AWS IoT Weather Monitoring System: Real-time Data Visualization and JavaScript Web Application Integration

Submitted by
Saptajit Banerjee,
3rd year B.Tech student
SCOPE, VIT Chennai
23rd June, 2023

Guided by,
Dr. Satya Aditya,
TIH Foundation for IoT and IoE (TIH-IoT),
Indian Institute of Technology,
Bombay - 400076

Executive Summary

This report presents a project that combines IoT, Cloud Computing, and Web Development to create an integrated weather monitoring system. The project utilizes the ESP32 S2 Feather micro-controller and the inbuilt BME 280 sensor to collect weather data, which is then sent to AWS IoT. The collected data is stored securely in DynamoDB, along with user authentication data stored in a separate table.

The system provides real-time visualization of weather data through a web application, leveraging asynchronous web sockets for seamless data updates. Additionally, the web application incorporates a Passport JS authentication system to ensure secure user access.

The successful implementation of this project showcases the power of IoT, Cloud Computing, and Web Development in creating a user-friendly and reliable weather monitoring system. The utilization of ESP32 S2 Feather, AWS IoT, DynamoDB, Lambdas, API Gateway, asynchronous web sockets, and Passport JS authentication enhances data collection, storage, visualization, and user authentication capabilities.

Introduction

The rapid advancements in technology have paved the way for innovative solutions in various domains. One such application is the integration of Sensors, Cloud Computing, and Web Development to create a comprehensive weather monitoring system. This project, developed by the author, demonstrates the potential of combining these technologies to deliver real-time weather data visualization and secure user access.

The project revolves around the utilization of the ESP32 S2 Feather microcontroller, equipped with the inbuilt BME 280 sensor, which enables the collection of accurate weather data. The weather data is the collection of temperature, humidity, pressure and altitude with date and time of receipt.The collected data is seamlessly transmitted to the cloud infrastructure through AWS IoT Core, thus enabling to leverage the power of Cloud Computing. Within the cloud ecosystem, the weather data is stored securely in DynamoDB, ensuring its availability for analysis and visualization.

To provide users with a user-friendly interface and real-time access to weather information, a web application has been developed. This web application incorporates asynchronous web sockets, allowing for instant updates and real-time visualization of weather data. Moreover, to ensure the security and privacy of user information, a Passport JS authentication system has been integrated, authenticating users based on users' data which is stored in a separate DynamoDB table.

The integration of Sensors, Cloud Computing, and Web Development in this project offers several advantages. Firstly, it enables the seamless collection, transmission, and storage of weather data, ensuring the availability of accurate and up-to-date information. Secondly, it provides a user-friendly interface through the web

application, allowing users to access weather data in real time and visualize it in a meaningful way. Lastly, the implementation of the Passport JS authentication system guarantees secure access to the system, protecting system information from unauthenticated entities.

Throughout the development process, careful consideration was given to the selection of appropriate tools, technologies, and frameworks to ensure a robust and efficient solution.

In the subsequent sections of this report, we will delve deeper into the project methodology, discuss the implementation of Sensors, Cloud Computing, and Web Development components, and present the outcomes and findings of this project. Additionally, we will explore potential areas for further development and improvement, highlighting the significance of this project in advancing the field of weather monitoring through the integration of Sensors, Cloud Computing, and Web Development.

Methodology

The project was implemented with a C++ code for the ESP32 S2 using the Arduino IDE. First, I conducted tests to verify the accuracy of the temperature readings from the built-in BME280 sensor. Then, the code was configured to establish a connection between the ESP32 and a available local WiFi access point, followed by connecting it to AWS IoT Core using the MQTT protocol. During the setup process, the ESP32 was registered as a single “Thing" in AWS IoT Core, and 2 auto-generated certificates were created to authenticate the MQTT connections. The first certificate is called the Device certificate which is used to identify the ESP32 as the single “Thing” on AWS and the second certificate is called “Root CA” certificate which is used to secure and authenticate MQTT connections between ESP32 and AWS IoT Core.

The device certificate is used in the AWS IoT system for authentication, secure communication, access control, and authorization. It verifies the device's identity, enables encryption for data privacy and integrity, imposes fine-grained permissions, and determines the device's level of access. The certificate ensures secure and trusted connectivity between devices and AWS IoT Core, allowing for secure data exchange and protecting the integrity of the IoT system. The Device Certificate contains sensitive information which are:

- **Public Key**: The device certificate includes the public key used for encryption and verification of digital signatures.
- **Private Key**: The device certificate is paired with a private key that is securely stored on the device and used for decryption and signing operations.
- **Certificate Identifier**: A unique identifier assigned to the certificate for identification and management purposes.
- **Subject Information**: The subject information identifies the device or thing to which the certificate is issued. It typically includes details like the device ID, serial number, and other identifying information.

- **Issuer Information**: The issuer information identifies the certificate authority (CA) that issued the device certificate. It may include details like the CA's name, location, and other metadata.
- **Validity Period**: The device certificate has a start date and an expiration date, defining the period during which it is considered valid.
- **Digital Signature**: The device certificate is digitally signed by the issuing CA using their private key, ensuring the integrity and authenticity of the certificate.
- **Certificate Chain**: The device certificate may also include the chain of intermediate certificates that link it to the root CA certificate, establishing trust in the certificate's authenticity.
- **Key Usage**: The device certificate may specify the allowed key usage purposes, such as encryption, digital signature, key agreement, or other purposes.
- **Extended Key Usage**: This field specifies any additional extended key usage purposes allowed for the certificate, such as client authentication, server authentication, or code signing.

This Root CA certificate contains sensitive information which are public key, issuer information, subject information, validity period, digital signature and certificate authority information. This certificate is for verifying the authenticity and integrity of TLS certificates used in MQTT connections. The TLS protocol is used for encryption of the MQTT Payload. Additionally, we created four policies for actions such as 'Subscribe', 'Publish', 'Connect', and 'Receive', specifying the respective topics. The certificate, along with the private and public keys for the connection, were downloaded. To secure the MQTT connection between the ESP32 and AWS IoT Core within TLS handshake process, RSA encryption was implemented for key exchange and authentication. The RSA key size was of 2048 bits.

The WiFi SSID, password, public and private keys, and certificate for MQTT authentication were stored in a .sh file. Supplementary code was written for the ESP32 to maintain a continuous connection, automatically reconnecting to AWS IoT Core if the connection is lost, before transmitting the weather data.

Once the ESP32 successfully established a connection with AWS IoT Core, the temperature data was transmitted over the MQTT connection to the AWS IoT Core service at a configurable interval of 2 minutes. The data was then forwarded to three AWS Lambda functions.

The first lambda function is named “Send Message”.This function sends the data to the web application via a WebSocket API managed by AWS API Gateway.

The second Lambda function, named "Connection," facilitates the establishment of the WebSocket API connection between the API and the web application. In the context of AWS, AWS Systems Manager provides a parameter store that can be utilized to store a connection ID. This connection ID is essential for the "Send Message Function" as it serves as a means to direct data to a specific connection designated by the ID. By leveraging the parameter store within AWS Systems Manager, the connection ID can be securely stored and retrieved when needed for

proper data routing. From this storage, the connection ID of the connection made by the web application to the WebSocket API is sent to the "Send Message" Lambda function, responsible for transmitting weather data to the web application. Each new connection from the web application to the WebSocket API receives a unique connection ID, which is essential for the "Send Message" Lambda function to send data to the specific connection on the socket. This connection corresponds to the one established by the web application to receive data from AWS IoT Core.

The third lambda function is named “Storage”. This function receives data from AWS IoT Core and parses it. After parsing the data, this function sends the parsed data to DynamoDB with time and date of receipt.

The WebSocket API was configured to connect with”Send Message” and “Connection” Lambda functions. AWS IoT Core was set up to route the weather data received from the ESP32 to the “Send Message” Lambda function. Simultaneously, the data was also stored in DynamoDB along with the corresponding date and time using the “Storage” function.

AWS IoT Core is again configured to have 2 rules. The first rule instructs AWS IoT Core to send the weather data to “Send Message” lambda function. The second rule instructs AWS IoT Core to send the weather data to the “Storage” lambda function.

In the web application, a combination of technologies was utilized to provide a comprehensive and interactive user experience. JavaScript, jQuery, Express.js, Passport.js, HighCharts.js, HTML, and CSS formed the core components of the web application.

JavaScript played a crucial role in both the backend and frontend development. It allowed for dynamic and interactive elements on the web pages, enabling seamless user interactions. The versatility of JavaScript ensured efficient data processing and manipulation.

To enhance the visual representation of the weather data, HighCharts.js was employed. This powerful charting library facilitated the creation of visually appealing and informative line charts. These charts effectively conveyed the trends and patterns of the weather data, making it easier for users to interpret and analyze. jQuery was utilized to provide a smooth and responsive user interface. It facilitated the seamless display of the latest weather data, enabling real-time updates without requiring a page reload. jQuery's lightweight nature and extensive feature set made it a valuable tool for efficient DOM (Domain Manipulation) manipulation and event handling.

Express.js, a robust web application framework for Node.js, was chosen to manage sessions within the web application. It provided a structured approach to handle user sessions, ensuring secure and reliable authentication. Express.js offered various middleware and routing capabilities, simplifying the development process and enhancing the overall performance of the application.

Passport.js served as a valuable authentication middleware for Node.js applications. It provided a comprehensive set of authentication strategies and mechanisms, making it easier to implement user authentication and session validation. By leveraging Passport.js, user credentials were securely managed, ensuring a robust and reliable authentication process.

HTML and CSS were fundamental in creating the structure, layout, and styling of the web pages. HTML provided the markup structure, defining the content and layout of the application. CSS was employed to enhance the visual appeal of the web pages, applying custom styles and design elements to create an intuitive and aesthetically pleasing user interface.

Overall, the combination of JavaScript, jQuery, Express.js, Passport.js, HighCharts.js, HTML, and CSS formed a powerful stack of technologies that enabled the development of a feature-rich and visually engaging web application. These technologies worked together seamlessly to deliver an interactive user experience and facilitate efficient data visualization and analysis.

'aws-sdk' library was also used to import special functions and objects to read and write data stored in DynamoDB. AWS Security was configured to generate secret access key and access key id for the root client. These items are used by the backend javascript code to connect with the AWS Account and issue commands and queries to AWS Cloud

The web application comprised three pages. The first page served as the login page, the second page handled user registration, and the third page displayed the visualized weather data. The third page also featured a logout button, which allowed users to end their session and delete their session data. Sessions were set to remain valid for 1 month, enabling users to access the third page without logging in again within that time period.

In the web application, a WebSocket object named 'socket' was created to reference the WebSocket API created in AWS API Gateway. This object was responsible for instructing the application regarding necessary actions. The application was instructed to connect to the WebSocket API and push weather data into arrays upon receiving messages from the WebSocket API. If the application detected that the connection to the WebSocket API had closed, it was automatically instructed to reconnect.

To convert the weather data from JSON to CSV format, the 'json-to-csv-export' library was employed. Users could download the weather data as a CSV file by clicking a button provided on the third page. This feature enabled users to analyze the weather data using data analysis applications like RStudio, which cannot directly access data stored in DynamoDB.

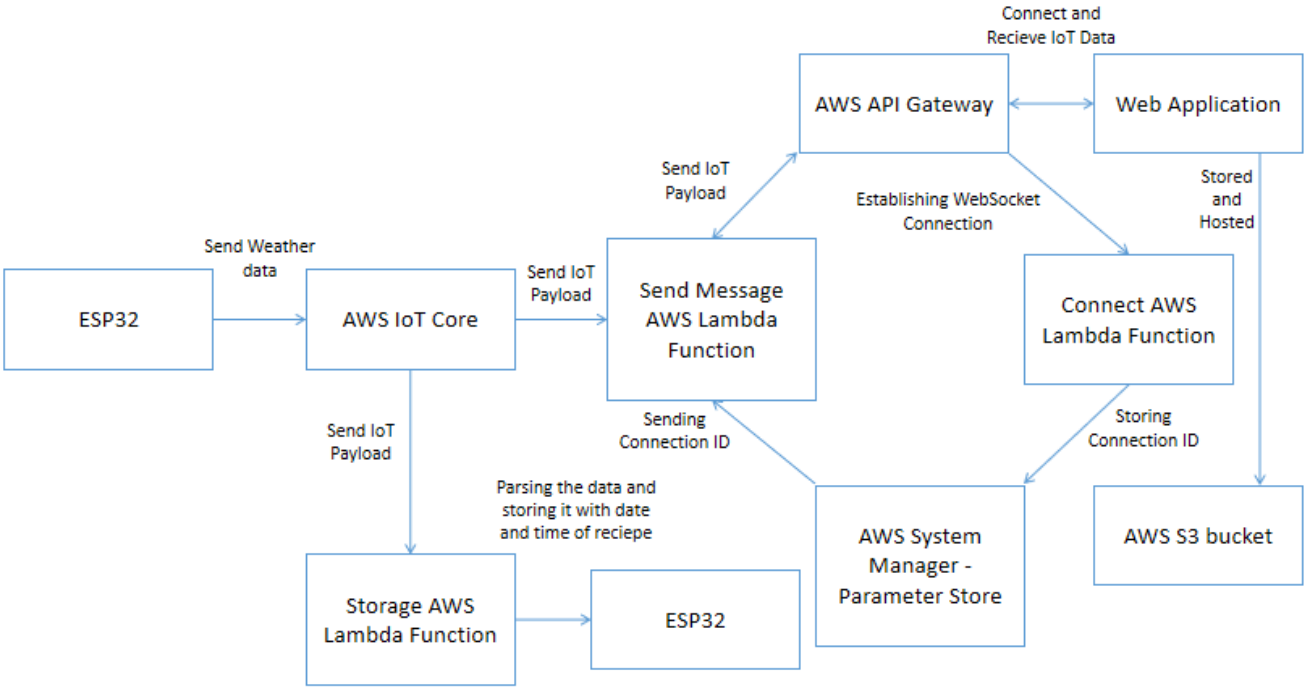
During user registration, the application securely encrypted the passwords using bcrypt.js. The encrypted user data, including the password, was then transmitted and stored in AWS DynamoDB for enhanced security. When users attempted to log in, the entered password was compared with the decrypted password for the corresponding email. If the passwords matched, the user was granted access. Otherwise, login was denied. The secret key for encryption and decryption was stored in a .env file, which remained inaccessible to users while the application was running.

To facilitate hosting the web application in a browser, we utilized the browserify library. This powerful tool allowed us to compile our JavaScript code, which originally could only run on the client-side, into a format that could be executed by the client's browser. By transforming our code using browserify, we enabled seamless execution of the JavaScript code within the web application, ensuring compatibility across different browsers and enhancing the overall user experience. This compilation process empowered us to deploy our web application in a browser environment, delivering its full functionality to end-users.

After ensuring the IoT system functioned flawlessly, optimizations were implemented for the ESP32 code. The optimizations included instructing the ESP32 to disconnect from WiFi immediately after sending weather data to AWS IoT Core and reconnecting with AWS IoT Core using the MQTT protocol after a 2-minute interval. This process was repeated indefinitely, resulting in substantial savings in networking resources.

Upon the completion of the internship, the third page would be publicly hosted as a static application in an AWS S3 bucket. This decision was made due to the cost and maintenance considerations associated with hosting and managing a full-stack Node.js application.

System Architecture



Hardware Used:

- ESP32
- USB Type C Cable

Software Used:

- Arduino IDE
- AWS
- VS Code Editor

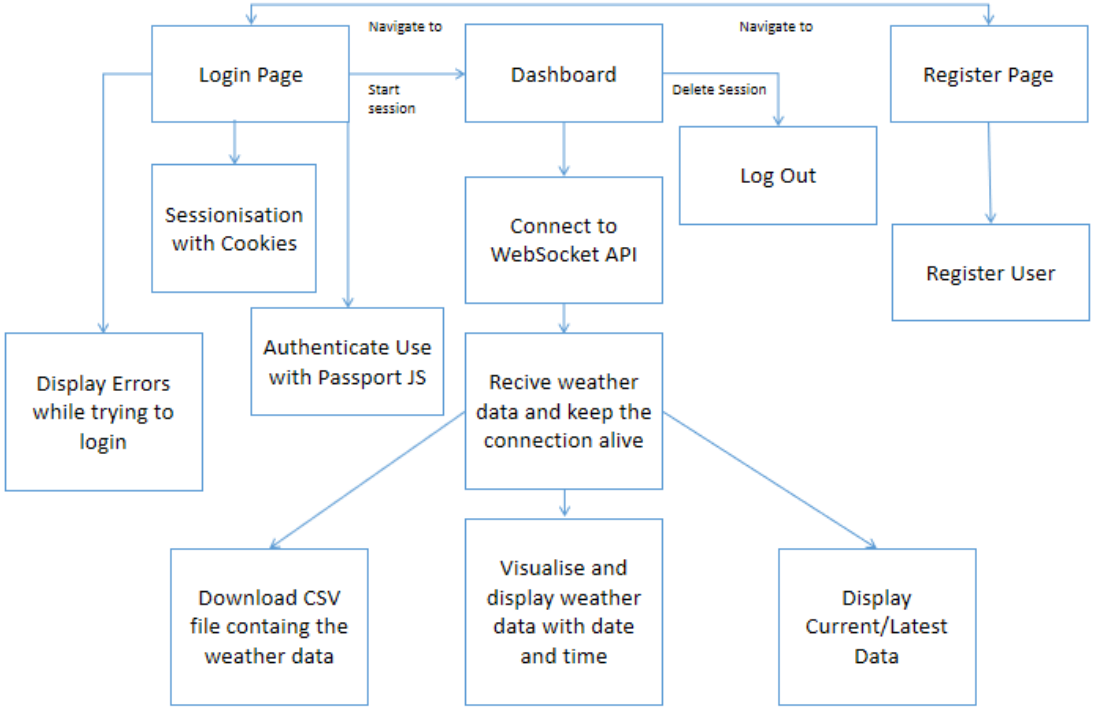
AWS Cloud services used:

- AWS Lambda
- AWS API Gateway
- AWS IoT Core
- AWS DynamoDB
- AWS System Manager (For parameter storage)
- AWS IAM
- AWS S3
- AWS CloudWatch

Programming Language Used:

- C++
- Python

Full Stack Architecture



Programming Languages used:

- JavaScript
- HTML
- CSS

JavaScript Libraries Used:

- Passport JS
- Bcrypt JS
- HighCharts JS
- jQuery
- aws-sdk
- json-to-csv-export
- browserify

JavaScript Frameworks used:

- Express JS
- Node JS

The login page of the web application features a user-friendly form with two labels, corresponding to the email address and password input fields, and a login button. In case of any errors during the login process, the form dynamically displays an error message to assist the user in identifying and resolving the issue. Additionally, the login page provides a convenient link to the Registration page, allowing users to easily navigate to the registration process. Upon successful login, users are automatically redirected to the Dashboard page, which serves as the main interface of the application.

The Registration page presents a similar form with two labels for the email address and password input fields, accompanied by a Registration button. Once users successfully register their credentials, they are automatically redirected to the login page to proceed with logging into the web application.

The security and privacy of user data are prioritized in the system. To achieve this, we store the users' data in a dedicated table within DynamoDB, ensuring proper segregation and organization of the information. Importantly, user passwords are stored in an encrypted format to enhance data protection.

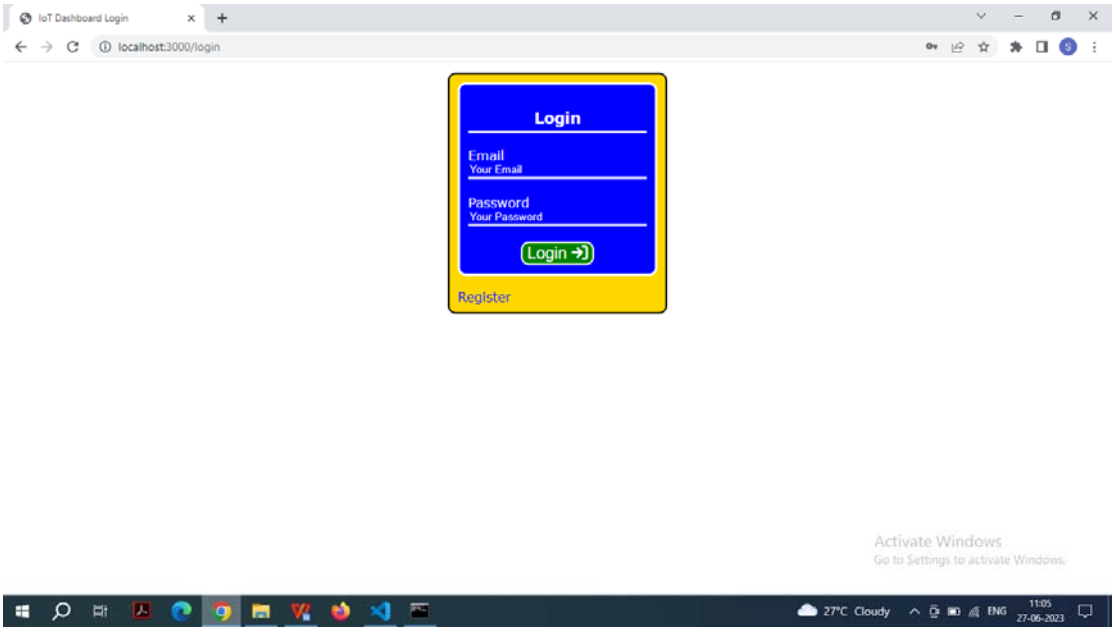
When a user attempts to log in, the system performs a secure authentication process. The provided password is compared with the decrypted password stored in DynamoDB. The encryption and decryption operations are conducted on the server-side, using the robust security features offered by Passport.js. This framework effectively manages the encryption and decryption of passwords, safeguarding sensitive user information from unauthorized access or potential breaches.

By employing encryption techniques and relying on Passport.js for password management, we maintain a high level of security and confidentiality for user data throughout the login process. These measures ensure that user credentials are adequately protected and provide peace of mind to our users regarding the safety of their personal information.

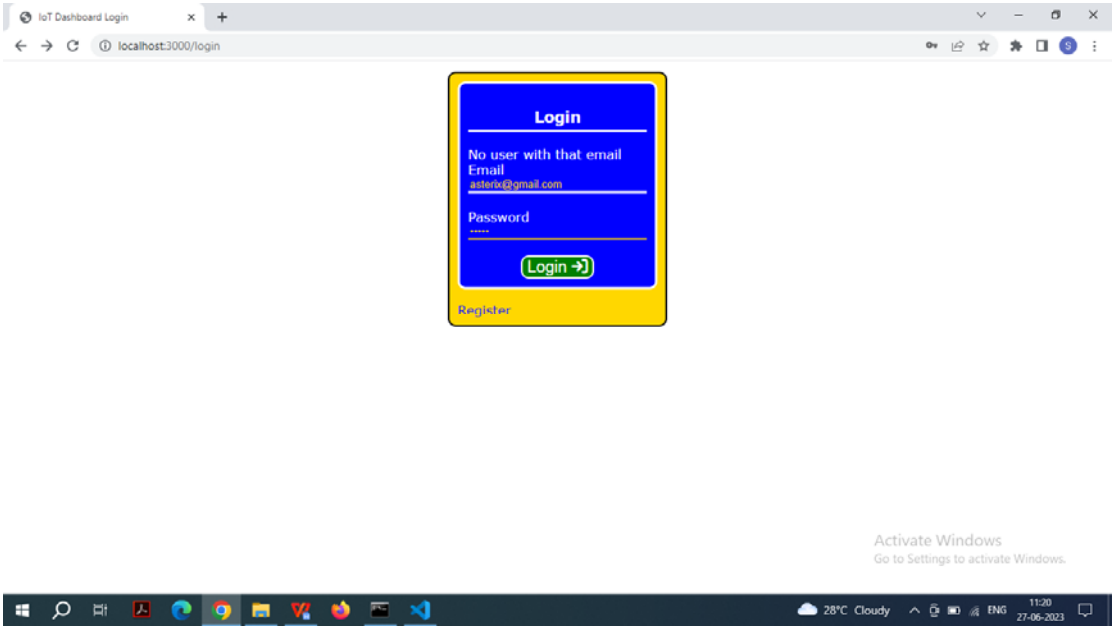
The Dashboard page is where the visualized weather data is displayed. Users can observe various charts and graphs representing the weather data. Moreover, the page features a dedicated button for users to download the weather data in CSV format, facilitating further analysis using external data analysis applications. To ensure the security of user sessions, a logout button is also available, enabling users to terminate their session within the web application. After successfully logging out, users are automatically directed back to the login page.

Users have uninterrupted access to the Dashboard page within a 1-month period, provided they do not manually log out. To enhance security, an auto logout feature is implemented, automatically terminating a user's session after 5 minutes of inactivity.

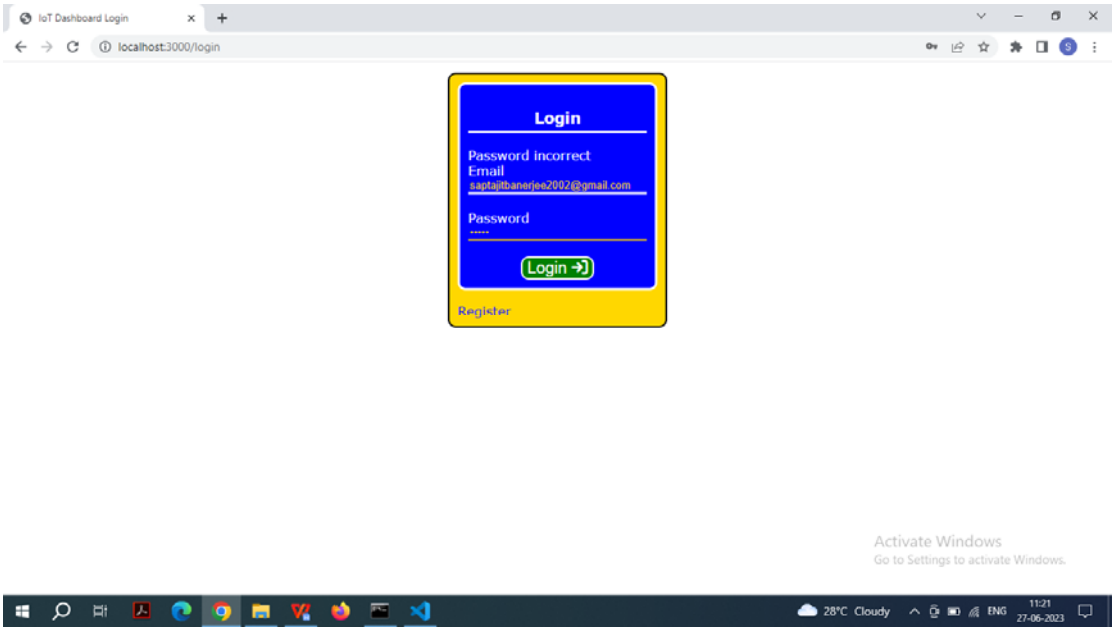
The Dashboard page initially visualizes the stored weather data from DynamoDB when the page is loaded. Subsequently, real-time weather data sent from the ESP32 device is continuously visualized in the web application, while simultaneously being stored in DynamoDB. In the event that the web application fails to plot certain weather data points, users can rest assured that the complete dataset is securely stored in DynamoDB. When the page is reloaded, users will be able to view the previously missed weather data points alongside the rest of the visualized data, ensuring a comprehensive and reliable representation of the weather information.



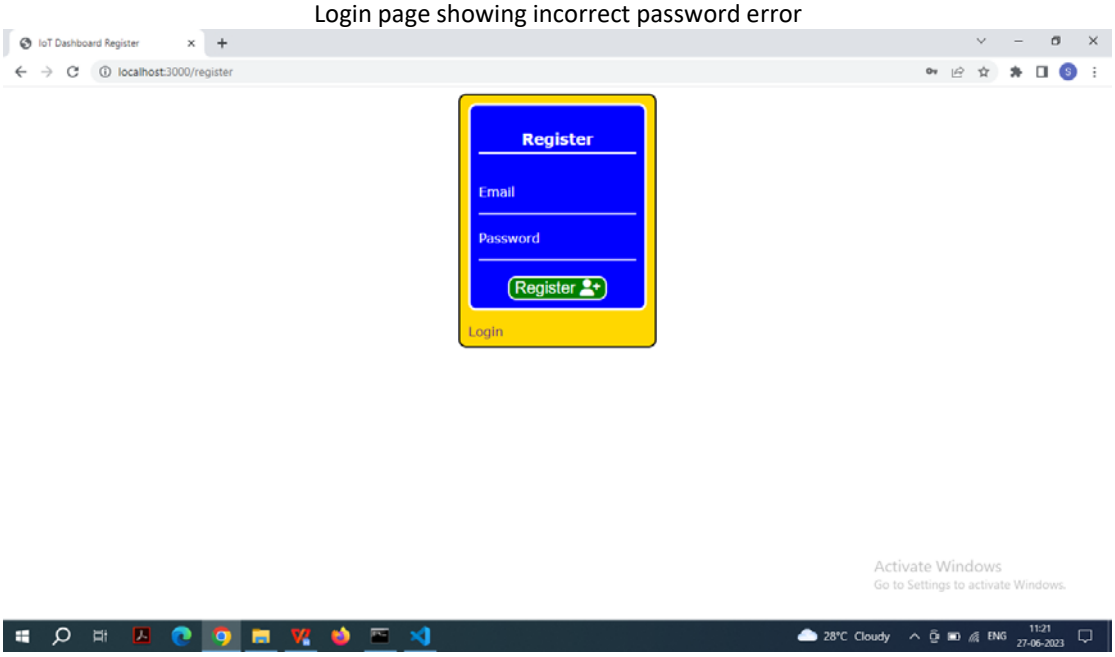
Login Page



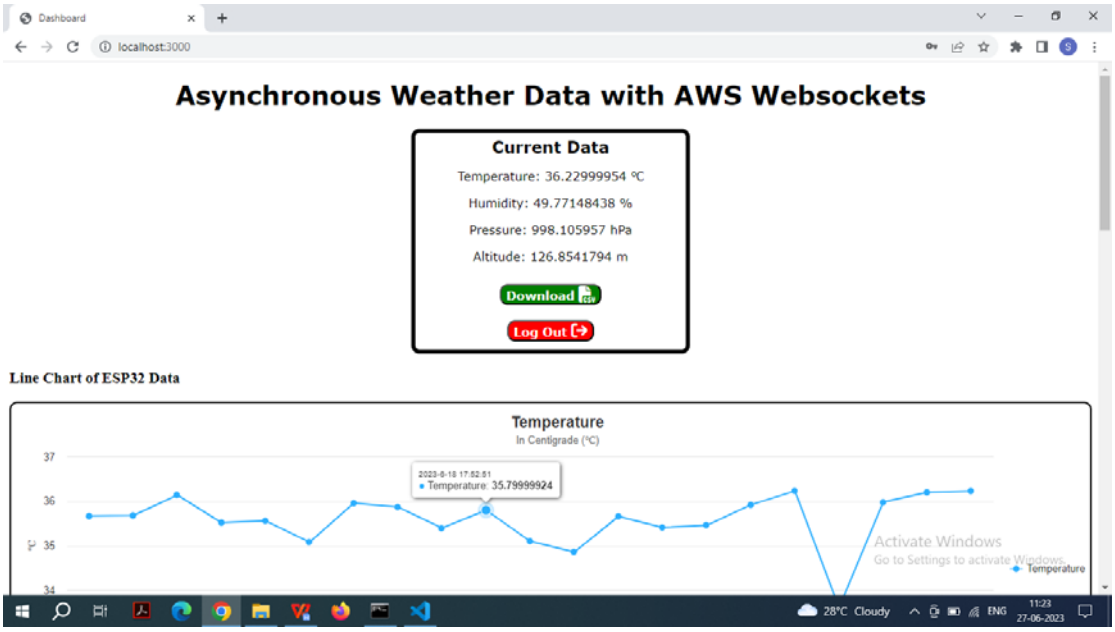
Login Page showing invalid email error



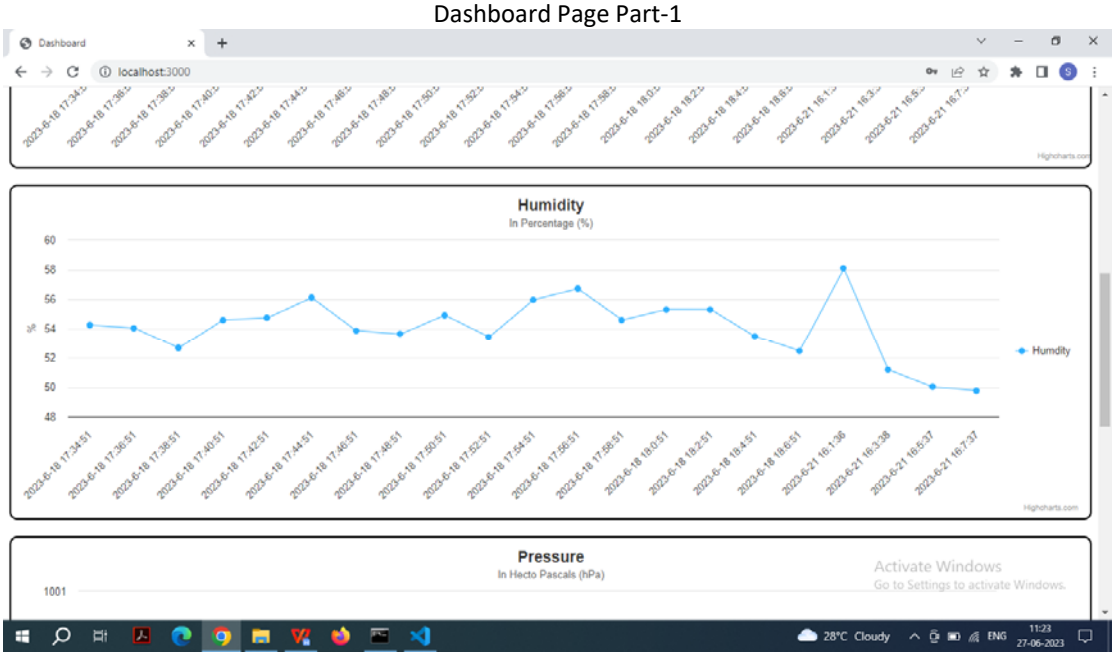
Login page showing incorrect password error



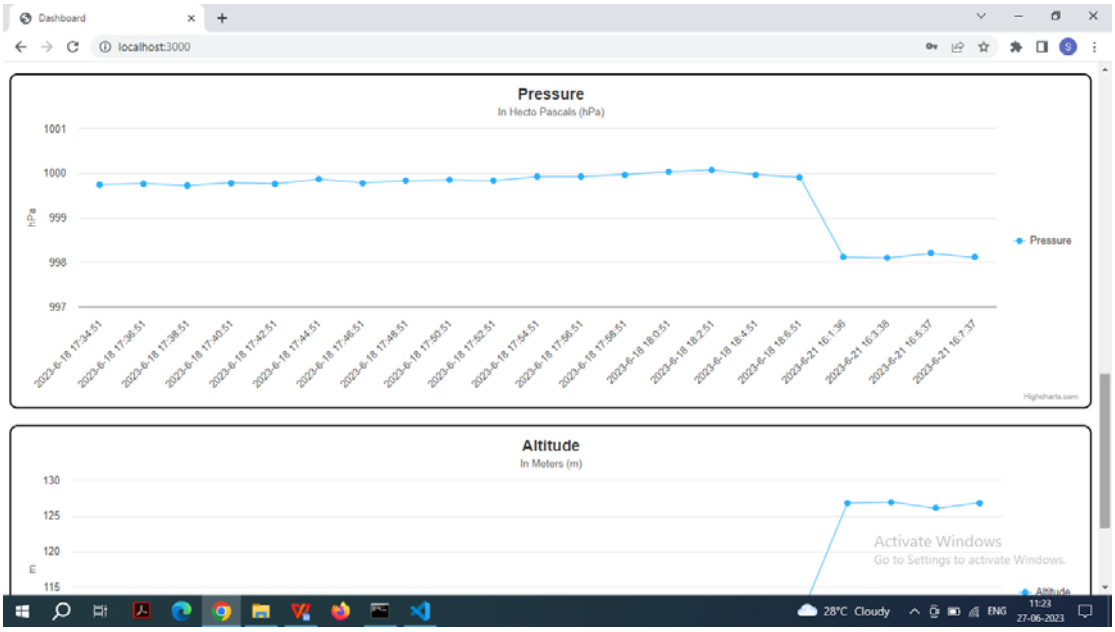
Register Page



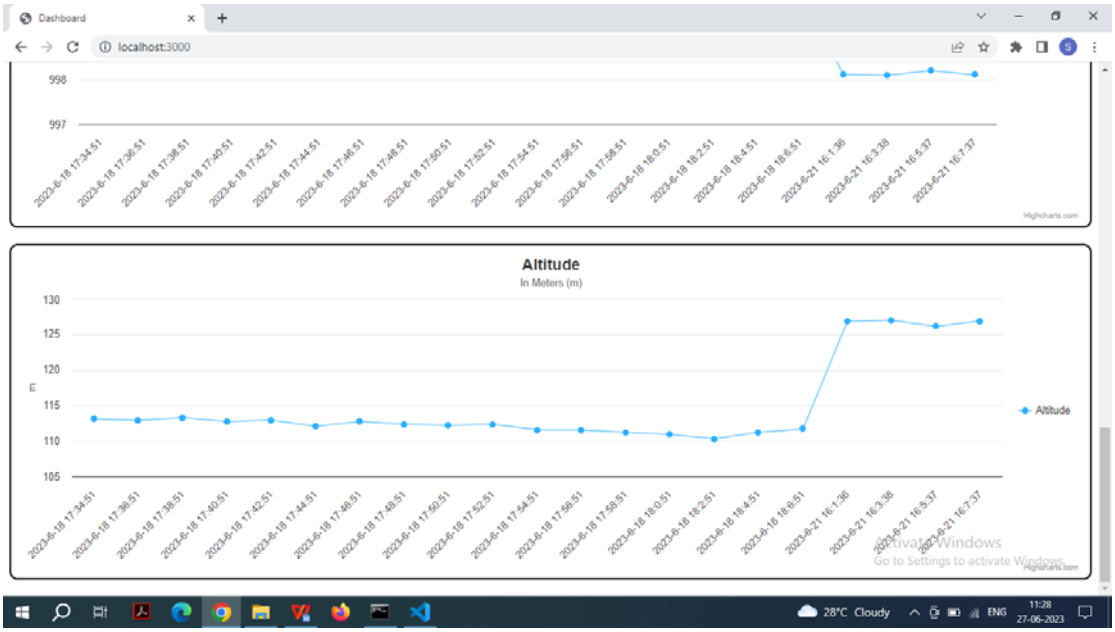
Dashboard Page Part-1



Dashboard Page Part 2



Dashboard Page Part 3



Dashboard Page Part 4

Results and Findings

Link to Dashboard Page hosted in AWS S3 Bucket : https://esp32-s3-tih.s3.ap-south-1.amazonaws.com/Dynamo+DB/hosting/index.html

Link to the directory of project in GitHub:
https://github.com/saptajitbanerjee/saptajitbanerjee-AWS-IoT-Weather-Monitoring-System

Successful Integration of ESP32 with AWS IoT Core: The ESP32 device was successfully configured to connect to AWS IoT Core using MQTT protocol. The device was registered as a single “Thing" in AWS IoT Core, and the necessary certificates and keys were generated for secure authentication. The ESP32 was able to establish a stable and reliable connection with AWS IoT Core, enabling seamless transmission of temperature data.

Real-time Weather Data Visualization: The web application effectively visualized real-time weather data received from the ESP32 device. HighCharts.js was utilized to create interactive line charts, providing users with a comprehensive representation of temperature trends over time. The JavaScript code facilitated the smooth rendering of weather data on the web application's frontend, enhancing the user experience.

User Authentication and Session Management: Passport.js and Express.js played crucial roles in implementing user authentication and session management functionalities. Users were able to register their email addresses and passwords securely, with the passwords being encrypted using bcrypt.js. The login process validated user credentials, granting access to the Dashboard page upon successful authentication. The web application successfully maintained user sessions and implemented an auto logout feature to enhance security.

Data Storage and Retrieval: AWS DynamoDB was utilized as the database to store weather data received from the ESP32 device. The data was stored with timestamps, enabling proper organization and retrieval. Users had the capability to download the weather data in CSV format, allowing for further analysis using external tools and applications. The integration with DynamoDB ensured reliable data storage and retrieval, even in cases where visualization on the web application might have encountered issues.

Optimization and Resource Efficiency: The project implemented several optimizations to enhance the performance and resource efficiency of both the ESP32 device and the web application. For the ESP32, optimizations were made to minimize power consumption and networking resources by disconnecting from WiFi immediately after data transmission and reconnecting after a specified interval. In the web application, efforts were made to streamline the JavaScript code, improve memory utilization, and optimize the WebSocket connection to minimize networking resources.

Overall, the project demonstrated the successful implementation of an IoT system for real-time weather data collection, transmission, visualization, and storage. The findings highlighted the effectiveness of the integrated components, the reliability of the data storage solution, and the user-friendly nature of the web application. The project's optimizations showcased the potential for resource savings and performance improvements, enhancing the overall efficiency of the system.

Discussion

The project successfully implemented an Internet of Things (IoT) system using an ESP32 S2 microcontroller, AWS IoT Core, and a web application. The ESP32 was programmed to read temperature data from the BME280 sensor and transmit it to AWS IoT Core using MQTT protocol. The AWS IoT Core acted as the central hub for receiving and processing the data. The web application, developed using JavaScript, jQuery, Express.js, Passport.js, HighCharts.js, HTML, and CSS, enabled users to visualize and interact with the real-time weather data.

One of the key findings of the project is the successful integration of the ESP32 with AWS IoT Core. The MQTT protocol provided a reliable and efficient communication channel for transmitting the temperature data to the cloud. The use of certificates and policies ensured secure and authenticated connections between the ESP32 and AWS IoT Core, safeguarding the data transfer.

The web application demonstrated effective visualization of the weather data using HighCharts.js. The line charts provided a clear representation of temperature trends over time. The use of jQuery allowed for the display of real-time temperature data, keeping users updated with the latest readings. The session management and user authentication implemented through Express.js and Passport.js ensured secure access to the web application.

In order to further enhance the performance of the web application, additional optimizations can be implemented. Two key optimizations have been identified.

The first optimization involves instructing the JavaScript code to disconnect immediately after receiving the weather data and then reconnecting after a specified interval, such as every 2 minutes. By allowing a short window of 5 seconds for the connection to remain alive, the web application can efficiently receive the weather data within this time-frame. This optimization significantly conserves networking resources and contributes to improved overall performance.

The second optimization eliminates the need for an additional array dedicated solely to the conversion of weather data from JSON format to CSV format. Instead, the JavaScript code can be modified to retrieve the data directly from the global variable arrays where the weather data is stored. This data can then be pushed into a local variable array, which is automatically destroyed by JavaScript upon completion of the CSV file download. This optimization effectively minimizes memory usage and optimizes resource allocation within the application.

Furthermore, an additional improvement that can be implemented is the inclusion of an auto log out feature. This feature automatically logs out the user after a specified period of inactivity. By implementing this auto log out feature, the web application's security capabilities are strengthened, ensuring that user sessions are appropriately managed and protected.

Conclusion

In conclusion, this project successfully implemented an Internet of Things (IoT) system for real-time weather data monitoring and visualization. The integration of the ESP32 S2 microcontroller with AWS IoT Core and the development of a web application provided an efficient and user-friendly solution.

The ESP32 S2 microcontroller effectively read temperature data from the BME280 sensor and transmitted it to AWS IoT Core using MQTT protocol. The secure connection established through certificates and policies ensured the confidentiality and integrity of the data during transmission.
The web application, built using JavaScript, jQuery, Express.js, Passport.js, HighCharts.js, HTML, and CSS, offered a visually appealing and interactive platform for users to access and analyze the weather data. HighCharts.js facilitated the creation of line charts, enabling users to observe temperature trends over time. The integration of jQuery provided real-time updates of the latest temperature readings, enhancing the user experience.

Key findings from the project include the successful integration of the ESP32 with AWS IoT Core, the efficient transmission of data using MQTT protocol, and the effective visualization of weather data through the web application.

Future improvements were identified to optimize the system further. These include implementing disconnect-reconnect mechanisms to save networking resources, eliminating the need for separate arrays during data conversion, and introducing an auto log out feature for enhanced security.

Overall, this project demonstrated the successful implementation of an IoT system for real-time weather data monitoring and visualization. The combination of the ESP32 S2 microcontroller, AWS IoT Core, and the web application showcased the potential of IoT technologies in collecting, transmitting, and visualizing data. The project's outcomes contribute to the field of IoT and provide valuable insights for future developments in weather monitoring systems and related applications.

References

- vladak. (n.d.). shield. GitHub. Retrieved from https://github.com/vladak/shield
- How To Electronics. (2021, December 29). Getting Started with Amazon AWS IoT Core using ESP32 || Creating Thing, Policy & Certificates [Video]. Youtube. https://www.youtube.com/watch?v=idf-gGXvIu4&t=185s
- Cumulus Cycles. (2022, December 17). Store data sent to AWS IoT in DynamoDB using Lambda [Video]. YouTube. https://www.youtube.com/watch?v=0RcVwTKSbSA
- Steve on IoT (Stephen Borsay). (2022, January 8). AWS Serverless IoT [Video][Playlist].YouTube. https://www.youtube.com/watch?v=MsyzeXMu23w&list=PLTUPrJfBdlQC8Bq9nV10FYStgol9mGl34
- CodeSpace. (2018, January 10). Read and Write to DynamoDB using NodeJs (aws-sdk) [Video]. YouTube. https://www.youtube.com/watch?v=SU4dZ-qgR1Y
- CodeSpace. (2017, December 31). Create multiple users under one account with different permissions (AWS cloud) [Video]. YouTube. https://www.youtube.com/watch?v=m5nCqLPwSsk
- npm. (n.d.). json-to-csv-export. npm. Retrieved from https://www.npmjs.com/package/json-to-csv-export
- Web Dev Simplified. (2019, July 13). Node.js Passport Login System Tutorial [Video]. YouTube. https://www.youtube.com/watch?v=-RCnNyD0L-s
- GeeksforGeeks. (2023, 20th March). How to Expire Session After 1 Min of Inactivity in Express Session of Express.js. Available at: https://www.geeksforgeeks.org/how-to-expire-session-after-1-min-of-inactivity-in-express-session-of-express-js/
- HTML JS Templates (Siva Kishore G). (2023, 21st March). User Inactive Auto Logout. Available at: https://htmljstemplates.com/html-js/user-inactive-auto-logout