Group 5 – FindFit

19th November 2021

Agenda

01

Introduction

Problem Statement, Brief Description of Solution

02

Details of Approach

IOT Architecture, Model

03

Implementations & Demo

Sensor Management, Demo

04

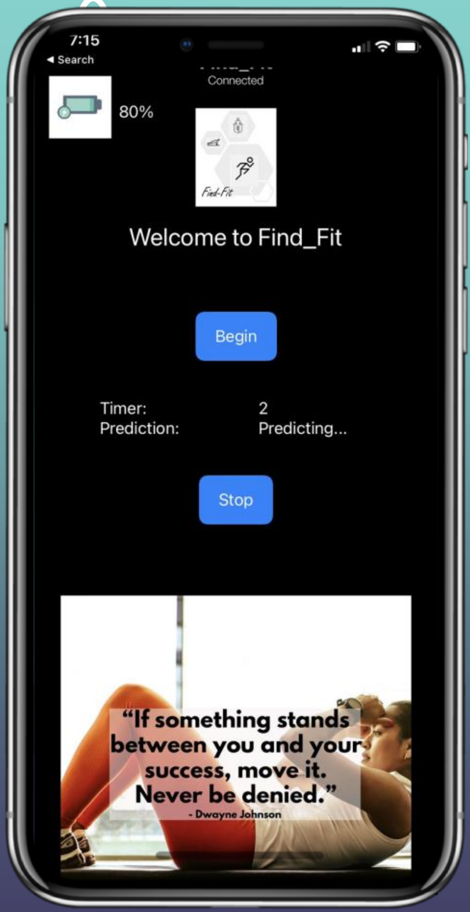
Challenges & Limitations

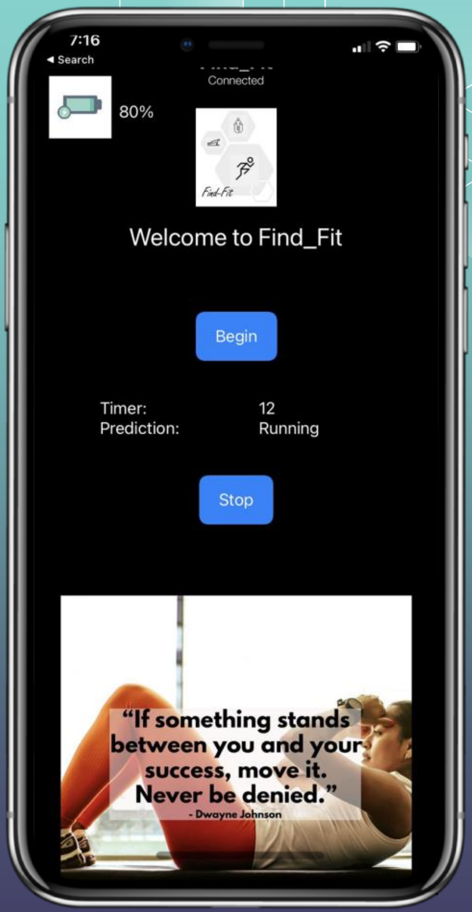
Challenges faced, takeaways

I. Introduction

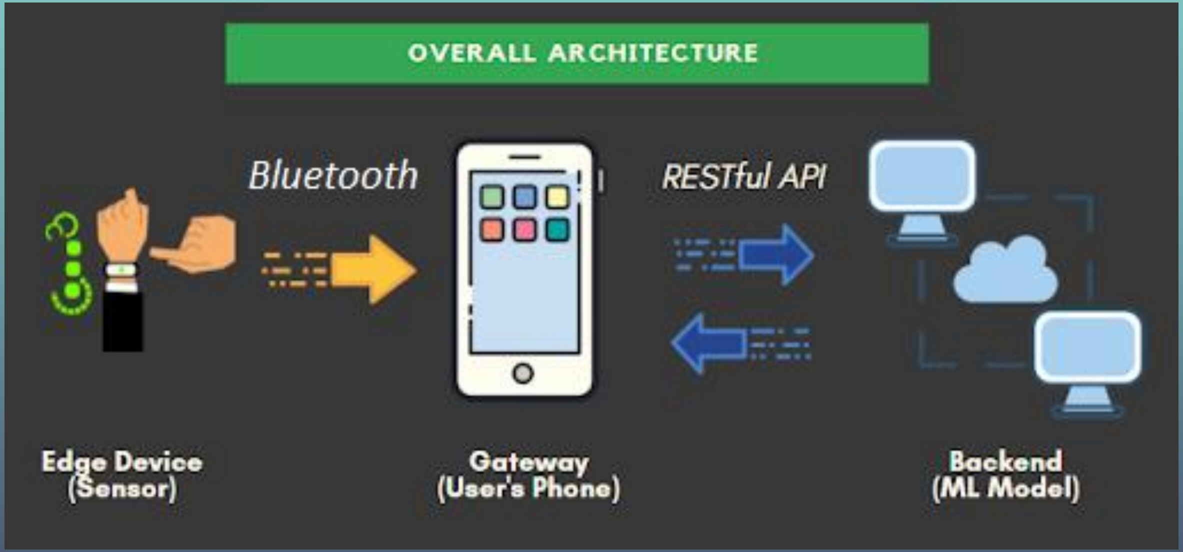
FindFit

- Simple Mobile Application
- Predicts exercise type
- Updates with exercise changes



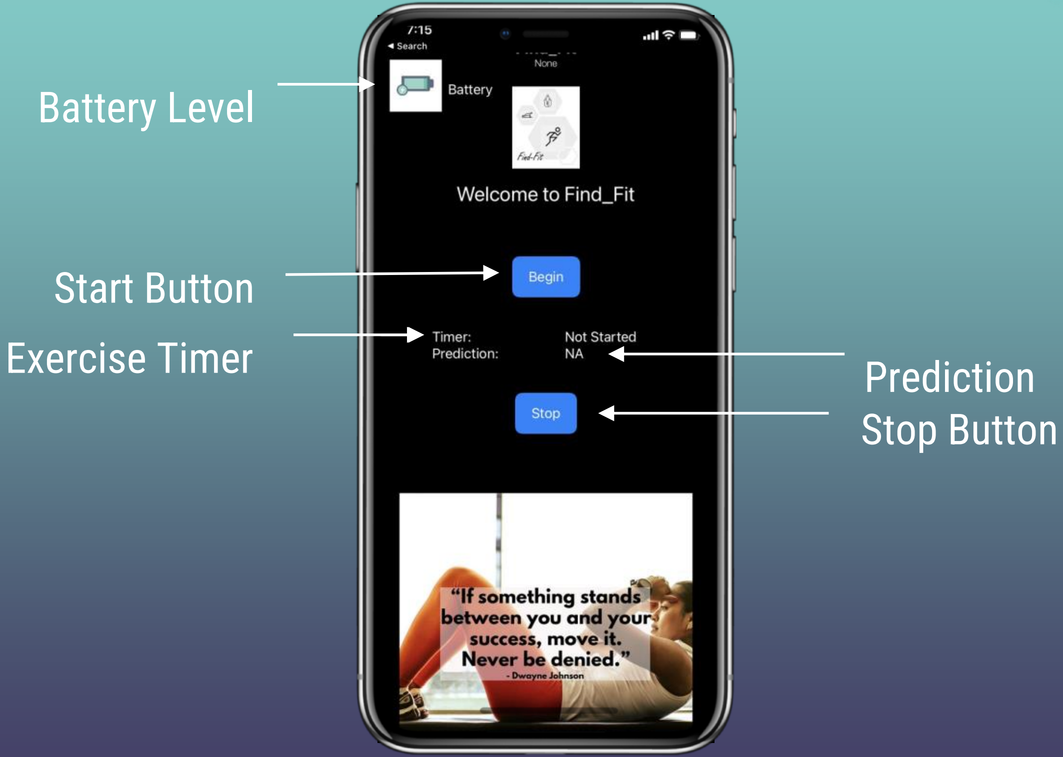


Solution Overview

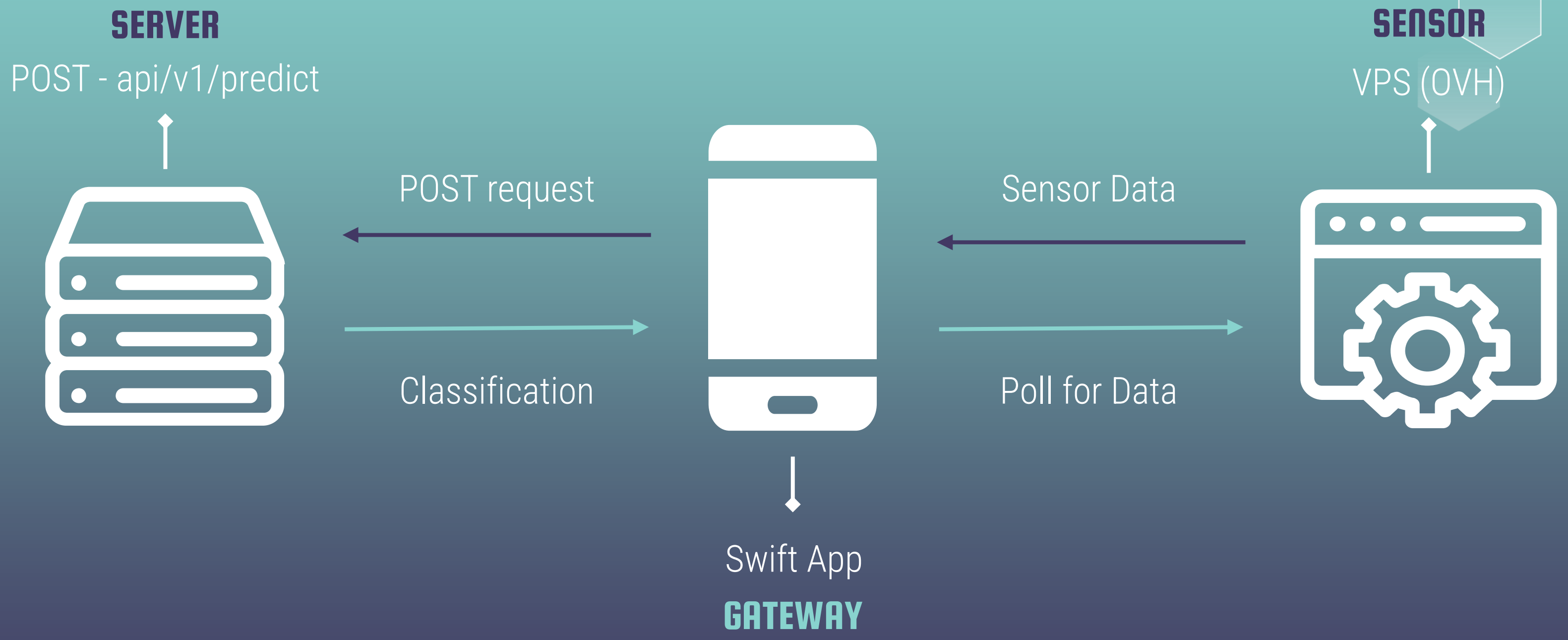


2. Details of the approach including IoT architecture, ML model

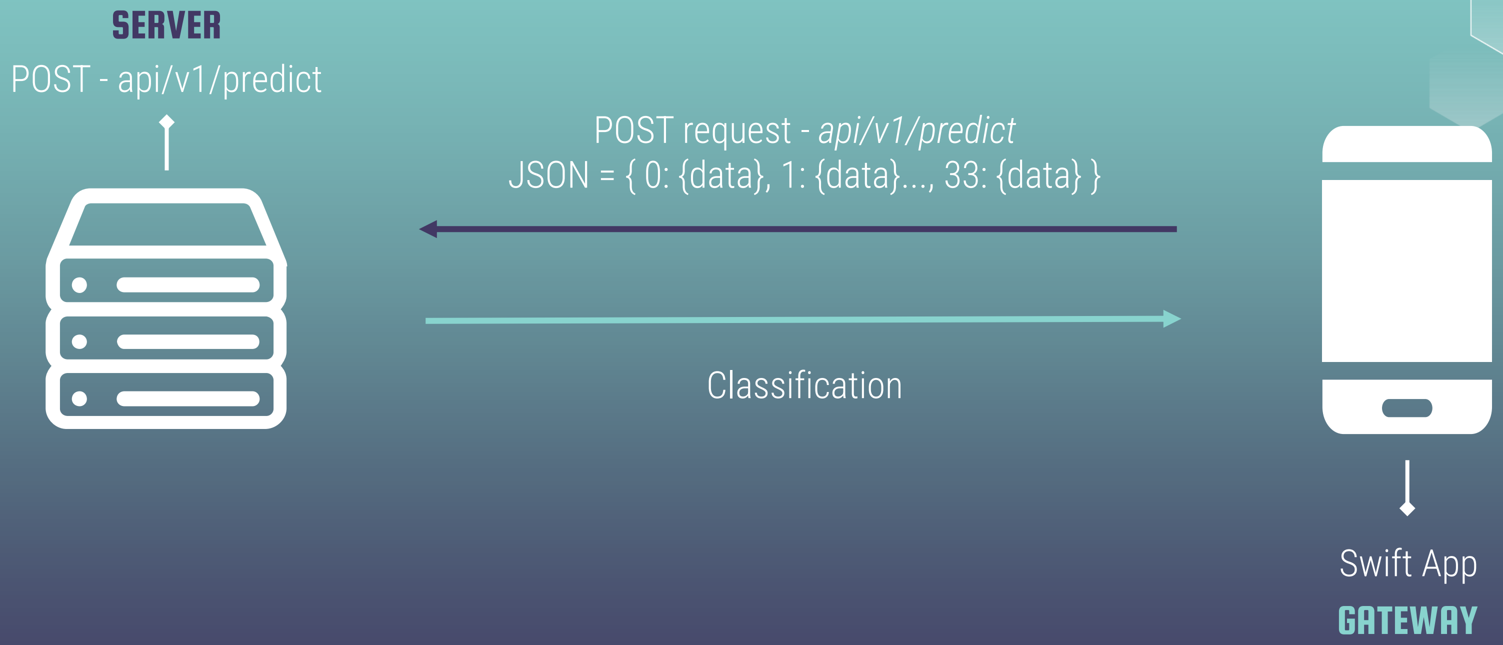
Client


Battery Level
Start Button
Exercise Timer
Prediction
Stop Button
7:15
◂ Search
None
Battery
Welcome to Find_Fit
Begin
Timer:
Not Started
Prediction:
NA
Stop
“If something stands between you and your success, move it. Never be denied.”
- Dwayne Johnson

Backend Server



Backend Server



Backend Server

SERVER

POST - api/v1/predict

1. POST request received by - *api/v1/predict*
   JSON = { 0: {data}, 1: {data}..., 33: {data} }
2. Convert the JSON data into a 1x1x33 numpy array
3. Pass the converted data into the ML Model to obtain the classification.
4. Return the response as a JSON

ML Model

Data Manipulation

To sanitize and augment data

Data Collection

- Self-collected via a Python Script
- Stored in a PostgreSQL database

Model Experimentation

- MLP
- LSTM
- CNN

Data Collection

1

Sensor mounted on the wrist

3.33hz

Sampling Frequency

- Non-standardized recording length
- Unique Session IDs
- Stored in PostgreSQL Database

Data Manipulation

1. For each time-series, the initial and end were trimmed to account for movement recorded to begin and stop the recording process.
2. Rows containing zero-values were dropped as they represent periods of idleness.
3. A rolling window of ~10s was then applied to the time-series to synthetically generate more data for training.

Other Possible Augmentations Considered:

- Step-Wise Windows
- Hamming Windows
- Time-series Elongation/Compression
- Min-Max Scaling

Model Experimentation

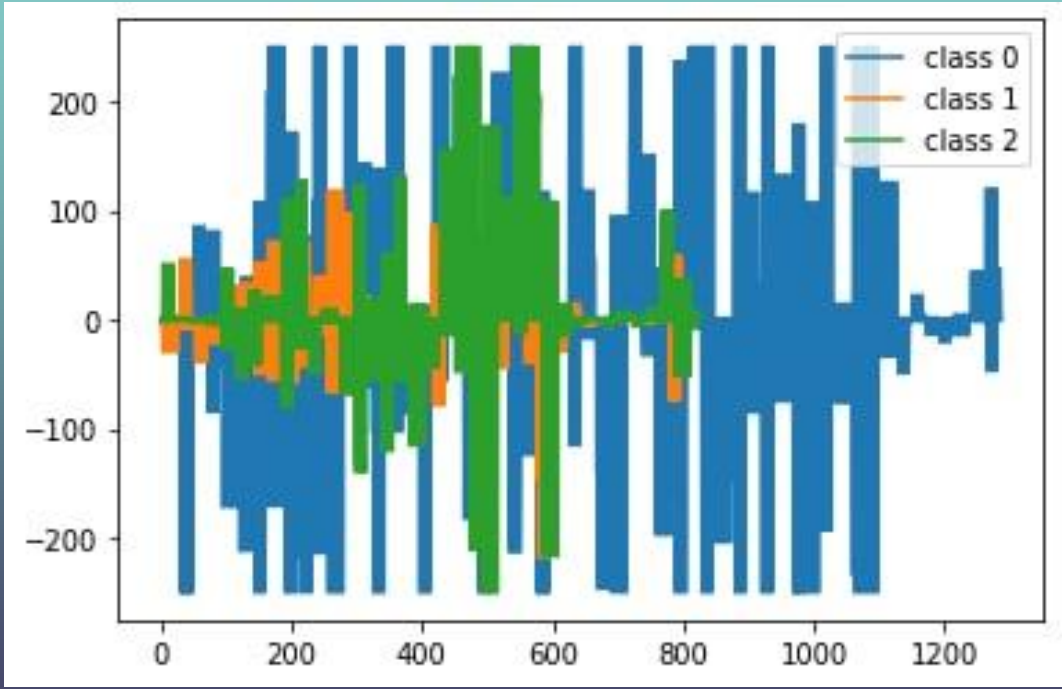
LSTM
Accuracy: 63%

MLP
Accuracy: 57%

PCA + MLP
Accuracy: 90%

CNN
Accuracy: 100%

Model Experimentation



3. Implementation Details and Demo

Implementation Details

2

Sensors on

3.33Hz

Sampling frequency

Core Bluetooth, Restful API and Flask

Communication protocol

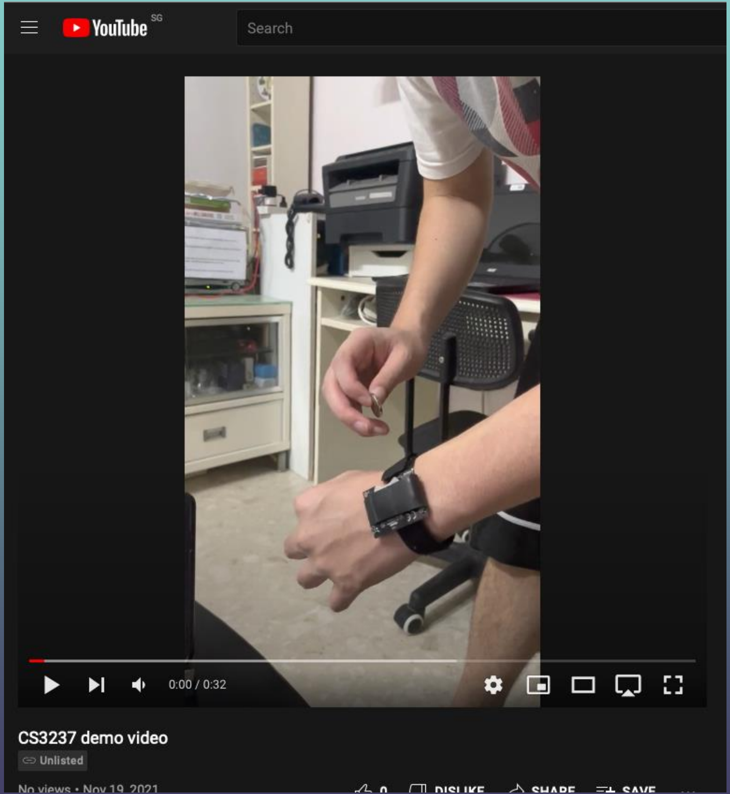
Power Management

Controlling the sensors to be turned on ->
Controlled via Swift UI

```
if SensorTag.validConfigCharacteristic(characteristic: thisCharacteristic) {
    // Enable Sensor
    var enableValue = thisCharacteristic.uuid == MovementConfigUUID ? 0x7f : 1
    let enablyBytes = NSData(bytes: &enableValue, length: thisCharacteristic.uuid == MovementConfigUUID ? MemoryLayout<UInt16>.size :
        MemoryLayout<UInt8>.size)
    self.sensorTagPeripheral.writeValue(enablyBytes as Data, for: thisCharacteristic, type: CBCharacteristicWriteType.withResponse)
}
// print(characteristic.uuid)
```

THE DEMO

Video (In case real life didn’t work)



Link: https://youtu.be/XFoTEagM7gg

5. Summary including challenges faced, limitations of the solution, possible future directions etc.

Challenges faced

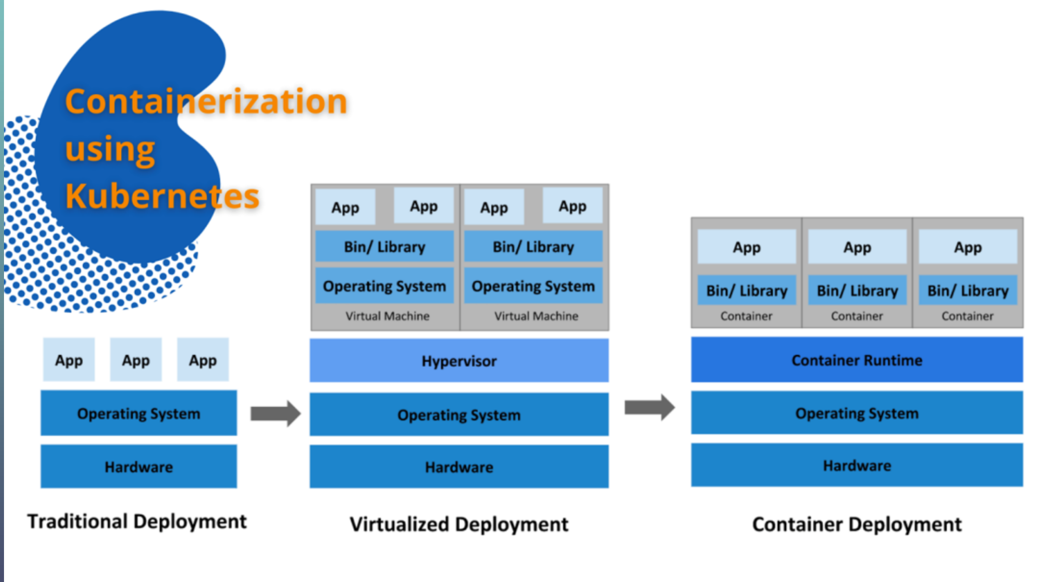
- Default RTOS project does not come with BLE stack
- Tried to run customised firmware using BLE stack
  - BLE no native support for non-Windows users
  - Different versions of compiler required
  - Lack of solid documentation
  - Lack of examples
- Unable to optimize sensortag well with SWIFT client
  - Power policies exist in RTOS library but not available in SWIFT
  - Default firmware uses polling
  - Unable to optimize idling/sleeping of device

Limitations of solution

- Lack of cross platform support
- Expensive API call
- Setup not ready for production

Possible future directions

- Implement code on lower level using RTOS
- Abstract layer for different clients to consume
- Websocket for prediction
- Production ready server



THANK YOU

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik

Please keep this slide for attribution