immediately without involving the CPU. The NIOS II system then waits for the human presence detection result by constantly reading from an address within SDRAM. Once the value stored in that address becomes the same as the database ID, a function which was loaded into the application on application start, via a lua script, is called to send the result, which is stored in (address + 1) to the cloud database via Wi-Fi.

Task 2 was implemented on the HPS system, which consists of the following main components:

1. Custom DNN accelerator with one Avalon master interface (memory-facing) and one Avalon slave interface (CPU-facing)
2. HPS to FPGA DMA controller
3. JTAG to FPGA Bridge

Once the FPGA-to-HPS DMA controller from the previous part wrote to the DNN accelerator address, the DNN accelerator started the dot product calculation using the weights and input data read from SDRAM. After calculation, the database ID written on to the DNN accelerator from the previous step was then copied to the result address that the NIOS II system was constantly reading from. Finally, the JTAG to FPGA bridge allowed the access to the addresses of the system components from our C code and performed operations on them.

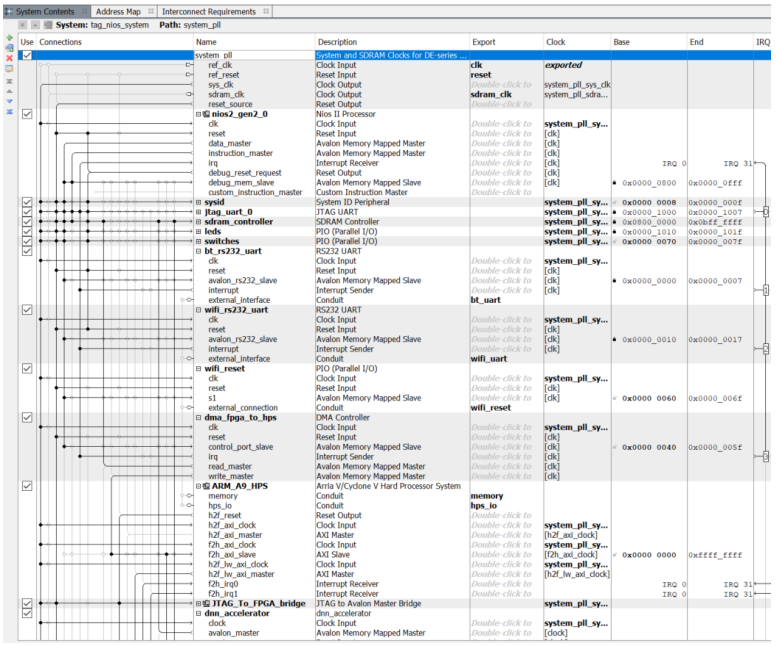

Fig 6.5 HPS/FPGA system in Qsys

7. VERIFICATION AND VALIDATION

The verification process for the prototype starts with running unit tests on each component to ensure each individual component within a subsystem works as intended, followed by a validation process for each subsystem, finally testing and validating the subsystem interfaces to verify the functionalities of the prototype as a whole.

A. Hardware Testing

Well-known and reliable libraries were used to handle sensor readings and data collection on Raspberry Pi 4 computer. Hence, we decided that intensive unit testing on sensors integration is not required and simple manual testing (eg. compare ambient temperature data read by DHT-11 with the room temperature obtained from a mobile app) was sufficient. To verify the functionality of Raspberry Pi 4 and serverless backend interface, a DynamoDB table data injection test script was created.

A unit testing application was created for each main component in the HPS/FPGA system:

1. RS232 UART component for serial communication with HC-05 module in NIOS II system
2. RS232 UART component for serial communication with Wi-Fi module in NIOS II system
3. Custom DNN accelerator component in HPS

And another application was written to verify the functionality of the subsystem as a whole.

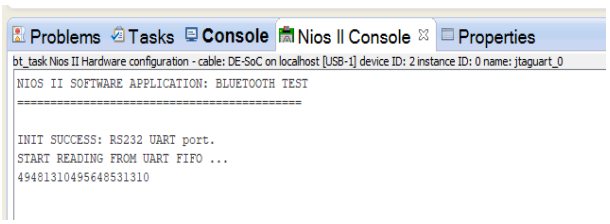

Fig 7.1 BT Unit Test in NIOS II Eclipse

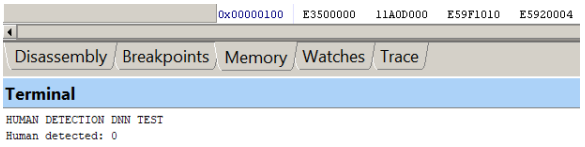

Fig 7.2 HW-accelerated DNN testing in Monitor Program

B. Software Testing

For the front-end of the website, the testing was done by running the code for every feature when integrating it with the back-end of the website. Moreover error handling was implemented and tested to handle cases where the back-end returns incorrect responses.

For the back-end, there are unit tests for the commonly used code (can be seen in the __tests__ folder). For integration testing, we have created a script that can create randomly generated sensor data and send it to our API. Through this, we are able to verify our whole backend system working.

For the machine learning scripts, there were numerous testing done for both the decision tree and the neural network model. The decision tree model was tested by using the testing dataset obtained through the built-in Sklearn train_test_split function. The “predict” function is used to predict the output of the model and is then compared to the expected outputs of the model, and a function will then compute the accuracy score of the model. We generated the model for 5 times, and it has a consistent average mean of 0.80 accuracy.

```
Accuracy: 0.8125
Y_test expected:  314    abnormal
394    abnormal
375    abnormal
267     walking
367    abnormal
         ...
96     sleeping
311    abnormal
252     walking
167     sitting
128     sitting
Name: Behavior, Length: 80, dtype: object
Y predicted values:  ['abnormal' 'abnormal' 'abnormal' 'walking' 'abnormal' 'walking' 'walking'
 'abnormal' 'abnormal' 'abnormal' 'sleeping' 'sleeping' 'walking'
 'walking' 'abnormal' 'sleeping' 'sitting' 'sitting' 'sleeping' 'sitting'
 'walking' 'walking' 'sitting' 'sitting' 'sitting' 'sleeping' 'abnormal'
 'sitting' 'abnormal' 'sleeping' 'walking' 'sitting' 'walking' 'abnormal'
 'walking' 'abnormal' 'sitting' 'walking' 'abnormal' 'sitting' 'abnormal'
 'sleeping' 'sitting' 'sleeping' 'walking' 'walking' 'abnormal' 'walking'
 'sitting' 'abnormal' 'sleeping' 'sitting' 'sleeping' 'abnormal'
 'abnormal' 'sitting' 'sitting' 'abnormal' 'sleeping' 'sleeping'
 'abnormal' 'sleeping' 'abnormal' 'walking' 'abnormal' 'abnormal'
 'abnormal' 'walking' 'sleeping' 'walking' 'sleeping' 'sitting' 'walking'
 'sleeping' 'sleeping' 'sleeping' 'abnormal' 'walking' 'sitting' 'sitting']
```

Fig 7.3 Decision Tree Model Accuracy Score

The neural network was tested in the same way by using the testing dataset obtained through the train_test_split function in Sklearn, just like the decision tree model. However, the neural network model has more repetitions of testing due to the complex nature of hyperparameter tuning. After getting the right hyperparameters, a function was also created to compute the

accuracy score between the expected output and the predicted output, which in our model's case, it averages an accuracy of about 0.70 over 5 runs.

Overall, the prototype was also tested as a whole by attaching the TAG onto a human. We validated the product by checking all the features specified: wildfire detection alert, geofencing, human presence detection and animal behaviour prediction. The TAG was attached to a person and was exposed to particular conditions such as high temperature to make sure that a certain feature gets alerted, which is the wildfire detection alert in this case. The validations can be further seen in the project demo video.

8. REFERENCES

---

[1] Cornell ECE5760 - DE1-SoC: ARM HPS and FPGA Addresses and Communication
[2] SoC-FPGA Design Guide [DE1-SoC Edition]
[3] DE1-SoC User Manual
[4] FPGA-to-HPS bridges Design Example
[5] CPEN311 Digital Systems Design, Lab 4, Department of ECE, UBC, 2020
[6] Youtube tutorials on the Serverless Framework with AWS
[7] Serverless Framework Documentation
[8] React Documentation
[9] Leaflet Map Library
[10] Leaflet Heat Plugin
[11] Chart Js
[12] Tensorflow Sequential Class
[13] Sklearn DecisionTreeClassifier class
[14] Learning rate configuration

9. Links

---

[1] Website link

You can log in with a credential filled with dummy data:

- Username: Vincent
- Password: Passw0rd!

[2] Source code on Github

10. INDIVIDUAL APPENDICES

APPENDIX A. Andrew Shieh

Individual Contribution

1. Using the Serverless Framework, setup and maintain all the AWS resources used.
2. Wrote the code for all the Lambda functions used.
3. Implemented all the APIs used.
4. Wrote the code for the animal creation form in the web app, in collaboration with Vincent.
5. Collaborated with Vincent to implement authentication in our web app.

Team Effectiveness

Overall, our team is pretty effective at achieving our goals. Constant progress was made all the time, and communication was effective and clear, leading to ease when integrating our subsystem. All of us were not afraid of asking for help and giving them, leading to a collaborative environment.

Other Comments

I made sure that my tasks integrate with the others by communicating with them and setting up mini-demos, where we would test the APIs that I have made that work well with other parts of the system. Before starting on the project, I had zero experience working with AWS which made it difficult at first to learn and build the backend at the same time. Thankfully, there are plenty of online resources which helped me go through. I always test out my APIs, either with my own dummy data or with actual data fed by the hardware or web app. This allows me to be sure that the backend works and that we will show success during the demo. I learned a lot about cloud computing and more specifically AWS cloud architecture. Due to the nature of my tasks, I also learned to collaborate with my teammates setting up tests and demos together to make sure everything is connected properly. I am also in charge of editing most of our pre-recorded demo video (other things I spent time in).

APPENDIX B. Aayush Bisht

Individual Contribution

Working with the RPI subsystem of the tag and respective interfaces that allow the pi to interact with other subsystems

1. GPS module*****
   - NEO-7M GPS module to provide location of the animal for tracking purposes.
2. 3-axis accelerometer
   - ADXL345 module used to track movement of the animal.
3. Camera Module*****
   - Using RPI Camera module to take pictures near animal wearing the TAG.
4. Bluetooth Module*****
   - Send data todFPGA over Bluetooth connection.

5. Wi-Fi Module
   - Used to send data to our database which is displayed on our web app.
6. Heartbeat/Pulse Sensor
   - Pulse & Heart Rate sensor to measure heart rate of the animal.
7. Temperature Sensor
   - DHT-11 temperature and humidity sensor to monitor surrounding temperature and predict wildfire proximity.
8. (Collaboration with Shade Wong) Working on RFS wifi ESP worm*****
   - Writing lua script to interact with the database.

*Marked ***** indicates tasks required extra effort because of issues mentioned in design constraint and general difficulty of the task.*

Team Effectiveness

Team was effective and asked for help in time when needed. Very helpful environment in the team and effective communication. Scrum meetings were effective where mostly everyone was punctual and on time.

Other Comments

To ensure that my tasks would integrate with the rest, I did research before putting up a proposal. Most integration was done over well tested serial routes and protocols.
Some hurdles I faced include weird errors with the pi, where it would show red light indication right amount of power being supplied but would not work properly with the sensors in my house. To improve the likelihood of success, I asked for help when needed and received appropriate help and also offered help to others in a timely manner. During this project I learned to better communicate with the team and tackle problems as a team rather than all by myself.

APPENDIX C. Elbert Ng

Individual Contribution

1. Did all the training and testing for the neural network
2. Sampled the image dataset for the neural network
3. Did all the training and testing for the decision tree
4. Collaborated with Aayush as he sent me the data points to make up the dataset for the decision tree
5. Collaborated with Andrew to import the model to the AWS back-end
6. Collaborated with Shade to save the weights and biases in the FPGA

Team Effectiveness

I do believe that our team was very effective in integrating the design as a whole. We communicated well and were able to complete more tasks, especially testing and validating the prototype as a whole, more than in module 1. We always have 2 scrum meetings per week to keep track of everyone's progress, and we all always openly state whenever we meet any roadblocks so we can help each other.

Other Comments

I ensured that my tasks would integrate by constantly communicating with my team, and have a short meeting whenever we need to share our code with each other. We also always test it after successfully integrating the tasks, to ensure that it's working perfectly. Most of the hurdles that I have to overcome is when implementing the neural network model. It took days to tune the hyperparameters of the neural network, and it also took a lot of time to sample and re-sample the image dataset. I also always research more on the machine learning algorithms whenever I am unsure, and read the tensorflow and sklearn libraries for its documentation to ensure that I fully understand what code I am writing. I also watched a lot of YouTube videos on neural networks as I am new to the branch of Machine Learning. I learned a lot from this project, especially in the field of Machine Learning. Before I did the project, I had almost near zero experience in Machine Learning, and now I learned how neural networks work and how decision trees work, which are 2 of the most popular Machine Learning algorithms out there. Luckily for us, there were no issues with the team as a whole.

I ended up spending a lot of time researching machine learning algorithms and got really interested in it, especially deep neural networks. I learned about neural networks more in depth such as the different activation functions (Sigmoid, Tanh, ReLu, Softmax) and the techniques used to create a neural network model (Backpropagation and forward feedback)

APPENDIX D. Shade Wong

Individual Contribution

1. Completed Exercise 1.3, 1.9, 1.9.1 and 1.9.2 individually
2. Designed and built HPS/FPGA computer system that conforms with the subsystem specifications from scratch *
3. Implemented the custom DNN accelerator component on HPS
4. Completed Wi-Fi communication between HPS/FPGA system and DynamoDB in conjunction with Aayush *
5. Setup Bluetooth slave communication mode on FPGA system to receive image from Raspberry Pi *
6. Implemented unit tests for each main component in the HPS/FPGA system using NIOS II Eclipse Software, SoC EDS, and Altera Monitor Program *

** Tasks that required especially large amount of effort.*

Team Effectiveness

An extensive amount of planning was carried out during each SCRUM meeting in the first month, which contributed to the team effectiveness in the later phase of the development process. With a well-defined set of specifications for each subsystem, the research process for each team member was a lot smoother. Overall, the team was effective in implementing and testing each subsystem as well as integrating all the parts as a whole.

Other Comments

To ensure that my tasks/subsystem would integrate with the group's, I communicated with other group members frequently to discuss the components that overlapped between two or more subsystems, and ensure that the component design would work conveniently for all the subsystems involved. In addition, quick manual testing was carried out for every collaboration before writing a more comprehensive test application to ensure that the system works at the very least. Since I do not have much exposure to building a FPGA/HPS system using Qsys previously, building the entire system, including the use of RS232 serial communication port with external chips, and interfacing with the hardware components using addresses defined in Qsys within my C code was challenging. To ensure the success of the subsystem implementation, I started looking into tutorials and exercises on Canvas in a very early stage and did extensive research on methods to integrate the BT and Wi-Fi chip on a HPS system online. However, though the RS232 serial communication port was implemented successfully, I cannot seem to get it to work with the HC-05 module. Hence I went for the HPS/FPGA system for the HAL libraries that can be used in NIOS II-based system application. I would say this was indeed a very challenging but enjoyable project for me and I learnt a lot about the communication between HPS and FPGA components, as well as interacting with the components within a custom system through C code.

APPENDIX E. Vincent Sastra

Individual Contribution

1. Created front-end web application with React & TypeScript
2. Style the website using Bootstrap components
3. Implemented maps and its geofence editor
4. Implemented state management for web application
5. Deployed the website build to a S3 Bucket and setup the CloudFront service
6. Collaborated with Andrew to handle web application and serverless backend communication using HTTP and websockets

Team Effectiveness

Our team is effective at communicating and designing the interfaces surrounding their subsystem. The project is divided equally to 5 subsystems that are very modular and thus, the project's integration went very smoothly.

Other Comments

To integrate my task with Andrew's backend subsystem, we created a Postman account to provide detailed documentation of our API and communication protocol. Moreover, we often do integration testing (around once or twice a week) to make sure that the implementation communicates correctly. Furthermore, I also have the AWS credentials to see the raw tables & lambda functions to develop my understanding of the backend subsystem.

Most of my hurdles come from integrating different front-end libraries that do not directly support each other. The React framework has a very strict set of rules and making sure that React is compatible with multiple front-end features and components can be difficult. For

example, one of the most prominent bugs comes from the browser event handler not being correctly cleaned up after React deallocates an object. This is finally fixed by manually removing every event handler before the object deallocation which requires strong React framework understanding.

I learned a lot about the AWS ecosystem and front-end React development. After this project, I feel confident with my ability in developing React applications that have a complicated state management system. Furthermore, integrating my website with AWS and deploying it to CloudFront taught me how to use the serverless ecosystem as well as its strengths and weaknesses.

During this project, I spent a considerable amount of time learning about front-end and website design. I wanted to create a website that is aesthetically pleasing and intuitive for the user. In the end, I am proud of my work and its simple and clean final design.

APPENDIX F. Image of User Interface

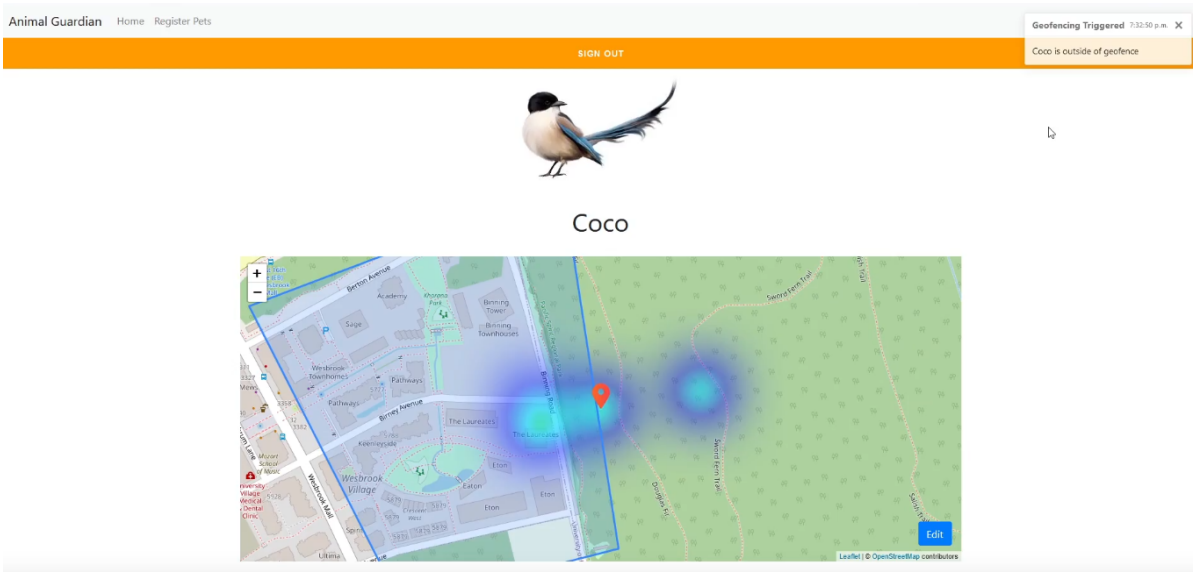

Fig 10.1. Home page with Notification triggered

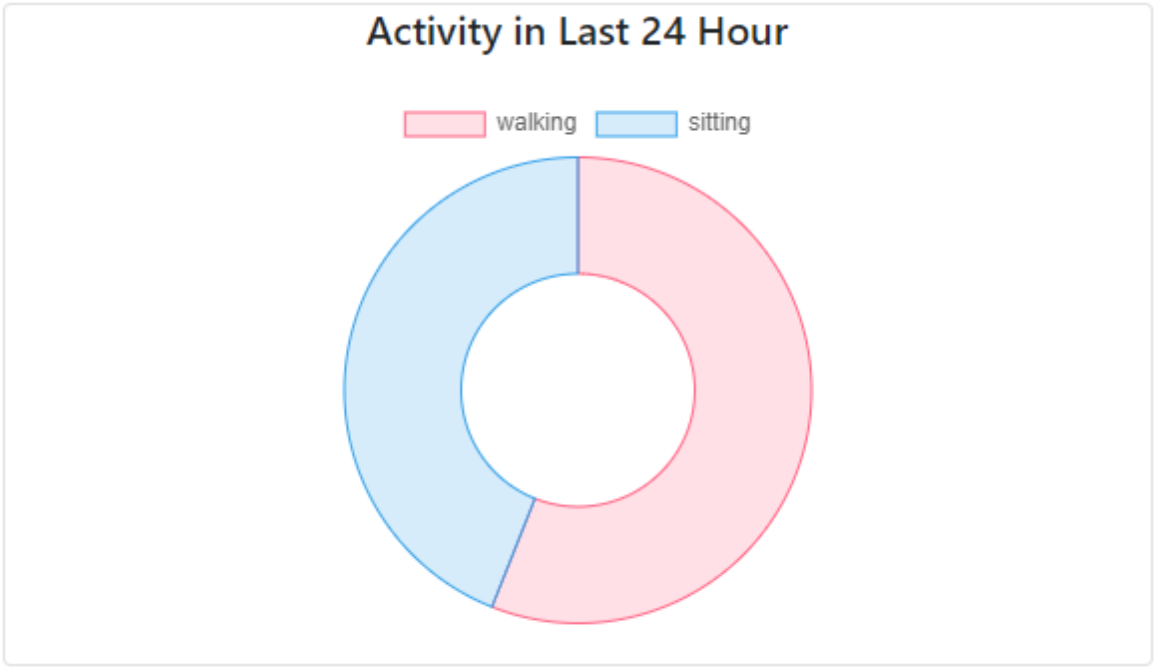

Fig 10.2. Activity Doughnut. A concise way to display animal's past activity

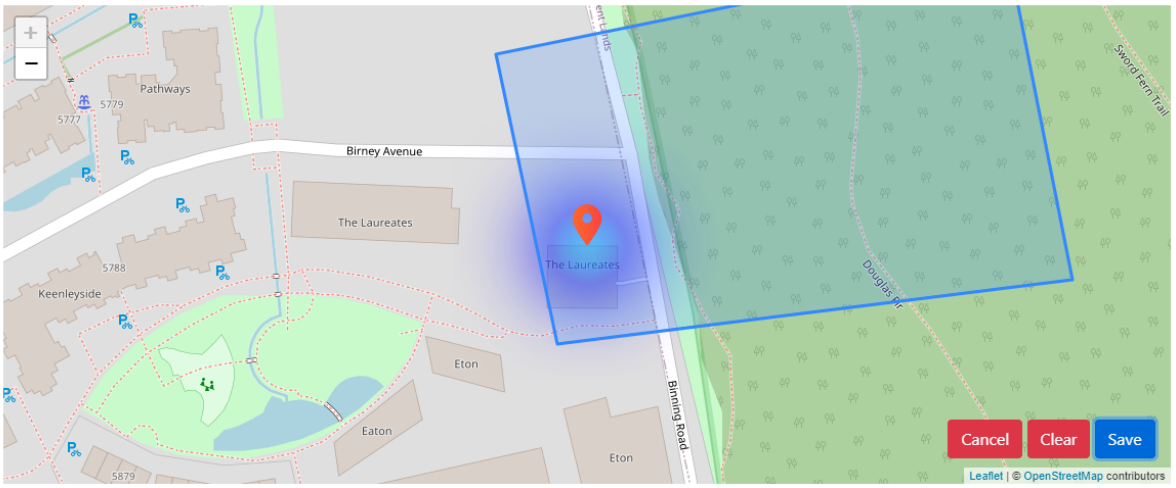

Fig 9.3. Maps when the Geofence is being edited

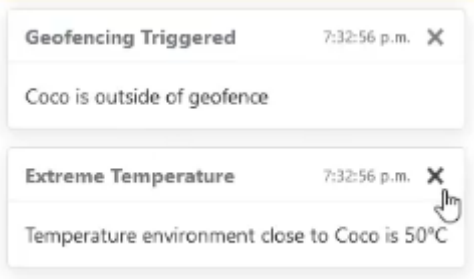

Fig 10.4. Close up of the notifications

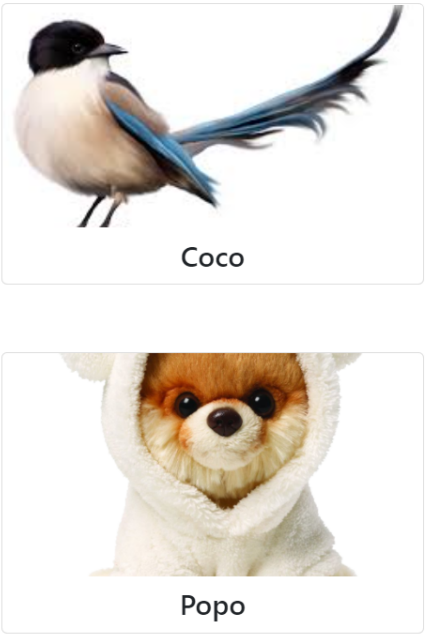

Fig 10.5. Pet card automatically use new rows on smaller screen

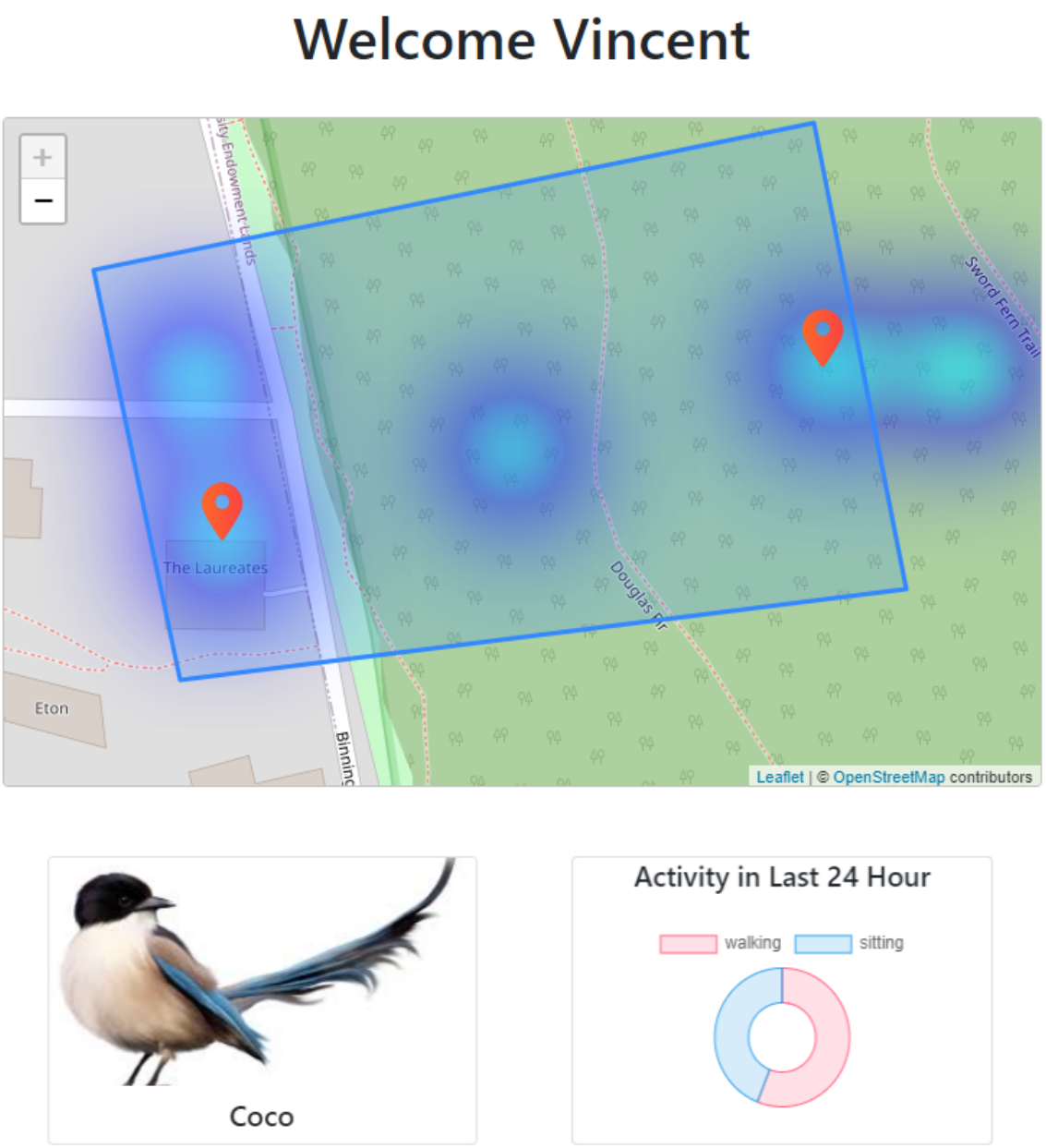

Fig 10.6. Home Page with one of the pet's card flipped

Fig 10.7. Navigation Bar minimized using the burger button for smaller screen