rIoT Solutions

Team Members: Avary McCormack
University Affiliation: University of Florida
Project Title: Smart Bike Companion

Content

Executive Summary

My project was designed with the goal to provide conveniences to bikes and scooters that tend to be reserved for cars such as safety features, data collection, and real-time monitoring. The ESP32 microcontroller is able to communicate through a WebSocket connection via Amazon Web Services (AWS) with smart devices in order to provide information to a front-end interface where users may interact with the sensors and features. On the backend, I wrote a Lambda function to manage data flow, while a database (DynamoDB) was used to store historical sensor data for insightful graphs. Additionally, real-time GPS monitoring and automatic lighting systems add practicality and safety, with a map interface providing users with location and vehicle status.

Introduction

Bicycle and scooter users are extremely common especially around campus; however, many, especially cyclists, have limited safety features and technologies. It is common to have several sensors and safety features in cars and newer scooters but hard to find for cyclists. I decided to focus on this issue and develop a project that provided more features for these users. For example, many cyclists forget to turn their lights on at night, which is very dangerous; therefore, I decided to integrate this safety feature into my project. My device will sense when it is dark whether at night or a particularly cloudy day and will automatically turn the lights on. Another common issue is the fear of leaving your bike alone, even locked. Bikes can be stolen easily, so I planned to integrate a GPS monitoring system as well. Features such as these are what I focused on when designing this project.

Objective

The objective of this project was to create a system that integrates various sensors and functionalities into a bike using an ESP32 microcontroller capable of communicating with smart devices. Furthermore, I wanted to create a front-end interface that would allow users to easily view information. Along with the front-end, I wanted to incorporate a database in order to display informational graphs to users of previously recorded sensor readings. Some key features I wanted to implement were GPS monitoring, automatic lighting, and ambient environmental readings.

Methodology

When designing this project, I had to consider many different things such as hardware selection and integration, communication protocols, cloud services, and backend and front-end features included. I decided to use the EPS32 microcontroller because of its small size, availability on the market, affordability, and Wi-Fi capabilities.
The integration of hardware and software through communication protocols was my most important decision and my design underwent several iterations before I was satisfied. I chose to leverage AWS for all cloud services including the WebSocket connection (API Gateway), backend (Lambda Function), and database (DynamoDB). I then created a backend and deployed it as a Lambda Function to handle ESP32 data and database operations. This Lambda Function provided seamless communication with the front-end I constructed, enabling user information display.
I opted for WebSocket over HTTP due to its continuous open channels, essential for delivering real-time data in this project. I also chose to use a database in this project for persistent data so that when users toggled a button, it would be remembered and for storing a history of sensor readings. Finally, the features I decided to include in my front-end including graphs, a map, and sensor readings were those I prioritized as beneficial, whether that be safety or convenience, for cyclists.

Technical Details

Hardware

- ExpressIf ESP32
- Photoresistor (GM5539)
- Temperature/Humidity sensor (DHT11)

- GPS (NEO-6M)
- LEDs
- wires, resistors, breadboard

Software

- PlatformIO
- Vite + React
- Amazon Web Services (AWS)
  - DynamoDB
  - API Gateway
  - Lambda Functions
  - Amazon Location Map

Implemented Features

Front-end

- A control panel was created to display all information to users
- A map indicating the user's location and vehicle's location
- Toggles to control different modes

Modes

- Automatic Light Mode - when activated, lights will turn on automatically when it is dark outside or night time hours
- Manual Light Mode - when activated, lights will turn on no matter the time or brightness
- Parking Mode (GPS Monitoring) - when activated, the vehicle's gps location will be saved, the live monitoring will begin, and if the vehicle's newly recorded coordinates begin to move drastically, alerts will be issued for the user. If a user forgets to turn off this mode, this mode will automatically detect if a user is nearby and likely with the vehicle.

Sensors and Graphs

- Sensors Recorded and all old values are available to view through graphs on the front-end
  - Temperature
  - Humidity
  - Heat Index
  - Altitude
  - Speed
- Graphs have capability of selecting pre-set intervals to display data (such as 'last 24 hours')

Database and WebSocket

- Due to the use of a database, all information persists regardless of whether the ESP32 or front-end are live – this means all mode values and sensor data (including parked location)
- The WebSocket connection allows for a quick always-open line of communication between the front-end, back-end, and esp32.

Technical Description

ESP32

In order to properly test and develop the ESP32 and adjacent hardware, I used PlatformIO on VSCode. On startup, the ESP32 will try to connect to the campus wifi or if it is unable to, simply any wifi. Following that, it will attempt to connect to the WebSocket that I have set up. Lastly, the WebSocket connection will remain open in order to listen for commands from the back-end, and to send sensor data from the esp32 which will collect and send it automatically.

AWS - WebSocket and Back-end

The WebSocket is hosted on AWS API Gateway. I chose to use a WebSocket because having open communication seemed the simplest and most effective in this project. The back-end is hosted on AWS as a Lambda function. I have this function set up in order to receive data from the back-end and front-end and handle logic for each accordingly.

Handling Connections When a new device joins the WebSocket connection, this function will take note of the new connection id and store it in a DynamoDB table. This is done to ensure some messages are not needlessly sent to the same connection id.

Database The DynamoDB contains 2 tables: Connections and Vehicles. Connections are created and deleted at each connection and disconection and simply store the connection id. Vehicles is the table that will store all history about a particular ESP32. This table will contain fields to store sensor values, toggle modes, and history to be used later by the front-end.

Handling Messages On the first message that is sent, the Lambda function will check if a field 'devID' exists which indicates that an ESP32 sent the message, and see if that device exists in the database already. If it does not exist, a new Entry will be created for the corresponding 'devID'.

Regarding the handling of data, the data will have a certain 'type' depending on who sent it and what needs to be handled. For instance, the back-end will always send data with types– 'output' , 'status', or 'error' and the front-end will always send data with types– 'get' or 'set'. All functionality revolves around either retrieving from the database or updating the database. In both cases (back and front-end), whenever a Vehicle entry related to the functionality of lights is updated, a function to check if the lights should be on or off is performed.

Front-end

The front-end was created using Vite+React. Whenever refreshed, the front-end will send a message requesting a full database retrieval. This is done in order to properly set the last recorded sensor and mode values. Additionally, whenever the ESP32 sends data through the WebSocket, that data is then automatically sent, parsed, and displayed by the front-end.

I implemented graphs that display previous data. These graphs have the ability to be filtered by time so that users can adjust the interval of data that they are seeing. I also implemented a map using Amazon's Location Map that is capable of marking the user and their vehicle's location.

The front-end is also responsible for keeping track of a user's location and their vehicle. While both locations are stored in the database, the front-end will be doing checks to ensure that when Park mode is on, the vehicle does not leave a certain radius. Additionally, if a user is with the Vehicle when it begins moving, this will override the warnings since the user may have forgotten to toggle off Park mode. A warning is administered when it appears the vehicle has moved 30 feet from its saved location, and an alert is administered when it has moved 50 feet from its saved location.

Results and Evaluation

Utilizing Amazon Cloudwatch, I was able to get important logging messages to ensure my code and sensors were acting accordingly. Accurate readings were an important part of this project; therefore, I needed to ensure all sensors were accurate. While testing, I discovered that my original TMP36 sensor was not accurate and bought a new DHT11 sensor. This upgrade not only ensured accuracy but also provides humidity and heat index readings. I also tested the photoresistors under different light conditions to monitor what the readings typically were to gauge when the automatic lights should be switched on. Another issue I noticed was a long delay between toggling manual lights and the lights actually turning on. The delay was around 3 seconds which is a pretty long pause for something very simple. I was able to look through my back-end handler function and fix the issue.

Challenges Faced

During the design and development of my project, several challenges were encountered, primarily revolving around setting up the backend, frontend, and ESP32 communication protocols. One of the initial hurdles was determining the architecture and setup for both the backend and frontend components. Deciding on the appropriate cloud infrastructure, selecting suitable AWS services, and designing the frontend interface posed challenges due to the complexity of options available. In order to address this, I researched and planned extensively to ensure the cloud services and frameworks picked were capable and fit for this project. Finally, time was my biggest constraint. As a one-person team, I had the freedom to work on anything, at any time; however, the lack of help limited my project. Although time can not be addressed perfectly, I worked as much as I could to achieve a lot by prioritizing essential features and functionalities !

Future Work

There are many future plans I have to extend this project. I hope to add more features such as a sign up/log in, ability to add more vehicles, toggling between vehicles, and a way to save multiple wifi connections.

Conclusion

My project focused on enhancing the cycling experience through the integration of advanced technologies and features. By incorporating automatic lights, GPS monitoring, and real-time data transmission, this project aims to improve safety and security for cyclists. Users can now enjoy a safer riding experience with enhanced visibility and real-time location tracking. The project successfully implemented sensor readings for temperature, humidity, and other environmental metrics. These readings, along with historical data stored in a database, enable efficient data collection and analysis, providing valuable insights for users. Through WebSocket communication and a user-friendly front-end interface, cyclists have access to real-time monitoring and visualization of sensor readings, graphical representations, and interactive maps. This enhances situational awareness and facilitates informed decision-making during rides. Overall, my project represents a step toward leveraging technology to improve the safety, convenience, and enjoyment for cyclists.