HomeDetect: Light-Switch IoT

Mini Project Report

Nick Nesterenko

CMP408: IoT and Cloud Secure Development

BSc (Hons) Ethical Hacking, Year 4

2022/23

Note that Information contained in this document is for educational purposes.

Contents

1 INTRODUCTION

1.1 WHAT IS HOMEDETECT?

The HomeDetect project represents a comprehensive implementation of an iOS application and IoT device, utilizing Amazon Web Services (AWS) as the cloud technology. The project aimed to create a smart home monitoring system that enhances security and convenience for the user to prove several concepts. To achieve this, the iOS application, developed using AWS Amplify, communicated with the Raspberry Pi (RPi) IoT device via AWS IoT Core, AWS Cognito, AWS Lambda, and AWS API Gateway.

The objective of the project was to implement a mobile application that would communicate the device location to AWS, which would then calculate the distance between the user and the predefined location of the IoT device (the "home" location). Based on the results, if the user was within 100 meters from home, the RPi IoT would automatically control the lights in the room via micro servo connected to a light switch.

The use of AWS as the cloud technology and Raspberry Pi as the IoT device allowed for secure and efficient communication between the mobile application and the RPi, thus making the system both user-friendly and secure.

The HomeDetect project is not only an innovative solution for smart home monitoring, but also highlights the importance of secure development in the IoT and Cloud technology fields. With the increasing popularity and use of IoT devices and cloud technology, the need for secure communication and data storage has become paramount. The use of AWS services such as AWS IoT Core, AWS Cognito, AWS Lambda, and AWS API Gateway in the HomeDetect project demonstrate how these technologies can be effectively utilised to create a secure system that not only enhances the user experience but also protects personal data.

The following report covers the overview of used services and techniques to ensure the overall security of the project, core functionality, implementation of Linux Loadable Kernel Modules, and why those implementations were chosen in terms of cybersecurity and data integrity.

1.2 AIMS/OBJECTIVES

The HomeDetect project aimed to:

- To create a smart home monitoring system that enhances security and convenience for the user.
- To develop an iOS application using AWS Amplify, communicate with the Raspberry Pi (RPi) IoT device via AWS IoT Core, AWS Cognito, AWS Lambda, and AWS API Gateway.
- To implement a mobile application that would communicate the device location to AWS, which would then calculate the distance between the user and the predefined location of the IoT device (the "home" location).
- To use RPi IoT would automatically control the lights in the room via micro servo connected to a light switch if the user was within 100 meters from home
- To use AWS as the cloud technology and Raspberry Pi as the IoT device to allow for secure and efficient communication between the mobile application and the RPi
- To create a secure system that enhances the user experience and protects personal data
- To use the latest security best practices and guidelines by using Swift 5 and SwiftUI framework in the development of the iOS application

- To use AWS services such as AWS IoT Core, AWS Cognito, AWS Lambda, and AWS API Gateway in the project to demonstrate how these technologies can be effectively utilized to create a secure system
- To use Linux Loadable Kernel Modules(LKM) to prove the concept of LKM,

2 PROCEDURE AND RESULTS

2.1 CLIENT APP: HOMEDETECT FOR iOS 16.2

One of the most significant parts of the HomeDetect project was the client mobile application that was developed for iOS 16.2 using Swift 5 and SwiftUI framework. The selection of iOS as the platform for the client application was a strategic decision, as it offered multiple advantages. The iOS platform was mainly considered for its known and reliable security features such as hardware-based encryption. It is important to note that iOS is a closed operating system and alongside with the App Store being the only option for application distribution minimises the risks of security threats such as malware and etc (Garg & Baliyan, 2021). Additionally, the use of Swift 5 and SwiftUI framework in the development of the application provided the latest features and security implementations, therefore providing an additional layer of protection for the end-user.

It was then decided to utilise AWS Amplify service and Amplify SDK to connect the iOS application to AWS. There are several advantages for using Amplify not only for developing mobile client applications but for web applications as well due to it being one of the latest and most advanced tools that are offered by AWS. It provides a unified and easy-to-use interface for building, deploying, and managing both the front-end and back-end of web and mobile applications. The Amplify CLI, was installed locally which allowed for seamless addition of services to the HomeDetect mobile client. The Amplify set-up guide for Swift can be found using the following link:

https://docs.amplify.aws/start/q/integration/ios/

The following steps provide a brief outline the Amplify SLI Configuration, append when needed:

1. Install amplify **npm install -g @aws-amplify/cli**.
2. Navigate to the XCode project directory.
3. Call configuration **amplify configure**, follow the steps outputted (e.g. connect AWS account etc.)
4. Initiate Amplify **amplify init**.
5. Upload configuration to the cloud **amplify push**
6. Add package dependencies to the XCode project using https://github.com/aws-amplify/amplify-swift URL.
7. Add Cognito to project **amplify add auth**, follow the steps outputted, configuring advanced settings are not necessary.
8. Update configuration to the cloud **amplify push**

Upload configuration to the cloud amplify pushAWS Cognito was used to manage user authentication for this project. Paired with the Amplify CLI and Amplify SDK v1.28.4, the set-up of Cognito and IAM policies was automated which eliminated the chance of user errors in line with the latest best practices and guidelines of AWS, which was critical when developing HomeDetect Project. The Cognito authentication was configured to use email and password for authentication in order to minimise the information stored about the user while maintaining secure access to the application.

After configuring the Amplify CLI it was necessary to build the iOS application itself. Due to the lack of knowledge of Swift programming language, it was necessary to research the key terminology for SwiftUI. The application was developed using 4 different Scenes (aka Views), which represent different user screens:

1. LoginView.swift – used for user login (See Section 2.1.1).
2. SignUpView.swift – used for registering new users (See Section 2.1.1).
3. ConfirmView.swift – used for confirmation of the email registered for new users (See Section 2.1.1).
4. ContentView.swift – the main screen of the application (See Section 2.1.2).

2.1.1 Authentication Flow

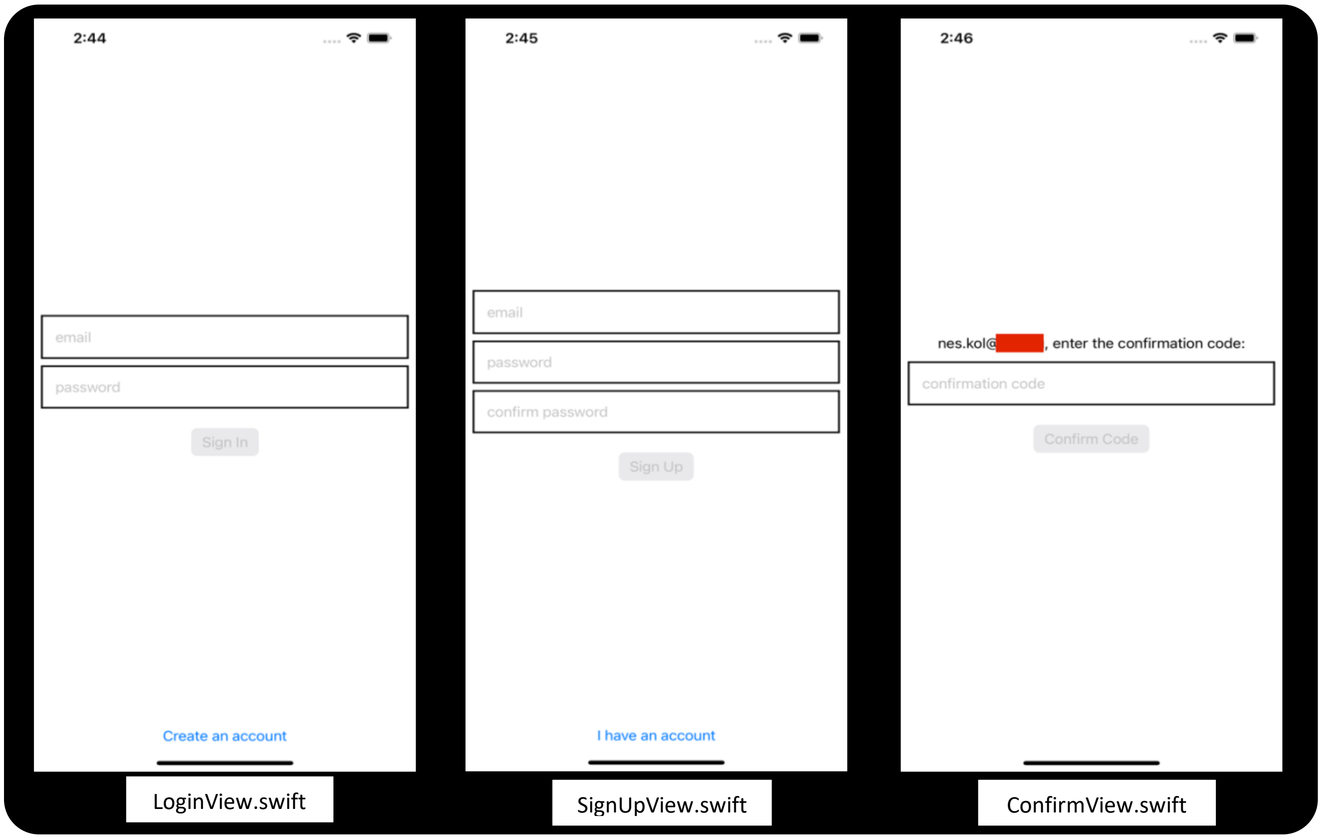
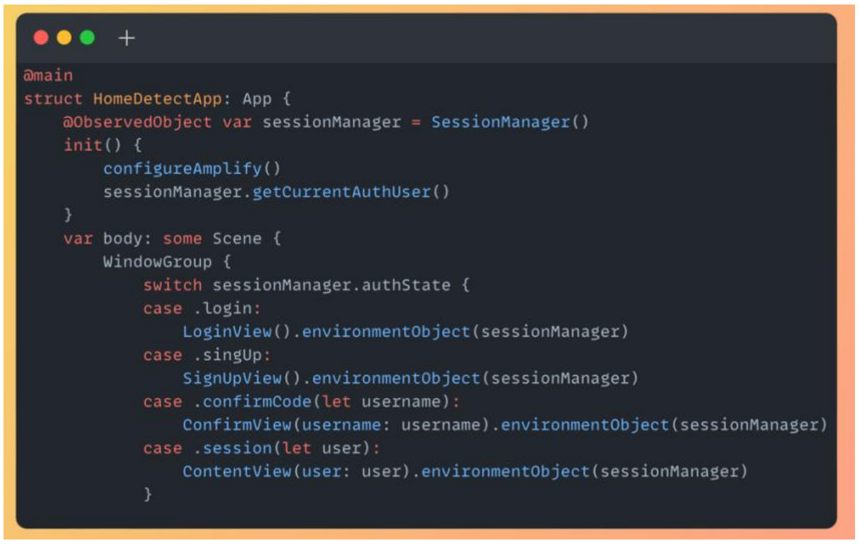


Figure 1: Authentication flow

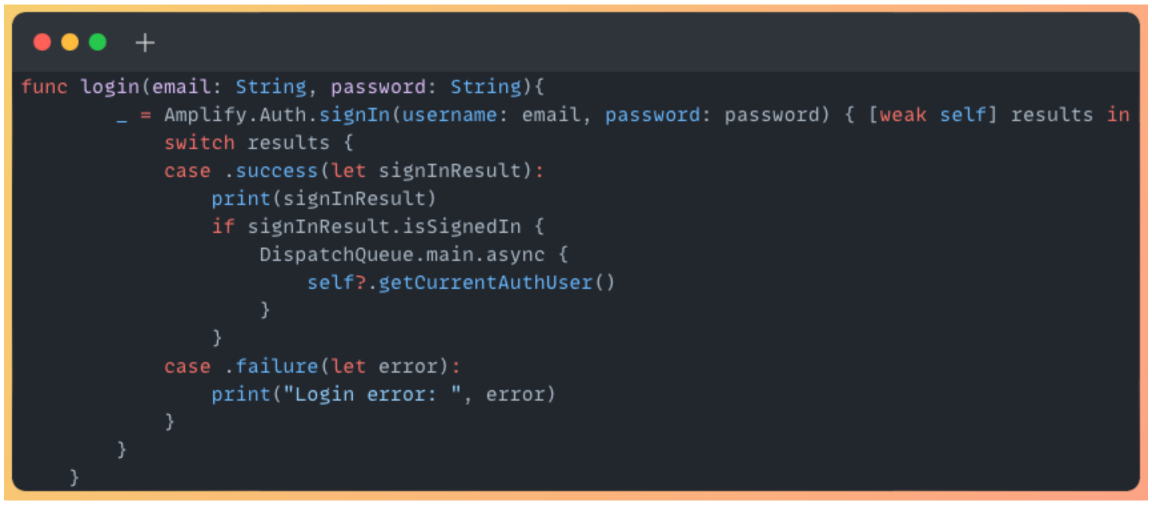
In order to limit the access to the main functionality of the application, the session manager was introduced which stores the currently authenticated user even if the user closes down the app and manages the navigation between the screens of the app (Refer to Appendix E). The user screens are then displayed based on the result of the session manager's authentication state, this means that if the user was previously logged on and the administrator deletes the account from AWS Cognito user pool, when the application launches, the user will be prompted with the login screen again which protects the access to the valuable resources (Figure 2):



```
@main
struct HomeDetectApp: App {
    @ObservedObject var sessionManager = SessionManager()
    init() {
        configureAmplify()
        sessionManager.getCurrentAuthUser()
    }
    var body: some Scene {
        WindowGroup {
            switch sessionManager.authState {
            case .login:
                LoginView().environmentObject(sessionManager)
            case .singUp:
                SignUpView().environmentObject(sessionManager)
            case .confirmCode(let username):
                ConfirmView(username: username).environmentObject(sessionManager)
            case .session(let user):
                ContentView(user: user).environmentObject(sessionManager)
            }
```

Figure 2: HomeDetect.swift main

The SessionManager() class was implemented with all necessary functions to provide comprehensive authentication functionality throughout the application. In case of LoginView.swift, the sessionManager.login() function was attached to the button (Figure 3):



```
func login(email: String, password: String){
        _ = Amplify.Auth.signIn(username: email, password: password) { [weak self] results in
            switch results {
            case .success(let signInResult):
                print(signInResult)
                if signInResult.isSignedIn {
                    DispatchQueue.main.async {
                        self?.getCurrentAuthUser()
                    }
                }
            case .failure(let error):
                print("Login error: ", error)
            }
        }
    }
```

Figure 3: Cognito login function

When calling the .login() function using the implemented input fields of email and password, the request is send to AWS using built in Amplify.Auth.signIn() request, if the login is successful, the authenticated user details will be fetched which would then be automatically picked up by the main if the user state is set to .session() which will prompt the user interface to switch to the main user interface of HomeDetect. The functions in SessionManager() class are implemented in accordance with the Amplify for Swift Documentation and have similar format. Refer to Appendix E to view all functions of the class. This is just the key examples of the code implementation, see Appendices A-F for full Swift code.

2.1.2 Main User Interface (Front-end)

The design of the main user interface was developed with the idea of minimalism and ease of use, just like the other user screens. After logging in with Cognito, the user is presented with an instance of Apple Maps and a location sharing toggle witch. This design was chosen to limit the available input, this was done to minimise the chances of injection-like attacks. The map displays the detection radius and the user location if the location toggle is switched on (Figure 4).

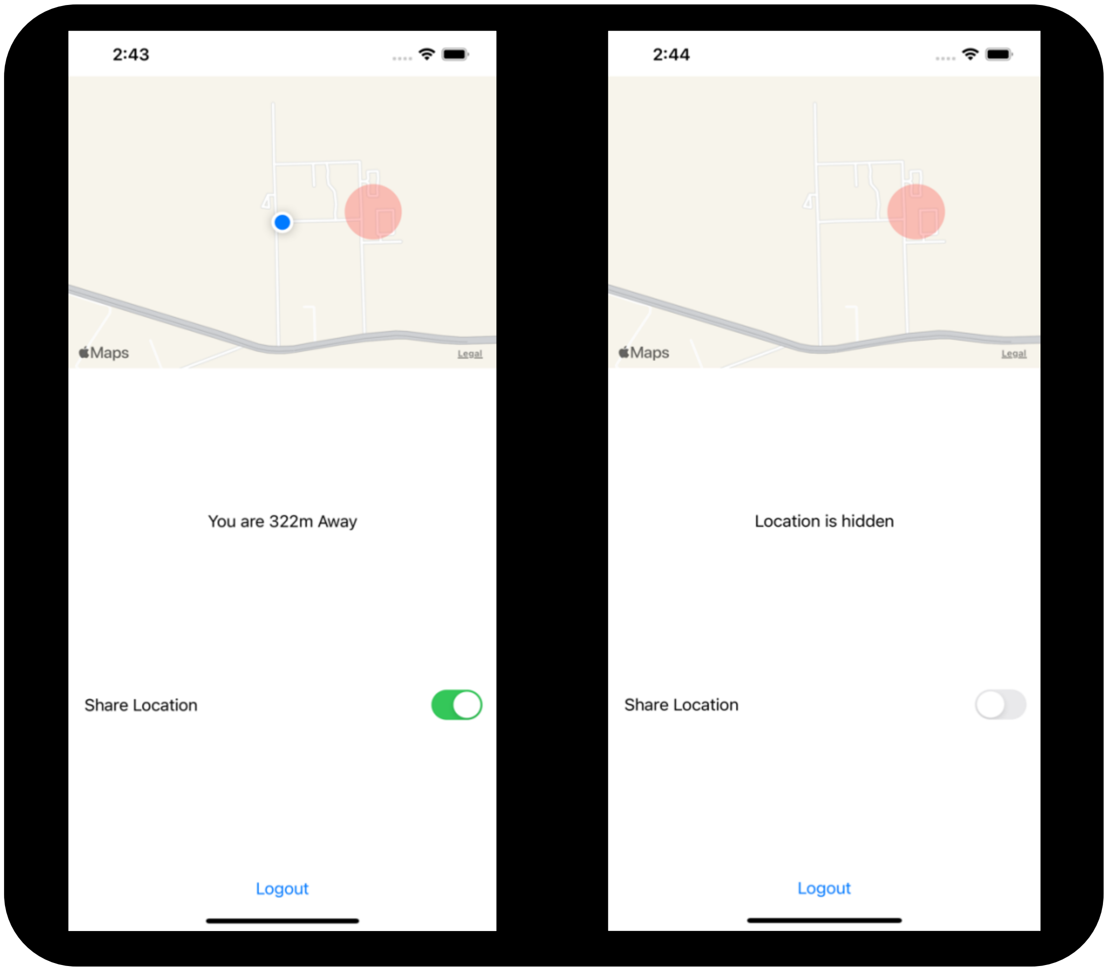


Figure 4: Main user interface

It was designed that when the user toggles the location sharing on, the application automatically sends the user location to a Lambda function through an AWS API Getaway, details of the backend were explained in Section 2.2. The location toggle also enables manual controls of the lights.

2.2 Cloud Technology Backend: Amazon Web Services (AWS)

The next step of HomeDetect development was the design of Cloud Based backend in a form of AWS Lambda function. Lambda service was chosen for its to automatically scale the application based on the number of requests, which helped to ensure that the application could handle sudden increases in traffic without interruption as well as more efficient billing compared to hosting a server using AWS EC2. Additionally, Lambda functions ran within a highly secure environment that was protected by Amazon VPC, which helped to protect the data and functions from unauthorised access.

2.2.1 Lambda Function

At this stage, it was planned that the location of the device would be communicated to the Lambda function in a form of longitude and latitude coordinate in some way. Using Amplify CLI and **amplify add function** command, the homedetectFunction was added to Lambda service. Node.js was selected as the programming language for easier local compilation and the 'Hello Word' template was set during the creation of the function.

It was required that the Lambda function would calculate the distance between two coordinates. It was decided to use Haversine formula (Veness, 2022), see full Lambda code in Appendix G:

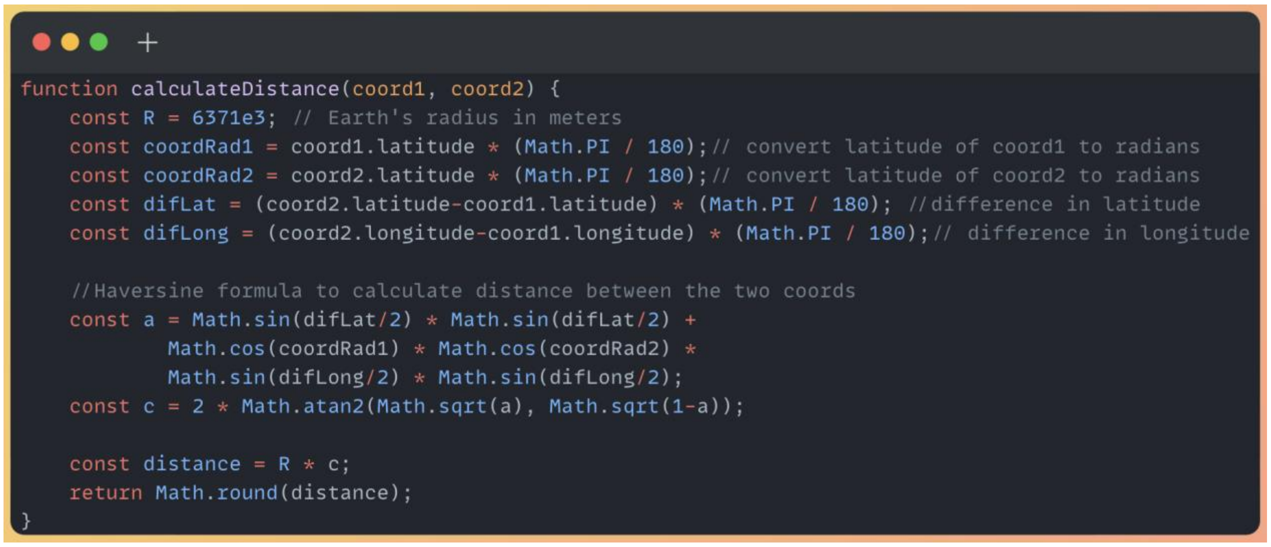
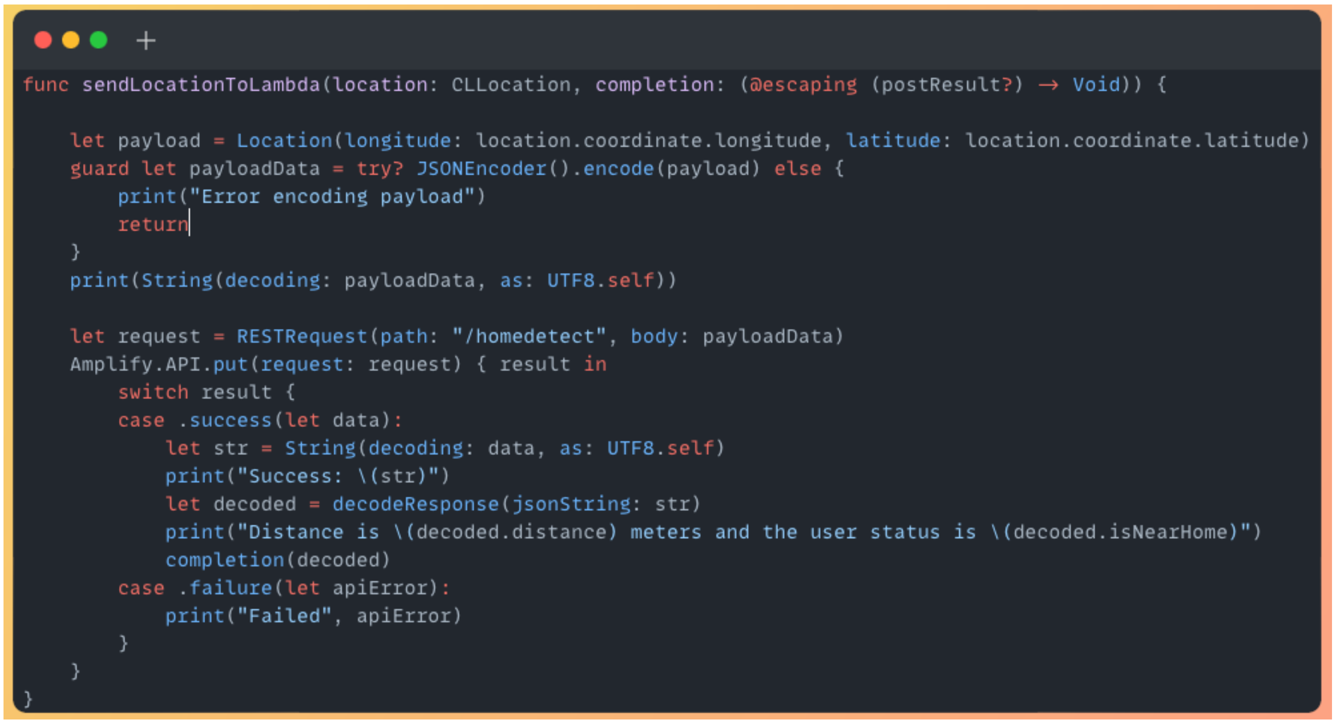

```
function calculateDistance(coord1, coord2) {
    const R = 6371e3; // Earth's radius in meters
    const coordRad1 = coord1.latitude * (Math.PI / 180);// convert latitude of coord1 to radians
    const coordRad2 = coord2.latitude * (Math.PI / 180);// convert latitude of coord2 to radians
    const difLat = (coord2.latitude-coord1.latitude) * (Math.PI / 180); //difference in latitude
    const difLong = (coord2.longitude-coord1.longitude) * (Math.PI / 180);// difference in longitude

    //Haversine formula to calculate distance between the two coords
    const a = Math.sin(difLat/2) * Math.sin(difLat/2) +
            Math.cos(coordRad1) * Math.cos(coordRad2) *
            Math.sin(difLong/2) * Math.sin(difLong/2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

    const distance = R * c;
    return Math.round(distance);
}
```

Figure 5: Calculating the distance between two coordinates in Lambda

2.2.2 AWS API Gateway and Location data communication

Next, it was needed to find a way to send the device location data, AWS API Gateway was chosen as the mode of data transmission as well as the Lambda trigger. API Gateway provided a secure and efficient way to transmit data between the mobile application and the Lambda function, ensuring secure and real-time data transmission. Additionally, it also provided a way to authenticate and authorise the mobile application, ensuring that only authorsed devices could send data to the Lambda function. The use of API Gateway helped to ensure the security and reliability of the data transmission, which was critical for the HomeDetect project. It was also created using the Amplify CLI with the **amplify add api**. The API Geteway was configured to use REST requests and connected to the Lambda fuction.

Finally the location was shared using the following function in the iOS application with accordance to the AWS Amplify documentation to insure secure data transmission of sensitive information (Figure):



```swift
func sendLocationToLambda(location: CLLocation, completion: (@escaping (postResult?) → Void)) {

    let payload = Location(longitude: location.coordinate.longitude, latitude: location.coordinate.latitude)
    guard let payloadData = try? JSONEncoder().encode(payload) else {
        print("Error encoding payload")
        return
    }
    print(String(decoding: payloadData, as: UTF8.self))

    let request = RESTRequest(path: "/homedetect", body: payloadData)
    Amplify.API.put(request: request) { result in
        switch result {
        case .success(let data):
            let str = String(decoding: data, as: UTF8.self)
            print("Success: \(str)")
            let decoded = decodeResponse(jsonString: str)
            print("Distance is \(decoded.distance) meters and the user status is \(decoded.isNearHome)")
            completion(decoded)
        case .failure(let apiError):
            print("Failed", apiError)
        }
    }
}
```

Figure 6: Sending data from iOS to Lmabda with API Geteway

As the RESTRequest() function was an asynchronous function, the “completion” attribute was added to fetch the response of the Lambda function, the full code of ContentView.swift can be found in Appendix A.

2.3 IoT Device: Raspberry Pi Zero W

Figure 7 demonstrates the layout of the RPi IoT device used for HomeDetect project, the IoT includes:

1. Micro Servo SG90 – attached to the body of the light switch, used to control the lights.
2. Status LED – shows the current status of the IoT, if **true** the lights should be on, if **false** the lights should be off.
3. Button – used for manually overriding the current lights state.

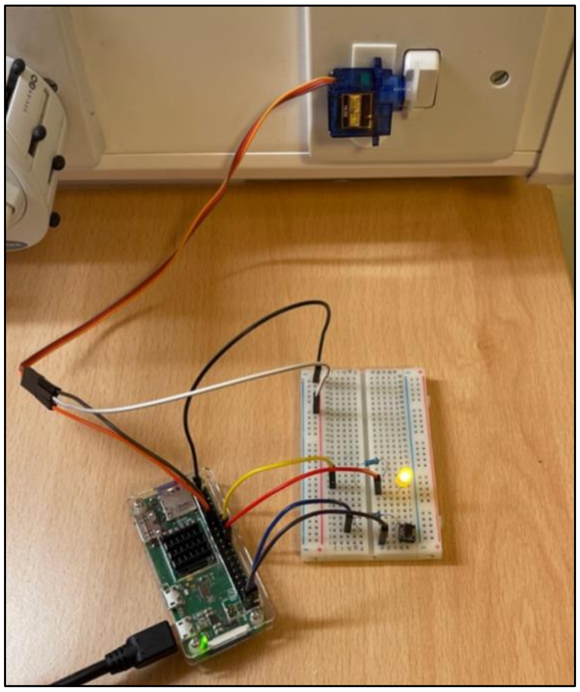


Figure 7: RPi IoT structure

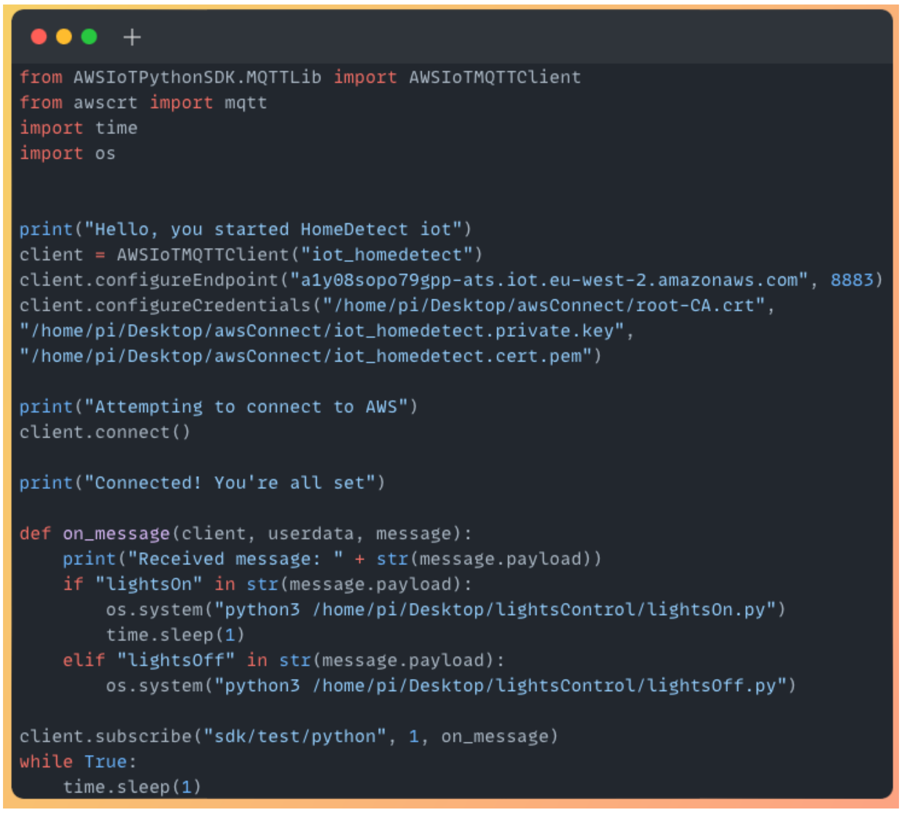
2.3.1 AWS IoT Core

In order to connect RPi Zero W to the AWS cloud, it was decided to use AWS IoT Core. IoT Core allows for secure and efficient communication between the RPi and the AWS cloud. MQTT messaging was used to transfer messages from the Lambda function to the RPi since it provides a secure and lightweight method of data transfer.

AWS IoT Core enables the use of the MQTT protocol to securely transmit messages between the Lambda function and the RPi, ensuring that the data is transmitted in real-time, which is essential for the HomeDetect project. Additionally, IoT Core also provides a way to authenticate and authorise the RPi, ensuring that only authorised devices can receive data from the Lambda function.

To connect RPi to AWS using IoT Core, AWS Console was used to create and register the RPi device on the IoT Core platform. A unique certificate and private key were then generated and transferred to the RPi IoT.

Next, using the generated certificate and private key, a Python script was constructed to use the AWS Python SDK to subscribe to an IoT topic which used the MQTT protocol to communicate with the Lambda function. It is important to note that in order to use IoT topics, the IAM of the Lambda needed to be manually configured to add the **AWSIoTFullAccess** policy (Figure 8):



```python
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
from awscrt import mqtt
import time
import os


print("Hello, you started HomeDetect iot")
client = AWSIoTMQTTClient("iot_homedetect")
client.configureEndpoint("a1y08sopo79gpp-ats.iot.eu-west-2.amazonaws.com", 8883)
client.configureCredentials("/home/pi/Desktop/awsConnect/root-CA.crt",
"/home/pi/Desktop/awsConnect/iot_homedetect.private.key",
"/home/pi/Desktop/awsConnect/iot_homedetect.cert.pem")

print("Attempting to connect to AWS")
client.connect()

print("Connected! You're all set")

def on_message(client, userdata, message):
    print("Received message: " + str(message.payload))
    if "lightsOn" in str(message.payload):
        os.system("python3 /home/pi/Desktop/lightsControl/lightsOn.py")
        time.sleep(1)
    elif "lightsOff" in str(message.payload):
        os.system("python3 /home/pi/Desktop/lightsControl/lightsOff.py")

client.subscribe("sdk/test/python", 1, on_message)
while True:
    time.sleep(1)
```

Figure 8: Python Script to use IoT Core and listen for messages from Lambda using IoT Topic

This script waits for messages in the **sdk/test/python** IoT topic, if the message contains instruction **lightsOn** or **lightsOff**, it triggers the corresponding Python script which adjusts the position of the servo to flip the light switch it was attached to. Refer to Appendices H, I and J for full code of the RPi python scripts.

2.3.2 Linux Loadable Kernel Modules (LKM)

Finally, a Linux Loadable Kernel Modules (LKM) was developed to use a button to manually override the current lights state. It was redundant as a Python script can be used to control the micro servo for this project, Python Script was considered to be more secure than LKM as it allows better process management and error handling, however, LKM was used to prove the concept. The call_usermodehelper() function was used to run the Python script from within the LKM, as the pwm.h library was not available on the RPi, which was necessary to allow precise and reliable control of the servo. There were some potential challenges related to the LKM as the button provided in the development kit was defective and occasionally triggered multiple times and for an unknown reason, the LKM would ignore button debouncing and trigger multiple times which sometimes made the RPi restart, however it was possible to capture instances where the LKM worked as expected. (Refer to Appendix K for full LKM code)

3 Discussion

3.1 Conclusion

The HomeDetect project aimed to create a smart home monitoring system that enhances security and convenience for the user. Through the use of an iOS application, Raspberry Pi IoT device, and Amazon Web Services (AWS) as the cloud technology, the project successfully achieved its objective. The iOS application, developed using AWS Amplify, communicated with the Raspberry Pi (RPi) IoT device via AWS IoT Core, AWS Cognito, AWS Lambda, and AWS API Gateway. The use of these technologies allowed for secure and efficient communication between the mobile application and the RPi, thus making the system both user-friendly and secure.

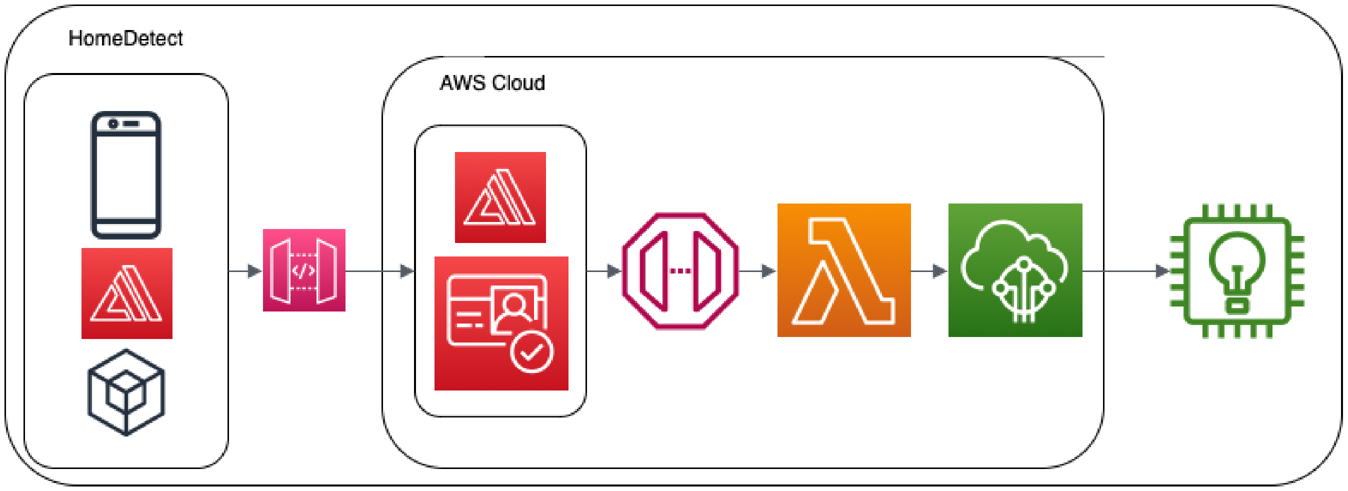


Figure 9: Project Structure

The use of AWS services such as AWS IoT Core, AWS Cognito, AWS Lambda, and AWS API Gateway in the project demonstrated how these technologies can be effectively utilized to create a secure system. Additionally, the use of Swift 5 and SwiftUI framework in the development of the iOS application ensured that the application was developed with the latest security best practices and guidelines.

However, the project faced a drawback while attempting to prove the concept of Linux Loadable Kernel Modules (LKM) as an alternative approach to control the micro servo. The LKM proved to be unreliable due to the unavailability of pwm.h library on the RPi meaning that the call_usermodehelper() function was required to call Python scripts from userspace. Although it was not a critical part of the project, it was clear that Python script was a better alternative for LKM to control the micro servo for this project, as it allows for better process management and error handling.

Overall, the HomeDetect project successfully met its aims and objectives. The final iOS application was developed in accordance with modern security standards, the data was transferred securely across the devices and cloud technologies using latest offerings and developer guides from AWS. The sensitive personal data of the users were not stored apart from the email addresses used form the login, which meant in case of data leak, the location data of the users would not be under risk.

3.2 FUTURE WORK

Future developments for the HomeDetect project include expanding the capabilities of the system by making it available on other platforms such as Android and wearable devices, such as smart watches. This would provide the users with more flexibility by allowing them to access the smart home monitoring system from multiple devices. Additionally, this would increase the system's accessibility, reaching a wider audience. The iOS application would need to be ported to the Android platform and necessary changes would be made to the Raspberry Pi IoT device to support communication with the wearable device.

Another aspect of future work for the HomeDetect project is the creation of a custom designed IoT device that can be tailored to meet the specific needs of the user. This can be achieved using 3D printing technology to create a more compact and aesthetically pleasing device. The design of the IoT device would require careful consideration, and necessary modifications would be made to the Raspberry Pi IoT device to support communication with the 3D printed device. This would provide a unique solution for the users who want to customise their smart home monitoring system.

REFERENCES

Lee, K. (2021) Create an iOS tracker application with Amazon Location Service and AWS Amplify, Amazon. Greenhaven Press/Gale. Available at: https://aws.amazon.com/blogs/mobile/create-an-ios-tracker-application-with-amazon-location-service-and-aws-amplify/ (Accessed: January 10, 2023).

Amazon Web Services (no date) AWS Amplify Documentation for iOS (Swift), Amplify docs. Available at: https://docs.amplify.aws/lib/q/platform/ios/ (Accessed: January 10, 2023).

StuartSmith (2016) Stuartsmith/raspberry-pi-control-SG90-servo-example: Control an SG90 servo motor from a Raspberry Pi, GitHub. Available at: https://github.com/StuartSmith/Raspberry-Pi-Control-Sg90-Servo-Example (Accessed: January 10, 2023).

Garg, S. and Baliyan, N., 2021. Comparative analysis of Android and iOS from security viewpoint. Computer Science Review, 40, p.100372.

Veness, C. (2022) Movable type scripts, Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript. Available at: https://www.movable-type.co.uk/scripts/latlong.html (Accessed: January 18, 2023).

Barnatt, C. (2020) Raspberry Pi Servo Motor Control, Explainingcomputers.com: Raspberry Pi Servo Motor Control. Available at: https://www.explainingcomputers.com/pi_servos_video.html (Accessed: January 24, 2023).

APPENDICES

APPENDIX A: HOMEDETECT IOS APP CODE CONTENTVIEW.SWIFT

```
//
//  ContentView.swift
//  HomeDetect
//
//  Created by Nick Revolver on 27/12/2022.
//

import Amplify

import SwiftUI
import MapKit
import CoreLocation

struct ContentView: View {

    let user: AuthUser

    @EnvironmentObject var sessionManager: SessionManager
    @Environment(\.colorScheme) var colorScheme: ColorScheme

    @State private var isLoginViewPresented = false

    @State private var region = MKCoordinateRegion(center: CLLocationCoordinate2D(latitude: 42.56381, longitude: 27.55713),
span: MKCoordinateSpan(latitudeDelta: 0.01, longitudeDelta: 0.01))

    @State private var shareLocation = true //used for toggling sharing location to AWS
    @State private var locBtnPressed = false //used to control permission request button
    @State private var locAlwaysBtnPressed = false //used to control always permission request button

    @State private var decodedResult = postResult(isNearHome: false, distance: 404)

    let locationManager = CLLocationManager()
```

```
@State private var previousLocation = CLLocation(
    latitude: 0.0000000,
    longitude: 0.0000000)

let MapLocations = [
        MapLocation(
            name: "Location of RaspberryPI",
            latitude: 42.56381,
            longitude: 27.55713),
        ]

@State private var sendToLambdaTime = Date().timeIntervalSince1970



var body: some View {
    VStack{
        GeometryReader{ geo in
            Map(
                coordinateRegion: $region,
                showsUserLocation: shareLocation,
                userTrackingMode: .constant(.follow),
                annotationItems: MapLocations) { location in
                    MapAnnotation(coordinate: location.coordinate) {
                        //Size per kilometer or any unit, just change the converted unit.
                        let kilometerSize = (geo.size.height/region.spanLatitude.value)
                        Circle()
                            .fill(Color.red.opacity(0.3))
                        //Keep it a circle
                            .frame(width: kilometerSize*200, height: kilometerSize*200)
                    }
                }
        }.frame(height: 300)

        Spacer()

        if locationManager.authorizationStatus == .notDetermined && locationManager.authorizationStatus != .denied
&& !locBtnPressed{
            Button("Give Location", action: {
                locationManager.requestWhenInUseAuthorization()
                locBtnPressed = true
            })
            .buttonStyle(.borderedProminent)
            .buttonBorderShape(.capsule)
        } else if locationManager.authorizationStatus != .authorizedAlways &&
locationManager.authorizationStatus != .authorizedWhenInUse{
            Spacer()
            Text("You did not give location permissions, the proximity features are unavailable").padding(.all)
            Spacer()
        }

        if locationManager.authorizationStatus == .authorizedAlways || locationManager.authorizationStatus
== .authorizedWhenInUse{ //check the user gave location permission
            if shareLocation{ //check user is sharing the location
                let locationUser = CLLocation(
                    latitude: locationManager.location?.coordinate.latitude ?? 0.0000000,
                    longitude: locationManager.location?.coordinate.longitude ?? 0.0000000)
                let locationPi = CLLocation(
                    latitude: MapLocations.first?.latitude ?? 0.0000000,
                    longitude: MapLocations.first?.longitude ?? 0.0000000)



                let distance = locationUser.distance(from: locationPi) //calculate distance from user to pi

                if distance < 1000 { //check distance is less than 1 km
                    Text("You are "+String(format: "%.0f", distance.rounded())+"m Away").onChange(of: locationUser) { newValue in
                        if abs(sendToLambdaTime - Date().timeIntervalSince1970) > 5{
                            sendLocationToLambda(location: newValue){ (decoded) in
                                if let decoded = decoded {
                                    decodedResult = decoded
                                    print(decodedResult.distance)
                                }
                            }
                            sendToLambdaTime = Date().timeIntervalSince1970
                        }
                    }
                } else {
```

```
                Text("You are "+String(format: "%.1f", distance/1000)+"km Away") //convert to km with 1 decimal place
              }

              // manual location send:
              /*
              Button("You are \(decodedResult.distance)m away"){
                sendLocationToLambda(location: locationUser){ (decoded) in
                  if let decoded = decoded {
                    decodedResult = decoded
                    print(decodedResult.distance)
                  }
                }
              }*/

            } else {
              Text("Location is hidden")
            }
            Spacer()
            Toggle("Share Location", isOn: $shareLocation)
              .padding(.all)

            if locationManager.authorizationStatus == .authorizedWhenInUse &&
locationManager.authorizationStatus != .authorizedAlways && !locAlwaysBtnPressed{
              Button("Give Always Location", action: {
                locationManager.requestAlwaysAuthorization()
                locAlwaysBtnPressed = true
              }).padding(.all)
            }
          }

          Spacer()
          Button("Logout") {
            sessionManager.logout()
          }
        }
      }
}
```

```
struct MapLocation: Identifiable {
    let id = UUID()
    let name: String
    let latitude: Double
    let longitude: Double
    var coordinate: CLLocationCoordinate2D {
        CLLocationCoordinate2D(latitude: latitude, longitude: longitude)
    }
}

extension MKCoordinateRegion{
    var spanLatitude: Measurement<UnitLength>{
        let loc1 = CLLocation(latitude: center.latitude - span.latitudeDelta * 0.5, longitude: center.longitude)
        let loc2 = CLLocation(latitude: center.latitude + span.latitudeDelta * 0.5, longitude: center.longitude)
        let metersInLatitude = loc1.distance(from: loc2)
        return Measurement(value: metersInLatitude, unit: UnitLength.meters)
    }
}



//Back-end
struct Location: Codable {
   var longitude: Double
   var latitude: Double
}

struct postResult {
    var isNearHome: Bool
    var distance: Int
}

func sendLocationToLambda(location: CLLocation, completion: (@escaping (postResult?) -> Void)) {

    let payload = Location(longitude: location.coordinate.longitude, latitude: location.coordinate.latitude)
    guard let payloadData = try? JSONEncoder().encode(payload) else {
        print("Error encoding payload")
        return
    }
    print(String(decoding: payloadData, as: UTF8.self))

    let request = RESTRequest(path: "/homedetect", body: payloadData)
    Amplify.API.put(request: request) { result in
        switch result {
        case .success(let data):
            let str = String(decoding: data, as: UTF8.self)
            print("Success: \(str)")
            let decoded = decodeResponse(jsonString: str)
            print("Distance is \(decoded.distance) meters and the user status is \(decoded.isNearHome)")
            completion(decoded)
        case .failure(let apiError):
            print("Failed", apiError)
        }
    }
}


func decodeResponse (jsonString: String) -> postResult {
    var isNearHome: Bool = false
    var distance: Int = 404
    var tempString: String = ""

    if let range = jsonString.range(of: #", \"distance\": "#) {
        tempString = String(jsonString[..<range.lowerBound])

        var substring = jsonString[range.upperBound...] // or str[str.startIndex..<range.lowerBound]
        substring.removeLast()
        distance = Int(substring)!
    }
    else {
      print("String not present")
    }

    if let range = tempString.range(of: #""\"isNearHome\": "#) {
        var substring = String(tempString[range.upperBound...]) // or str[str.startIndex..<range.lowerBound]
        //print(substring)
        isNearHome = Bool(substring)!
    }
    else {
      print("String not present")
    }

    let decoded = postResult(isNearHome: isNearHome, distance: distance)
    return decoded
}
```

Appendix B: HomeDetect iOS app code LoginView.swift

```
//
//  LoginView.swift
//  HomeDetect
//
//  Created by Nick Revolver on 29/12/2022.
//

import SwiftUI

struct LoginView: View {

    @EnvironmentObject var sessionManager: SessionManager
    @Environment(\.colorScheme) var colorScheme: ColorScheme

    @State private var email: String = ""
    @State private var password: String = ""

    var body: some View {
        VStack{
            Spacer()

            HStack{
                Spacer()
                TextField("email", text: $email)
                    .keyboardType(.emailAddress)
                    .autocorrectionDisabled()
                    .autocapitalization(.none)
                    .padding(/*@START_MENU_TOKEN@*/.all/*@END_MENU_TOKEN@*/)
                    .border(colorScheme == .dark ? Color.gray : Color.black, width: 2)
                Spacer()
            }

            HStack{
                Spacer()
                SecureField("password", text: $password)
                    .padding(/*@START_MENU_TOKEN@*/.all/*@END_MENU_TOKEN@*/)
                    .border(colorScheme == .dark ? Color.gray : Color.black, width: 2)
                Spacer()
            }

            Button("Sign In"){
                sessionManager.login(email: email, password: password)
            }
            .disabled((email.isEmpty || password.isEmpty) ? true : false)
            .buttonStyle(.borderedProminent)
            .padding(.all)

            Spacer()

            Button("Create an account") {
                sessionManager.showSignUp()
            }
        }
    }
}

struct LoginView_Previews: PreviewProvider {
    static var previews: some View {
        LoginView()
    }
}
```

APPENDIX C: HOMEDETECT IOS APP CODE SIGNUPVIEW.SWIFT

```swift
//
//  SignUpView.swift
//  HomeDetect
//
//  Created by Nick Revolver on 30/12/2022.
//

import SwiftUI

struct SignUpView: View {

    @EnvironmentObject var sessionManager: SessionManager
    @Environment(\.colorScheme) var colorScheme: ColorScheme

    @State private var username: String = ""
    @State private var email: String = ""
    @State private var password: String = ""
    @State private var confirmPassword: String = ""

    var body: some View {

        VStack{

            Spacer()

            HStack{
                Spacer()
                TextField("email", text: $email)
                    .keyboardType(.emailAddress)
                    .autocorrectionDisabled()
                    .autocapitalization(.none)
                    .padding(/*@START_MENU_TOKEN@*/.all/*@END_MENU_TOKEN@*/)
                    .border(colorScheme == .dark ? Color.gray : Color.black, width: 2)
                Spacer()
            }
            HStack{
                Spacer()
                SecureField("password", text: $password)
                    .padding(/*@START_MENU_TOKEN@*/.all/*@END_MENU_TOKEN@*/)
                    .border(colorScheme == .dark ? Color.gray : Color.black, width: 2)
                Spacer()
            }

            HStack{
                Spacer()
                SecureField("confirm password", text: $confirmPassword)
                    .padding(/*@START_MENU_TOKEN@*/.all/*@END_MENU_TOKEN@*/)
                    .border(colorScheme == .dark ? Color.gray : Color.black, width: 2)
                Spacer()
            }

            Button("Sign Up"){
                sessionManager.signUp(email: email, password: password)
            }
            .disabled(checkInput(email: email, password: password, confirmPassword: confirmPassword))
            .buttonStyle(.borderedProminent)
            .padding(.all)

            Spacer()

            Button("I have an account") {
                sessionManager.showLogin()
            }

        }
    }
}

private func checkInput (email: String, password: String, confirmPassword: String) -> Bool {
    if email.isEmpty || password.isEmpty || confirmPassword.isEmpty {
        return true
    } else if password != confirmPassword {
        return true
    } else {
        return false
    }
}

struct SignUpView_Previews: PreviewProvider {
    static var previews: some View {
        SignUpView()
    }
}
```

APPENDIX D: HOMEDETECT IOS APP CODE CONFIRMVIEW.SWIFT

```
//
//  ConfirmView.swift
//  HomeDetect
//
//  Created by Nick Revolver on 30/12/2022.
//

import SwiftUI

struct ConfirmView: View {

    let username: String

    @EnvironmentObject var sessionManager: SessionManager
    @Environment(\.colorScheme) var colorScheme: ColorScheme

    @State private var confirmCode: String = ""

    var body: some View {
        VStack{

            Spacer()

            Text(username+", enter the confirmation code:")

            HStack{
                Spacer()
                TextField("confirmation code", text: $confirmCode)
                    .keyboardType(.numberPad)
                    .autocorrectionDisabled()
                    .autocapitalization(.none)
                    .padding(/*@START_MENU_TOKEN@*/.all/*@END_MENU_TOKEN@*/)
                    .border(colorScheme == .dark ? Color.gray : Color.black, width: 2)
                Spacer()
            }
            Button("Confirm Code"){
                sessionManager.confirm(username: username, code: confirmCode)
            }
            .disabled(confirmCode.isEmpty)
            .buttonStyle(.borderedProminent)
            .padding(.all)

            Spacer()
        }
    }
}

struct ConfirmView_Previews: PreviewProvider {
    static var previews: some View {
        ConfirmView(username: "Nick")
    }
}
```

APPENDIX E: HOMEDETECT IOS APP CODE SESSIONMANAGER.SWIFT

```
//
//  SessionManager.swift
//  HomeDetect
//
//  Created by Nick Revolver on 30/12/2022.
//

import Amplify
import AWSCognitoAuthPlugin
import SwiftUI

enum AuthState{
    case singUp
    case login
    case confirmCode(username: String)
    case session(user: AuthUser)
```

```
}

class SessionManager: ObservableObject {
    @Published var authState: AuthState = .login

    func getCurrentAuthUser() {
        if let user = Amplify.Auth.getCurrentUser() {
            authState = .session(user: user)
        } else {
            authState = .login
        }
    }

    func showSignUp() {
        authState = .singUp
    }

    func showLogin() {
        authState = .login
    }

    func signUp(email: String, password: String) {
        //let attributes = [AuthUserAttribute(.email, value: email)]
        //let options = AuthSignUpRequest.Options(userAttributes: attributes)

        _ = Amplify.Auth.signUp(username: email, password: password) { [weak self] results in

            switch results {
            case .success(let signUpResult):
                print("Sign up result: ", signUpResult)

                switch signUpResult.nextStep {
                case .done:
                    print("Finished sign up")
                case .confirmUser(let details, _):
                    print (details ?? "no details")

                    DispatchQueue.main.async {
```

```
                    self?.authState = .confirmCode(username: email)
                }
            }

        case .failure(let error):
            print("Sign up error: ", error)
        }
    }
}

func confirm(username: String, code: String) {
    _ = Amplify.Auth.confirmSignUp(for: username, confirmationCode: code) { [weak self] results in
        switch results {
        case .success(let confirmResult):
            print(confirmResult)
            if confirmResult.isSignupComplete {
                DispatchQueue.main.async {
                    self?.showLogin()
                }
            }

        case .failure(let error):
            print("Sign up code error: ", error)
        }
    }
}

func login(email: String, password: String){
    _ = Amplify.Auth.signIn(username: email, password: password) { [weak self] results in
        switch results {
        case .success(let signInResult):
            print(signInResult)
            if signInResult.isSignedIn {
                DispatchQueue.main.async {
                    self?.getCurrentAuthUser()
                }
            }
        case .failure(let error):
                print("Login error: ", error)
            }
        }
    }

    func logout() {
        _ = Amplify.Auth.signOut(){ [weak self] results in
            switch results {
            case .success:
                DispatchQueue.main.async {
                    self?.getCurrentAuthUser()
                }
            case .failure(let error):
                print("Logout error: ", error)
            }

        }
    }
}
```

APPENDIX F: HOMEDETECT IOS APP CODE HOMEDETECTAPP.SWIFT

```
//
//  HomeDetectApp.swift
//  HomeDetect
//
//  Created by Nick Revolver on 27/12/2022.
//

import Amplify
import AWSCognitoAuthPlugin
import AWSAPIPlugin

import SwiftUI
import CoreLocation
import MapKit

@main
struct HomeDetectApp: App {

    @ObservedObject var sessionManager = SessionManager()

    init() {
        configureAmplify()
        sessionManager.getCurrentAuthUser()
    }

    var body: some Scene {
        WindowGroup {
            switch sessionManager.authState {
            case .login:
                LoginView().environmentObject(sessionManager)
            case .singUp:
                SignUpView().environmentObject(sessionManager)
            case .confirmCode(let username):
                ConfirmView(username: username).environmentObject(sessionManager)
            case .session(let user):
                ContentView(user: user).environmentObject(sessionManager)
            }

        }
    }
}

private func configureAmplify() {
    do {
        try Amplify.add(plugin: AWSCognitoAuthPlugin())
        try Amplify.add(plugin: AWSAPIPlugin())
        try Amplify.configure()
        print("Amplify configured with API and Auth plugin")
    } catch {
        print("An error occurred setting up Amplify: \(error)")
    }
}

//not really neccessary -----------------------------------------------------------
class LocationModel: NSObject, ObservableObject {
    private let locationManager = CLLocationManager()
    @Published var authorisationStatus: CLAuthorizationStatus = .notDetermined

    override init() {
        super.init()
        self.locationManager.delegate = self
    }

    public func requestAuthorisation(always: Bool = false) {
        if always {
            self.locationManager.requestAlwaysAuthorization()
        } else {
            self.locationManager.requestWhenInUseAuthorization()
        }
    }
}

extension LocationModel: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus) {
        self.authorisationStatus = status
    }
}
```

APPENDIX G: AWS LAMBDA FUNCTION

```
const AWS = require('aws-sdk');
const iot = new AWS.IotData({endpoint: 'a1y08sopo79gpp-ats.iot.eu-west-2.amazonaws.com'});

const homeLocation = {
    latitude: 56.46072,
    longitude: -2.98128
};

exports.handler = async (event) => {
    try{
        const jsonData = JSON.parse(JSON.stringify(event.body));
        const jsonBody = JSON.parse(jsonData)

        const userLocation = { latitude: jsonBody.latitude, longitude: jsonBody.longitude};
        const distance = calculateDistance(homeLocation, userLocation);

        const topic = 'sdk/test/python';

        if (distance < 50) {
            const params = {
                topic,
                payload: "lightsOn",
                qos: 0
            };
            await iot.publish(params).promise();
            return {
                statusCode: 200,
                body: JSON.stringify('"isNearHome": true, "distance": ' + distance),
            };
        } else {
            const params = {
                topic,
                payload: "lightsOff",
                qos: 0
            };
            await iot.publish(params).promise();
            return {
                statusCode: 200,
                body: JSON.stringify('"isNearHome": false, "distance": ' + distance),
            };
        }
    } catch (error) {
        console.log(error);
    }
};

function calculateDistance (coord1, coord2) {
    const R = 6371e3; // Earth's radius in meters
    const coordRad1 = coord1. latitude * (Math.PI / 180); // convert latitude of coord1 to radians
    const coordRad2 = coord2. latitude * (Math.PI / 180); // convert latitude of coord2 to radians
    const difLat = (coord2. latitude-coord1. latitude) * (Math.PI / 180); //difference in latitude
    const difLong = (coord2. longitude-coord1. longitude) * (Math.PI / 180); // difference in longitude
//Haversine formula to calculate distance between the two coords
const a = Math. sin(difLat/2) * Math. sin(difLat/2) +
    Math.cos (coordRad1) * Math.cos (coordRad2) *
    Math.sin (difLong/2) * Math.sin(difLong /2) ;
const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt (1-a));

const distance = R * C;
return Math. round (distance);
```

APPENDIX H: RPI IOT CORE SUBSCRIPTION HOMEDETECTIOT.PY

```python
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
from awscrt import mqtt
import time
import os


print("Hello, you started HomeDetect iot")
client = AWSIoTMQTTClient("iot_homedetect")
client.configureEndpoint("a1y08sopo79gpp-ats.iot.eu-west-2.amazonaws.com", 8883)
client.configureCredentials("/home/pi/Desktop/awsConnect/root-CA.crt",
"/home/pi/Desktop/awsConnect/iot_homedetect.private.key", "/home/pi/Desktop/awsConnect/iot_homedetect.cert.pem")

print("Attempting to connect to AWS")
client.connect()

print("Connected! You're all set")

def on_message(client, userdata, message):
    print("Received message: " + str(message.payload))
    if "lightsOn" in str(message.payload):
        os.system("python3 /home/pi/Desktop/lightsControl/lightsOn.py")
        time.sleep(1)
    elif "lightsOff" in str(message.payload):
        os.system("python3 /home/pi/Desktop/lightsControl/lightsOff.py")

client.subscribe("sdk/test/python", 1, on_message)
while True:
    time.sleep(1)
```

APPENDIX I: RPI LIGHTSON.PY (BARNATT, 2020)

```
# Import libraries
import RPi.GPIO as GPIO
import time

# Set GPIO numbering mode
GPIO.setmode(GPIO.BOARD)

# Set pin 11 as an output, and define as servo1 as PWM pin
GPIO.setup(11,GPIO.OUT)
servo1 = GPIO.PWM(11,50) # pin 11 for servo1, pulse 50Hz

# Start PWM running, with value of 0 (pulse off)
servo1.start(0)

# Loop to allow user to set servo angle. Try/finally allows exit
# with execution of servo.stop and GPIO cleanup :)

try:
    #Ask user for angle and turn servo to it
    angle = float(60)
    servo1.ChangeDutyCycle(2+(angle/18))
    time.sleep(0.5)
    servo1.ChangeDutyCycle(0)

finally:
    #Clean things up at the end
    servo1.stop()
    GPIO.cleanup()
```

APPENDIX J: RPI LIGHTSOFF.PY (BARNATT, 2020)

```
# Import libraries
import RPi.GPIO as GPIO
import time

# Set GPIO numbering mode
GPIO.setmode(GPIO.BOARD)

# Set pin 11 as an output, and define as servo1 as PWM pin
GPIO.setup(11,GPIO.OUT)
servo1 = GPIO.PWM(11,50) # pin 11 for servo1, pulse 50Hz

# Start PWM running, with value of 0 (pulse off)
servo1.start(0)

# Loop to allow user to set servo angle. Try/finally allows exit
# with execution of servo.stop and GPIO cleanup :)

try:
    #Ask user for angle and turn servo to it
    angle = float(0)
    servo1.ChangeDutyCycle(2+(angle/18))
    time.sleep(0.5)
    servo1.ChangeDutyCycle(0)

finally:
    #Clean things up at the end
    servo1.stop()
    GPIO.cleanup()
```

APPENDIX K: RPI LKM ISERVO.C

```
/*
 ================================================================================
 Name        : iServo.c
 Author      : Nick Nesterenko, inspired by Dr Abdul Razaq
 Version     : 0.7
 Copyright   : See Abertay copyright notice
 Description : RPi Zero Wireless - RPi IoT Core servo state manual override
 ================================================================================
 */
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kmod.h>
#include <linux/interrupt.h>
#include <linux/gpio.h>
#include <linux/timer.h>


static bool State = 0;
static unsigned int Led = 23;
static int button_gpio = 16;
static int irq_num;
static struct timer_list debounce_timer;

int lights_off( void )
{
    char *argv[] = { "/usr/bin/python3", "/home/pi/Desktop/lightsControl/lightsOff.py", NULL };
    static char *envp[] = {
    "HOME=/",
    "TERM=linux",
    "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL };

    return call_usermodehelper( argv[0], argv, envp, UMH_WAIT_PROC );
}

int lights_on( void )
```

```c
{
    char *argv[] = { "/usr/bin/python3", "/home/pi/Desktop/lightsControl/lightsOn.py", NULL };
    static char *envp[] = {
    "HOME=/",
    "TERM=linux",
    "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL };

    return call_usermodehelper( argv[0], argv, envp, UMH_WAIT_PROC );
}

static void debounce_timer_callback(struct timer_list *timer)
{
    int button_state = gpio_get_value(button_gpio);
    if (button_state == 0) {
        if (State == 1){
            State = 0;
            lights_off();
        } else {
            State = 1;
            lights_on();
        }
        gpio_set_value(Led, State);
        printk(KERN_INFO "It finally works!\n");
    }
}


static irqreturn_t button_isr(int irq, void *dev_id)
{
    mod_timer(&debounce_timer, jiffies + HZ/20);
    return IRQ_HANDLED;
}

int __init mod_entry_func(void)
{
    int ret;
    ret = gpio_request(button_gpio, "button_gpio");
    if (ret) {
        printk(KERN_ERR "Unable to request GPIO %d for button\n", button_gpio);
        return ret;
    }
    gpio_direction_input(button_gpio);
    irq_num = gpio_to_irq(button_gpio);
    ret = request_irq(irq_num, button_isr, IRQF_TRIGGER_RISING, "button_irq", NULL);
    if (ret) {
        printk(KERN_ERR "Unable to request IRQ %d for button\n", irq_num);
        gpio_free(button_gpio);
        return ret;
    }

    timer_setup(&debounce_timer, debounce_timer_callback, 0);


    return lights_off();
}

void __exit mod_exit_func(void)
{
    del_timer(&debounce_timer);
    free_irq(irq_num, NULL);
    gpio_free(button_gpio);
    return;
}

module_init(mod_entry_func);
module_exit(mod_exit_func);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("NN CMP408");
MODULE_DESCRIPTION("RPi IoT Core servo state manual override");
MODULE_VERSION("0.7");
```