IoT Temperature Analysis System/API

Preface

The system described herein represents a holistic integration of interconnected IoT (Internet of Things) devices and a sophisticated Spring Boot API, designed to monitor and transmit temperature and humidity data. At its core lies a powerful microcontroller, programmed with Arduino software, capable of interfacing with a DHT11 sensor to gather precise environmental data. This sensor plays a pivotal role in measuring temperature and humidity levels, crucial for applications spanning from climate control to environmental monitoring.

In conjunction with the IoT devices, the system relies on the Spring Boot API to facilitate seamless communication and data transmission. The API, leveraging the Spring framework, provides a robust platform for handling incoming requests, processing data, and interacting with external systems. It encapsulates business logic, implements security mechanisms, and ensures reliable data exchange between IoT devices and remote servers.

The setup process involves configuring both the IoT devices and the Spring Boot API. While the IoT devices are programmed to connect to a designated WiFi network and continuously monitor environmental conditions, the Spring Boot API orchestrates the reception, processing, and storage of incoming sensor data. Through intelligent data transmission strategies and efficient utilization of network resources, the system optimizes power consumption and network bandwidth, ensuring sustainable operation.

Throughout this document, we will explore the intricate interplay between the IoT ecosystem and the Spring Boot API, examining their architecture, functionalities, and potential applications. By gaining insights into the seamless integration of IoT technology with modern software frameworks like Spring Boot, readers will be equipped to harness the transformative potential of IoT across diverse domains, from smart home automation to industrial monitoring systems.

Java SpringBoot API Documentation

The provided API offers a comprehensive suite of functionalities for managing devices, user authentication, and temperature records within a system, all while emphasizing robust security measures and data integrity.

At its core, the Device Controller streamlines device management operations, providing endpoints for retrieving, adding, updating, and deleting device information. Whether users need to access device details collectively or by specific device ID, the API accommodates these needs through endpoints like GET /devices and GET /devices/{id}. Additionally, administrators can create new devices (POST /devices), modify existing ones (PUT /devices/{id}), and remove devices from the system (DELETE /devices/{id}).

The Jwt Authentication Controller enhances security by enabling user authentication and token generation through POST /authenticate. Importantly, access to authentication tokens necessary for executing PUT and DELETE operations is restricted to elevated admin users, configurable within the application.properties. This measure ensures that only authorized personnel can perform critical data modifications, thereby bolstering system security and safeguarding against unauthorized access.

The Temperature Controller facilitates the management of temperature records with a range of functionalities. Users can retrieve all temperature records (GET /temperatures), add new records (POST /temperatures), update existing ones (PUT /temperatures/{id}), and delete records (DELETE /temperatures/{id}). Additionally, advanced features such as retrieving average, maximum, minimum, and median temperatures and humidity levels, along with filtering options based on parameters like day, device name, hour, location, month, and year, empower users to analyze temperature data effectively according to their specific requirements.

Overall, the API presents a robust solution for device management, secure authentication, and temperature record handling within a system. Its emphasis on stringent access controls, configurable admin user privileges, and comprehensive data management functionalities underscores its suitability for deployment in environments where security, data integrity, and operational efficiency are paramount concerns.

Preliminary API Endpoint Description

Device Controller

GET /devices: Retrieves all devices.
POST /devices: Adds a new device.
GET /devices/{id}: Retrieves a device by ID.
PUT /devices/{id}: Updates a device by ID.
DELETE /devices/{id}: Deletes a device by ID.

JWT Authentication Controller

POST /authenticate: Creates an authentication token.

Temperature Controller

GET /temperatures: Retrieves all temperatures.
POST /temperatures: Adds a new temperature.
GET /temperatures/{id}: Retrieves a temperature by ID.
PUT /temperatures/{id}: Updates a temperature by ID.
DELETE /temperatures/{id}: Deletes a temperature by ID.
GET /temperatures/average: Retrieves the average temperature and humidity.
Query Parameters: day, deviceName, hour, location, month, year
GET /temperatures/filtered: Retrieves temperatures with filters.
Query Parameters: day, deviceName, hour, location, month, year
GET /temperatures/latest: Retrieves the latest temperature record.
GET /temperatures/max: Retrieves the maximum temperature and humidity.
Query Parameters: day, deviceName, hour, location, month, year
GET /temperatures/median: Retrieves the median temperature and humidity.
Query Parameters: day, deviceName, hour, location, month, year
GET /temperatures/min: Retrieves the minimum temperature and humidity.
Query Parameters: day, deviceName, hour, location, month, year

API Directory Structure

config/
  - Contains configuration files for the application, such as database configuration, security configuration, etc.

controller/
  - Houses the controller classes responsible for handling incoming HTTP requests and mapping them to appropriate service methods.

exception/
  - Contains custom exception classes used for error handling and exception management within the application.

model/
  - Holds the domain model classes representing entities within the application, such as Device, Temperature, etc.

repository/
  - Contains repository classes responsible for database interactions, including ORM CRUD operations and data access logic.

service/
  - Houses the service classes containing business logic and application-specific functionalities.

startup/
  - Includes startup or initialization classes responsible for bootstrapping the application, setting up initial configurations, etc.

TemperatureApplication.class
  - Main class of the application, serving as the entry point for Spring Boot. Contains the main method to launch the application.

This directory structure adheres to commonly used conventions in Java Spring Boot applications, promoting organization, modularity, and maintainability.

Software Stack Overview

Java Spring Boot

The project is built using Spring Boot, a popular Java framework for building web applications and microservices. Spring Boot simplifies the development process by providing conventions and out-of-the-box functionality for rapid application development.

Database

MySQL is used as the relational database management system (RDBMS) for storing application data. MySQL is a widely used open-source RDBMS known for its reliability, performance, and scalability.

Web Frameworks and Libraries

Spring Boot Starter Web: Provides essential dependencies for building web applications with Spring Boot, including embedded Tomcat server and Spring MVC.
Spring Boot Starter Security: Integrates Spring Security into the application, enabling features such as authentication and authorization.
Spring Boot Starter Data JPA: Simplifies data access using the Java Persistence API (JPA) with Spring Data repositories.
Spring Boot Starter HATEOAS: Adds support for implementing HATEOAS (Hypermedia as the Engine of Application State) in RESTful APIs.
Springfox Swagger: Enables API documentation using the Swagger UI, facilitating API exploration and testing.

JSON Web Token (JWT) Libraries

jjwt-api: Provides API for JSON Web Tokens (JWTs), enabling secure authentication and authorization mechanisms.
jjwt-impl: Implementation of JSON Web Tokens (JWTs) for runtime usage.
jjwt-jackson: Jackson JSON integration for JSON Web Tokens (JWTs).

Development and Testing Environment

Testing

H2 Database (Runtime Scope): Lightweight, in-memory database primarily used for testing and development purposes.
Springfox Swagger UI: Provides a user-friendly interface for exploring and testing RESTful APIs documented using Swagger.

Deployment Environment

Operating System: Vertically scalable Ubuntu server environment.
Application Deployment: Monolithic deployment architecture, where the entire application is deployed as a single, self-contained unit on the server.
Database: MySQL is used as the backend database management system.
Scalability: The deployment environment is vertically scalable, allowing for increased resource allocation (CPU, memory) to accommodate growing application demands.

This software stack and deployment environment combination provide a robust foundation for developing, deploying, and scaling Java Spring Boot applications with MySQL as the backend database.

The IoT Side: ESP32 and DHT11 Sensor

The IoT aspect described here involves an ESP32 microcontroller interfaced with a DHT11 sensor to monitor temperature and humidity data. The code snippet provided demonstrates how the ESP32 device connects to a Wi-Fi network and periodically sends temperature and humidity readings to a designated server endpoint via HTTP requests.

Components Used

- **ESP32 Microcontroller:** A powerful Wi-Fi and Bluetooth-enabled microcontroller known for its versatility and capabilities, suitable for IoT applications.
- **DHT11 Sensor:** A basic digital temperature and humidity sensor capable of providing accurate readings in various environments.

Code Overview

Library Inclusions

The code includes necessary libraries such as Arduino.h, WiFi.h, HTTPClient.h, and dht11.h to facilitate communication with the ESP32, Wi-Fi connectivity, HTTP requests, and DHT11 sensor interfacing, respectively.

Wi-Fi Initialization

The initWiFi() function initializes the ESP32's Wi-Fi connection, configuring it to operate in station (STA) mode and connecting to a specific Wi-Fi network using the SSID and password provided.

Setup Function

The setup() function is called once during the initialization phase. It initializes the serial communication for debugging purposes and establishes the Wi-Fi connection using the initWiFi() function. Additionally, it prints the received signal strength indicator (RSSI) to the serial monitor.

Loop Function

The loop() function is the main execution loop of the program and continuously runs after the setup phase.
It reads temperature and humidity values from the DHT11 sensor using the DHT11.read() function.
If the Wi-Fi connection is established (WL_CONNECTED), it creates an HTTP client object (HTTPClient http) and sends a POST request to the specified server endpoint (http://homeserver:8081/temperatures/).
The payload of the POST request contains temperature and humidity data along with device information in JSON format.
The HTTP response code is checked, and if the request is successful (HTTPCODECREATED), the response payload is printed to the serial monitor.
Error handling is performed in case of failed HTTP requests.

Delay Function:

The delay() function introduces a delay of 5 minutes (5 * 60 * 1000UL) between consecutive HTTP requests to avoid flooding the server with requests and conserve power.

Deployment

The ESP32 device is deployed in the desired environment, such as a room or outdoor location, where temperature and humidity monitoring is required.
The ESP32 connects to a Wi-Fi network to establish internet connectivity and communicate with the server.
The server endpoint (http://homeserver:8081/temperatures/) receives the HTTP POST requests containing temperature and humidity data, storing it for further processing or analysis.

Wrap-Up

This IoT solution provides a simple yet effective way to monitor temperature and humidity data remotely using the ESP32 microcontroller and DHT11 sensor, enabling real-time environmental monitoring and data collection for various applications such as home automation, weather monitoring, and industrial automation.

Final Overview

In summary, the IoT solution presented here seamlessly integrates the power of an ESP32 microcontroller with a DHT11 sensor for remote environmental monitoring. Alongside this hardware component, a robust Spring Boot API serves as the backbone for receiving, processing, and storing the transmitted data. The API, built on Spring Boot, offers a scalable and reliable platform for handling incoming requests, ensuring data integrity, and facilitating real-time analysis.

The API architecture follows RESTful principles, providing clear and intuitive endpoints for interacting with the system. Through HTTP requests, clients can retrieve temperature and humidity data, as well as perform CRUD operations on devices and temperature records. Exception handling mechanisms ensure robustness, with graceful error handling to maintain system stability.

Utilizing Spring Security, the API implements authentication and authorization mechanisms, ensuring that only authorized users can access sensitive endpoints and perform privileged operations. JSON Web Tokens (JWT) are commonly used for stateless authentication, enhancing security while maintaining scalability.

API documentation is an integral part of the system, providing clear guidelines on endpoint usage, request parameters, and response formats. Tools like Swagger or Springfox are often employed to automatically generate comprehensive API documentation, enabling developers to explore and interact with the API effortlessly.

In conclusion, this integrated IoT solution offers a practical and efficient means of remotely monitoring environmental conditions. By leveraging the capabilities of the ESP32 microcontroller and the robustness of the Spring Boot API, organizations and individuals can gain valuable insights into their surroundings, enabling informed decision-making and enhancing operational efficiency across various domains. Whether deployed in residential, commercial, or industrial settings, this IoT solution provides a versatile platform for real-time environmental data monitoring and management, supported by a resilient and scalable API infrastructure.