# Backend

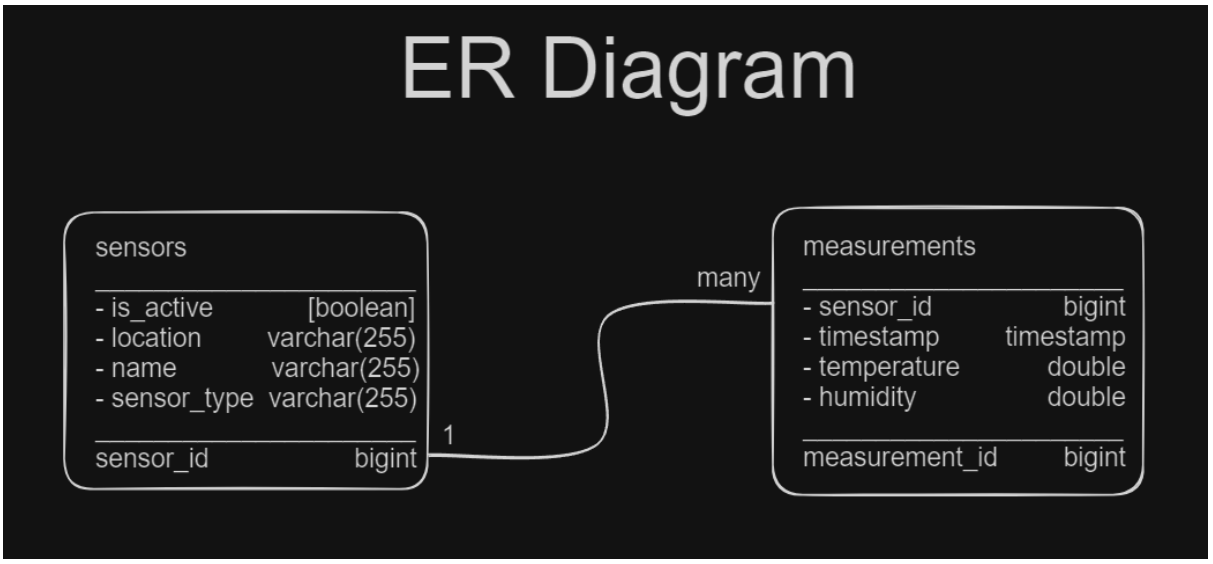
Welcome to our comprehensive report on the development of a RESTful web service using microservice architecture. This document serves as a detailed overview of our journey in creating a backend system capable of handling sensor data with persistence, efficiency, and scalability. Inside, you'll find a thorough exploration of our architectural choices, the technological hurdles we overcame, and the design decisions that shaped our project.

We begin with an entity-relationship diagram that maps out the database structure, followed by an architecture diagram that visually represents our setup, complete with component labels, IP addresses, and ports. Accompanying these diagrams is a technical description that delves into the intricacies of our development process, the challenges we faced, and the solutions we employed.

Furthermore, we have included a series of test calls for each operation to demonstrate the functionality of our system, complemented by screenshots that showcase our working setup. The source code, packaged as a ZIP file, is also provided for a more in-depth understanding of our implementation.

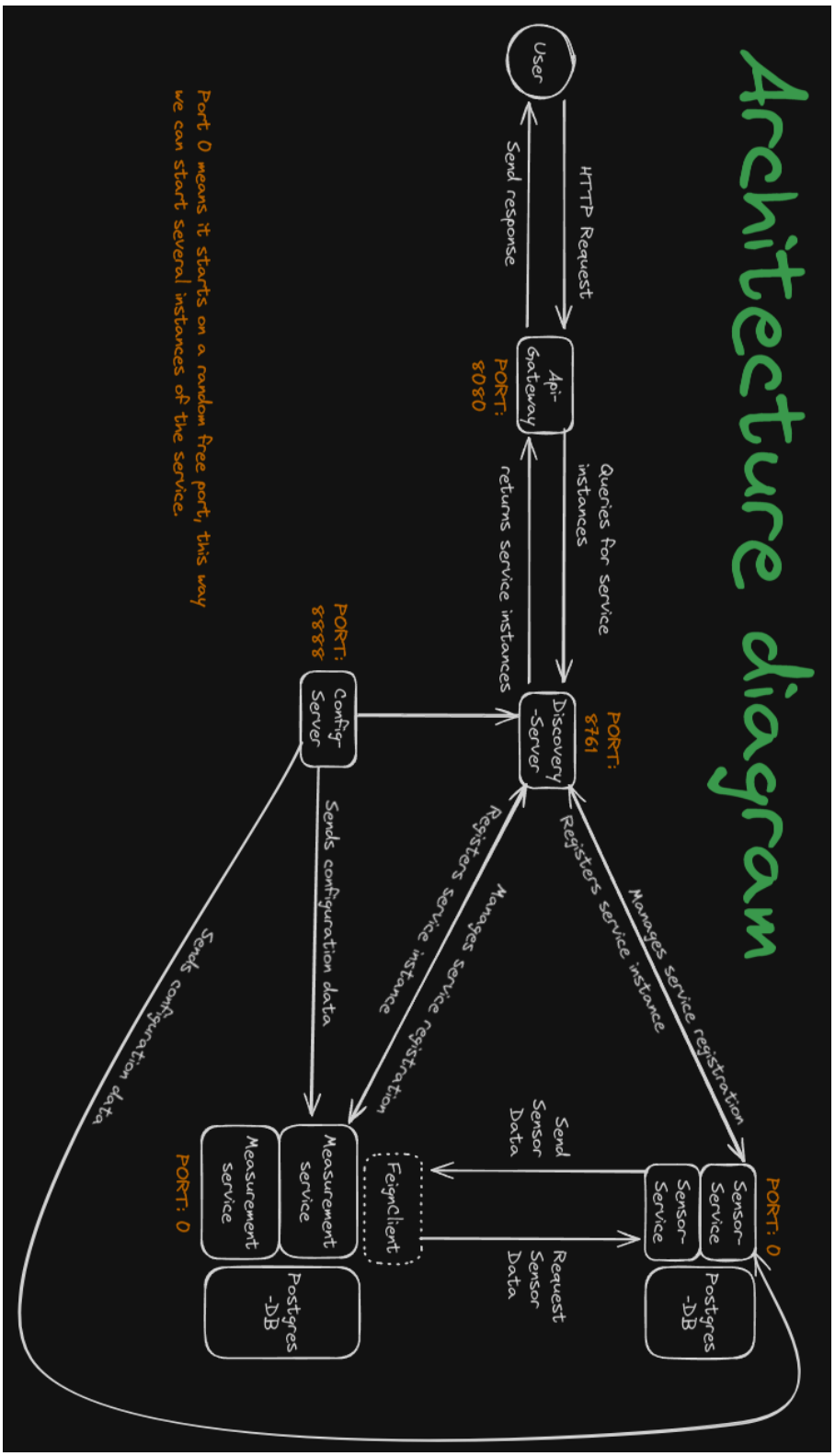
This document not only represents our technical journey but also serves as a testament to our learning process, highlighting our approach to problem-solving and innovation in the realm of backend development.

## ER Diagram:


ER Diagram
sensors
- is_active [boolean]
- location varchar(255)
- name varchar(255)
- sensor_type varchar(255)
sensor_id bigint
1
many
measurements
- sensor_id bigint
- timestamp timestamp
- temperature double
- humidity double
measurement_id bigint

## Architecture diagram:

Here you can find out architectural diagram, you find a hq version in the zip file in the test-data folder with other resources.



# Technical Report

## Introduction

This report outlines our experiences and challenges in implementing a RESTful web service using a microservice architecture. The service was designed for persistent storage and access of sensor data, contributing to a larger project.

## Technologies and Database

We initially employed MongoDB for its flexibility but faced complexities that led us to switch to PostgreSQL. This change was influenced by the ease of ID management in PostgreSQL, a crucial factor for our project's database schema.

## Initial Steps and Service Division:

Our journey began with uncertainty about where to start. We chose to first develop the API Gateway as it is the primary interface for user interaction. This step led to the creation of two distinct services: a sensor-service and a measurement-service. Though we later realized splitting the services wasn't mandatory, this structure seemed to fit our understanding of the project's requirements.

## Understanding Service Relationships:

One significant challenge was grasping how the sensor and measurement services were interconnected. This understanding was crucial for developing a coherent microservice architecture.

## Service Registry and Maven Hurdles:

After some progress, we decided to implement the Service Registry before the API Gateway, as this appeared to streamline subsequent development. However, a major obstacle we encountered was with Maven. The complexity of managing dependencies and versions was overwhelming, often leading to frustration and demotivation. Despite following numerous guides and seeking help from experienced Java backend developers, we faced persistent issues. This led to a complete redo of the project using Spring Initializr, which surprisingly resolved many earlier problems.

## Architecture and Design

Our microservice architecture comprises an API Gateway, Load Balancer, Service Registry, Config Server, and two instances of the web service. This structure ensures scalability and efficient load distribution. The database schema was meticulously designed to effectively manage sensor and measurement data.

## Challenges and Insights

The most significant challenges were understanding the microservices architecture, managing Maven dependencies, and grappling with technology choices. These experiences underscored the importance of selecting appropriate technologies and the potential need to restart a project with new insights for better outcomes.

## Conclusion

Despite the challenges, particularly with Maven, the project was a valuable learning experience. It reinforced the importance of adaptability and in-depth research in software development. The current state of the project, while not fully realizing our initial vision of containerization due to time constraints, meets the fundamental requirements and provides a solid foundation for future enhancements.

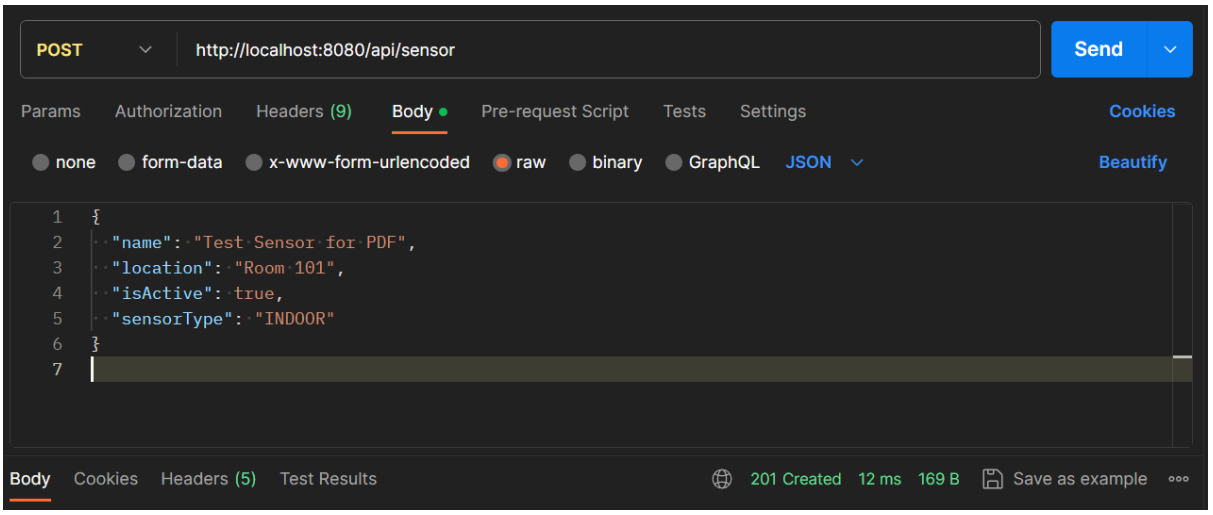
# TEST Calls

In the test-data-and-resources folder in the zip file you can find a postman collection with the test, for ease of testing. So please feel free to import that and use it!

Sensor Service API Documentation

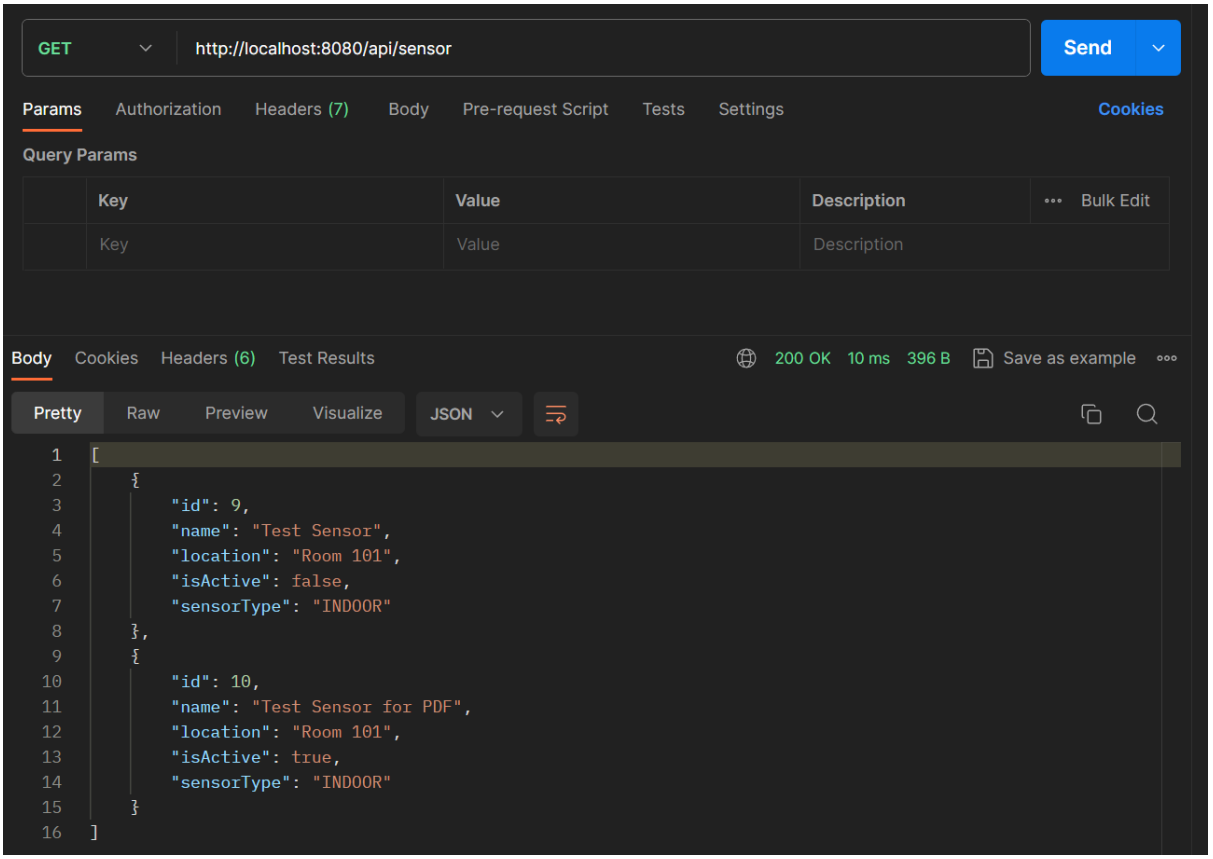
## 1. Create Sensor

- Method: POST
- URL: /api/sensor
- Request Body:
    - name: String (Name of the sensor)
    - location: String (Location of the sensor)
    - isActive: Boolean (Activation status of the sensor)
    - sensorType: SensorType (Type of the sensor, e.g., outdoor, indoor, water)
- Response:
    - Status: 201 Created
    - Body: None (Acknowledgement of sensor creation)



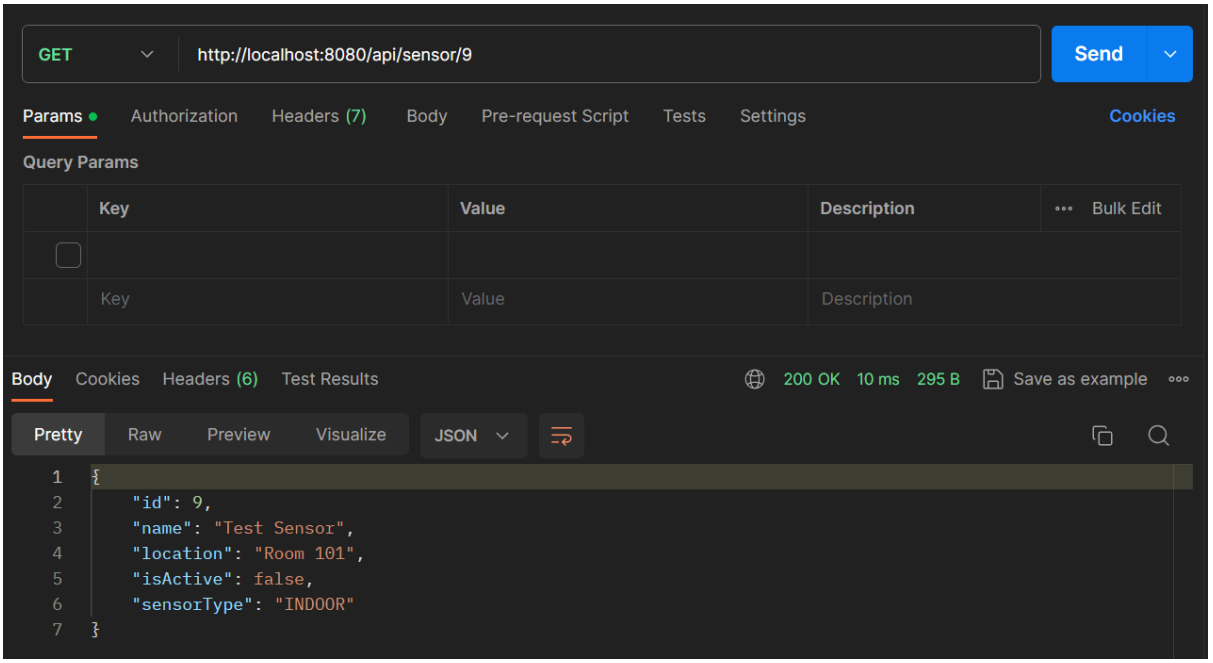
## 2. Get All Sensors

- Method: GET
- URL: /api/sensor
- Response:
    - Status: 200 OK
    - Body: List of SensorResponse containing details of all sensors



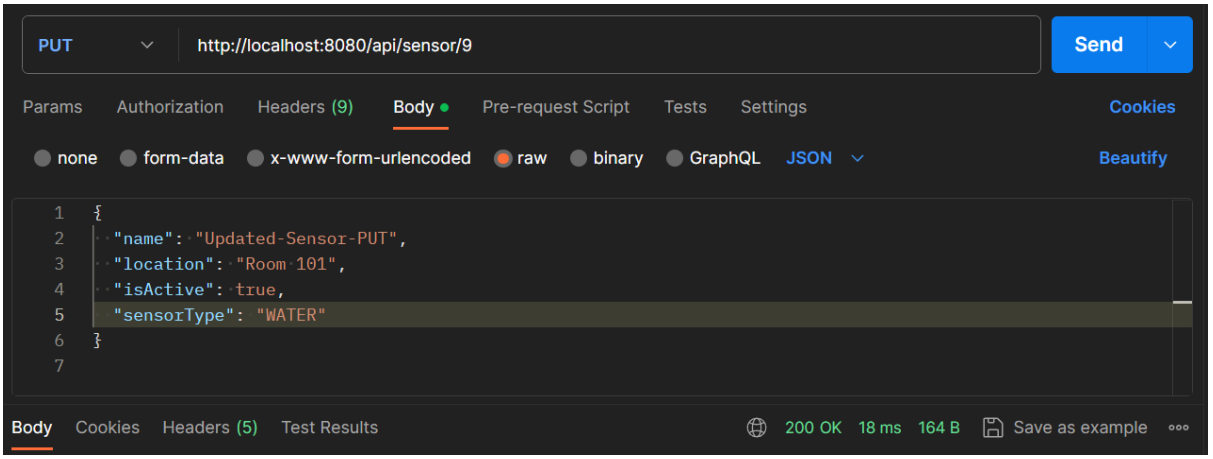
### 3. Get Sensor by ID

- Method: GET
- URL: /api/sensor/{id}
- Path Variable: id (ID of the sensor)
- Response:
    - Status: 200 OK
    - Body: SensorResponse containing details of the specified sensor



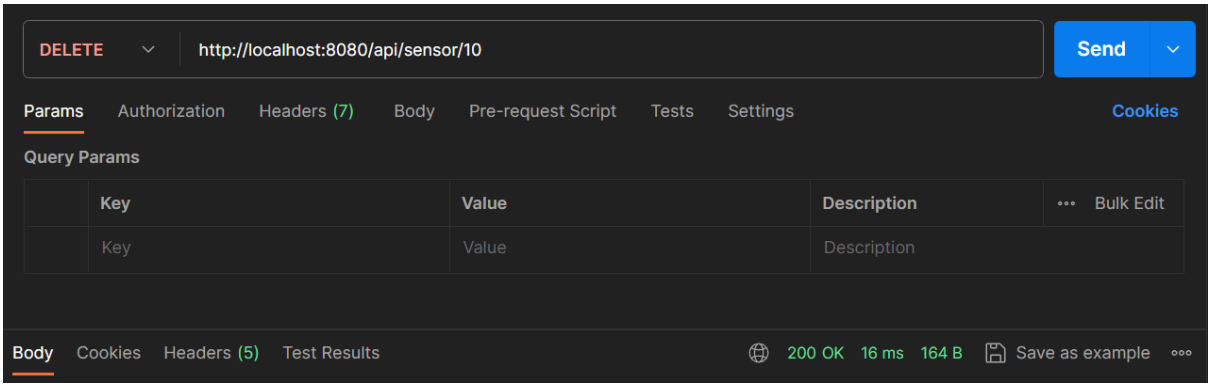
### 4. Update Sensor

- Method: PUT
- URL: /api/sensor/{id}
- Path Variable: id (ID of the sensor)
- Request Body: Same as Create Sensor
- Response:
    - Status: 200 OK
    - Body: None (Confirmation of sensor update)



### 5. Delete Sensor

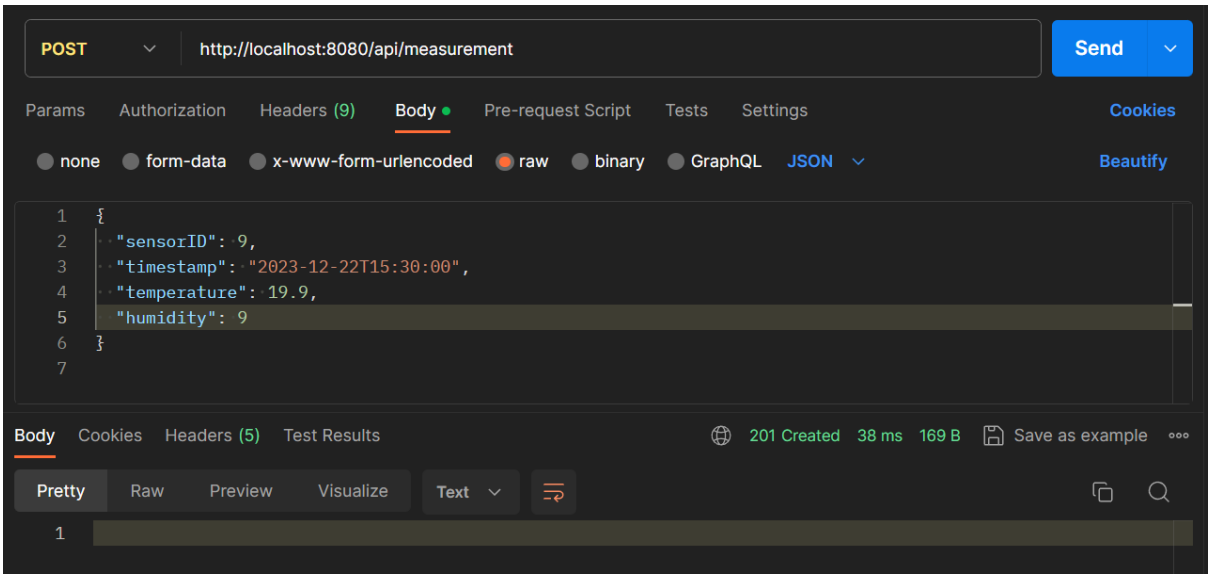
- Method: DELETE
- URL: /api/sensor/{id}
- Path Variable: id (ID of the sensor)
- Response:
    - Status: 200 OK
    - Body: None (Confirmation of sensor deletion)



# Measurement Service API Documentation

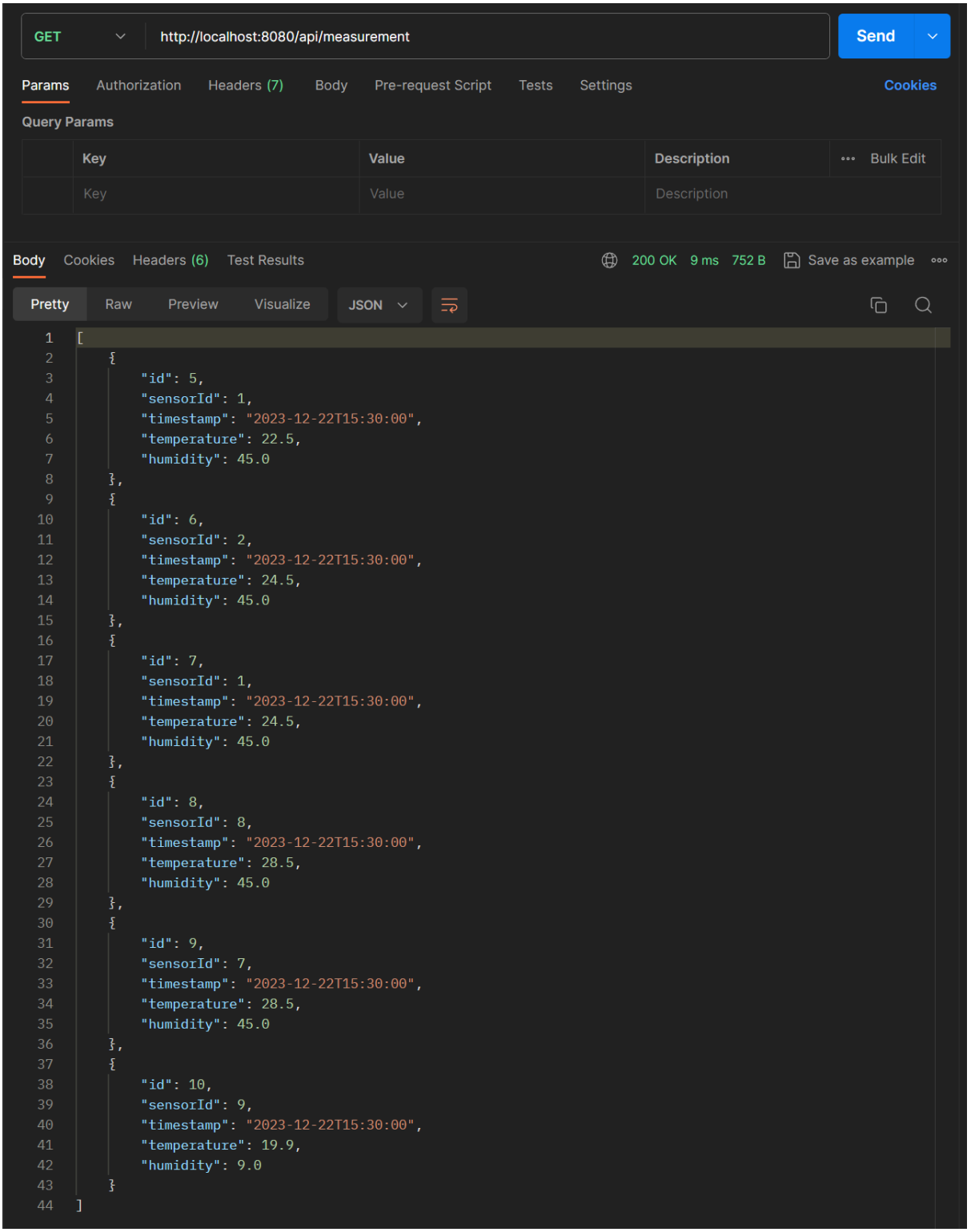
## 1. Create Measurement

- Method: POST
- URL: /api/measurement
- Request Body:
  - sensorID: Long (ID of the sensor)
  - timestamp: LocalDateTime (Time of measurement)
  - temperature: Double (Temperature value)
  - humidity: Double (Humidity value)
- Response:
  - Status: 201 Created
  - Body: None (Acknowledgement of measurement creation)



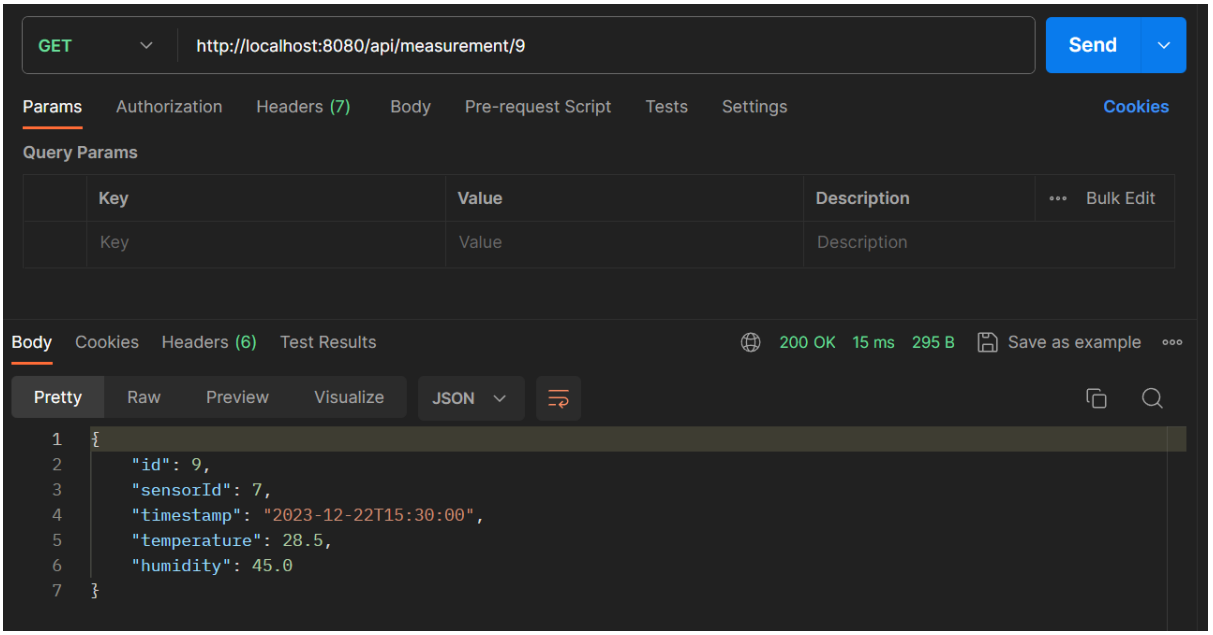
## 2. Get All Measurements

- Method: GET
- URL: /api/measurement
- Response:
    - Status: 200 OK
    - Body: List of MeasurementResponse containing details of all measurements



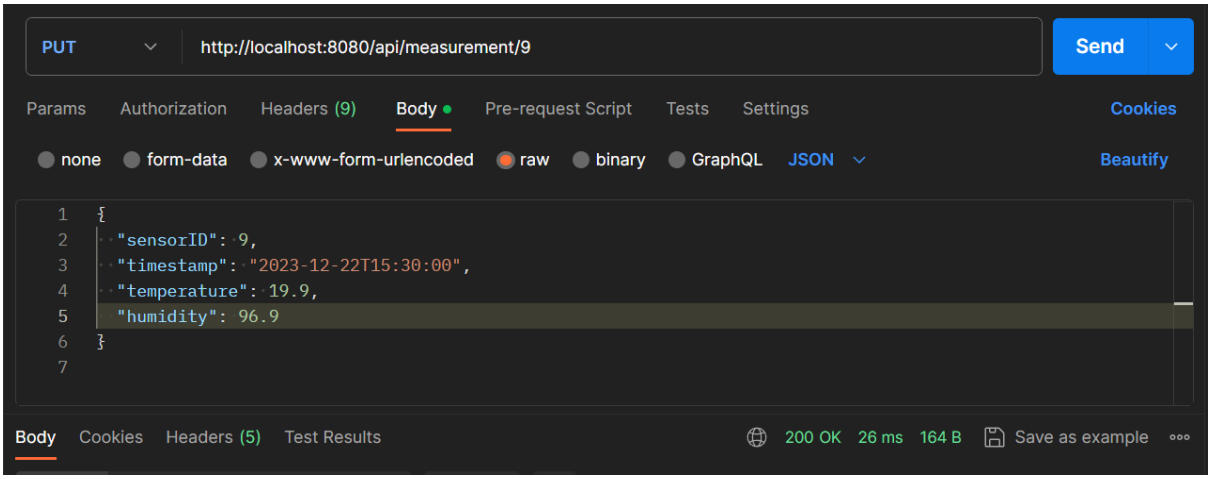
### 3. Get Measurement by ID

- Method: GET
- URL: /api/measurement/{id}
- Path Variable: id (ID of the measurement)
- Response:
    - Status: 200 OK
    - Body: MeasurementResponse containing details of the specified measurement



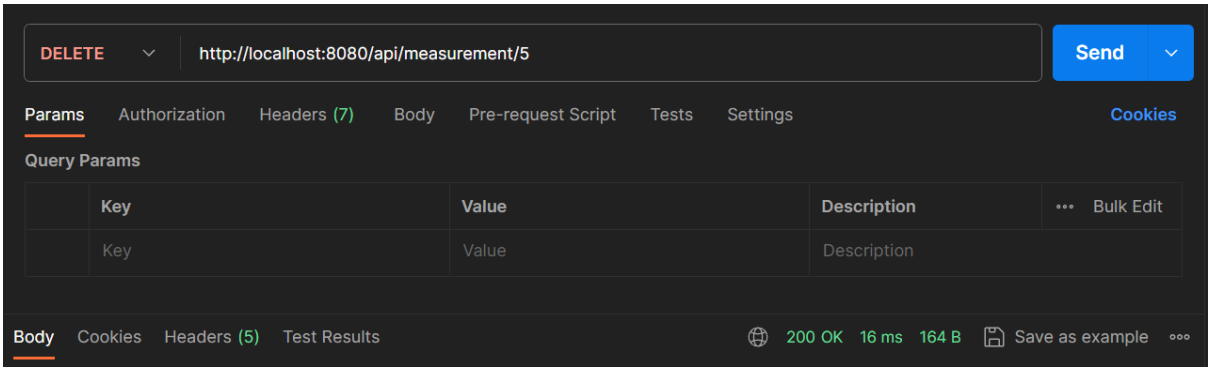
### 4. Update Measurement

- Method: PUT
- URL: /api/measurement/{id}
- Path Variable: id (ID of the measurement)
- Request Body: Same as Create Measurement
- Response:
    - Status: 200 OK
    - Body: None (Confirmation of measurement update)



### 5. Delete Measurement

- Method: DELETE
- URL: /api/measurement/{id}
- Path Variable: id (ID of the measurement)
- Response:
    - Status: 200 OK
    - Body: None (Confirmation of measurement deletion)



## Resources Folder Documentation

In our project repository, the 'test-data-and-resources' folder serves as a comprehensive hub for various essential materials and tools related to our web service implementation. The key contents of this folder include:

1. Postman Collection: A Postman we prepaid collection is available for convenient testing and interaction with the web services. This includes configured requests for all CRUD operations related to both Sensor-Service and Measurement-Service, facilitating easy demonstration and further development of the services.

2. Entity-Relationship Diagram: The folder contains a detailed entity-relationship diagram of our database, providing a visual representation of the data structure and relationships.

3. Architecture Diagram: A carefully crafted architecture diagram is included, showing the complete system setup. This diagram is annotated with labels for each component, their IP addresses, and ports, offering a clear understanding of our network architecture.

4. Screenshots: To visually demonstrate our working setup, the folder includes screenshots that show various aspects and functionalities of the application in action.

These resources in the 'resources' folder are integral to understanding the full scope and depth of our project. They not only provide insight into the technical aspects and decision-making processes but also offer practical tools for testing and evaluating our web services.

Screenshot of working env (can be found in the zip file as well):