# **Final Project: Alerts Notification System**

**Course**: PROG2200

**Student**: Cullen Murphy-Brady (W0445014)

**Submission Date**: December 2024

[Github Redirect...](https://github.com/Fall2024-NSCC-ECampus/final-project-alerts-notification-system-itscullenbrady)

## **Project Overview: Alerts Notification System**

For my final project in the PROG2200 course, I developed the *SafetyNet Alerts* system, a Spring Boot application designed to deliver emergency notifications such as fire, storm, and flood alerts to both first responders and the public. The system incorporates vital information about individuals in affected areas, including their phone numbers, addresses, medical conditions, and allergies, to enhance emergency preparedness and response efforts.

The project was implemented using the **Model-View-Controller (MVC)** design pattern and adhered to **SOLID principles** to ensure scalability and maintainability. The application provides several RESTful endpoints for accessing emergency-related data.

### **Key Features and Functionalities**

1. **Fire Station and Resident Data Retrieval**
   1. Fetches information about fire stations and the individuals they serve, based on station number or address.
2. **Child Identification in Emergency Zones**
   1. Identifies children residing in specific households or areas, critical for prioritizing assistance during emergencies.
3. **Emergency Contact System**
   1. Provides phone numbers of residents based on fire station jurisdictions for streamlined emergency communication.
4. **Resident Medical Data**
   1. Retrieves detailed information about individuals in affected areas, such as medical conditions, allergies, and medications, to assist first responders in providing appropriate care.

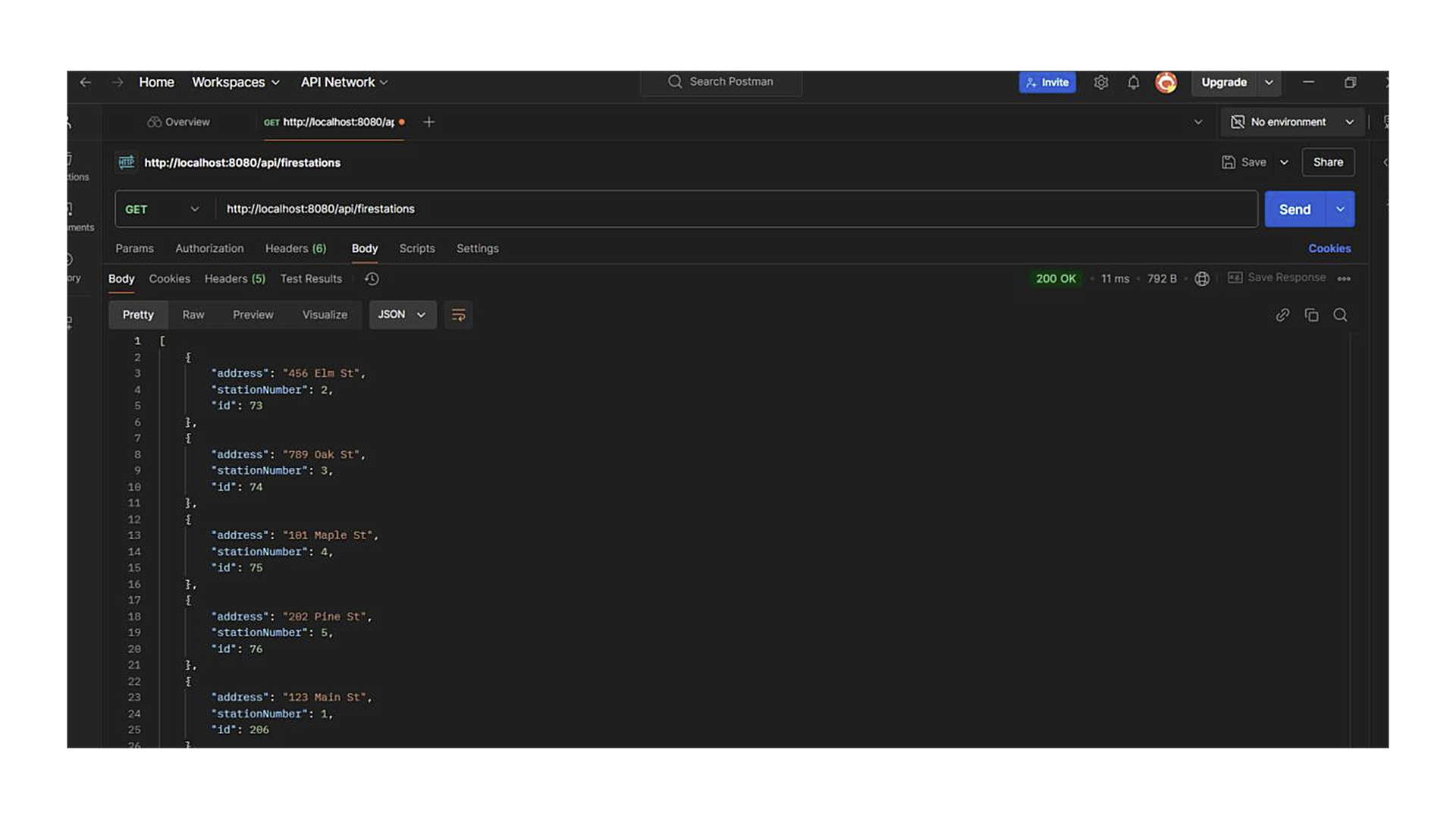
## **Test Results Summary for SafetyNet Alerts API**

The system's functionality was rigorously tested using **Postman** to ensure all API endpoints met the project requirements. Below is a summary of the endpoints and their respective outputs:

### **1. Retrieve All Fire Stations**

**URL**: <http://localhost:8080/api/firestations>

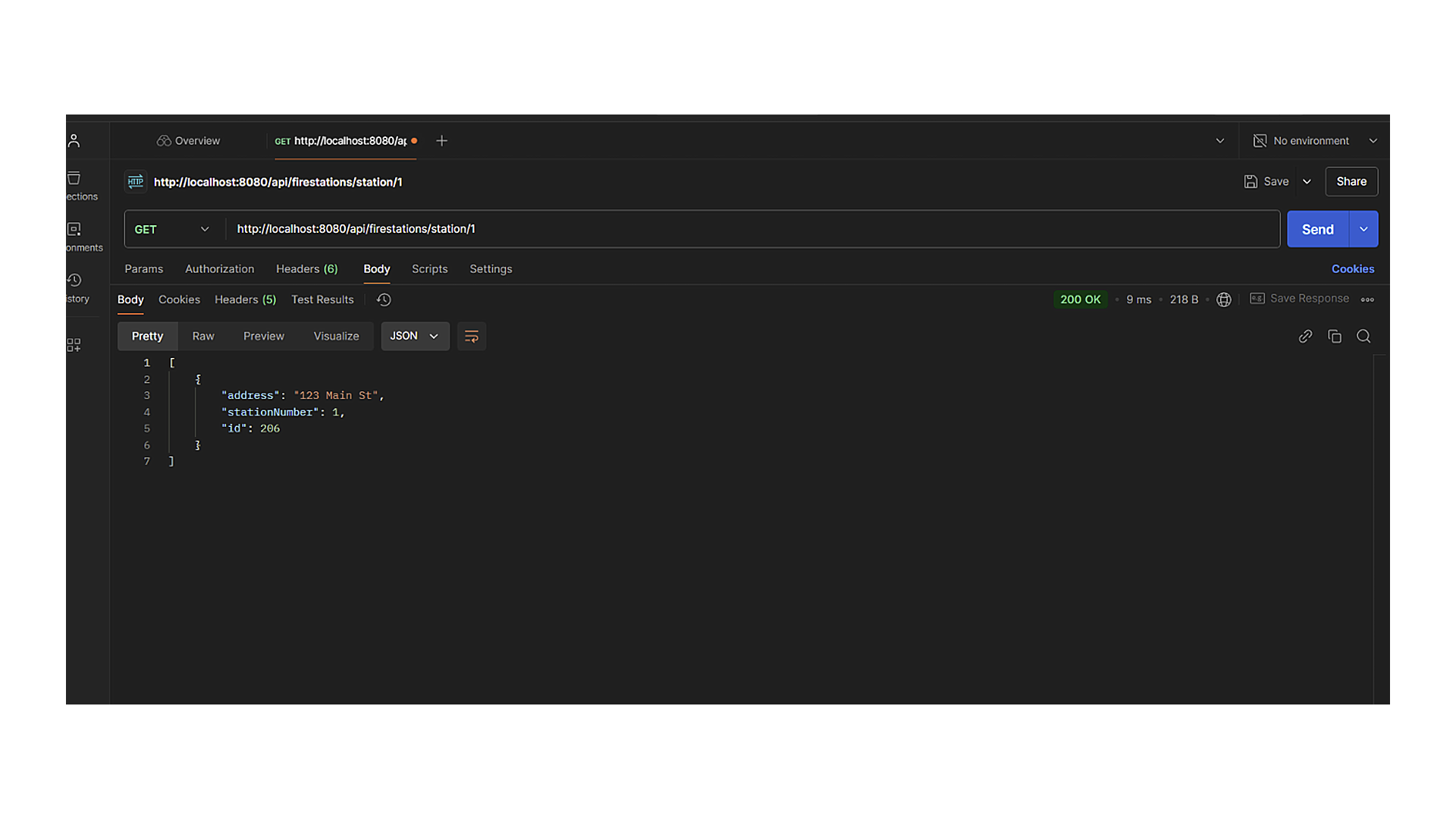
* Returns a complete list of fire stations, including station numbers and addresses.



### **2. Retrieve Fire Station by Station Number**

**URL**: <http://localhost:8080/api/firestations/station/1>

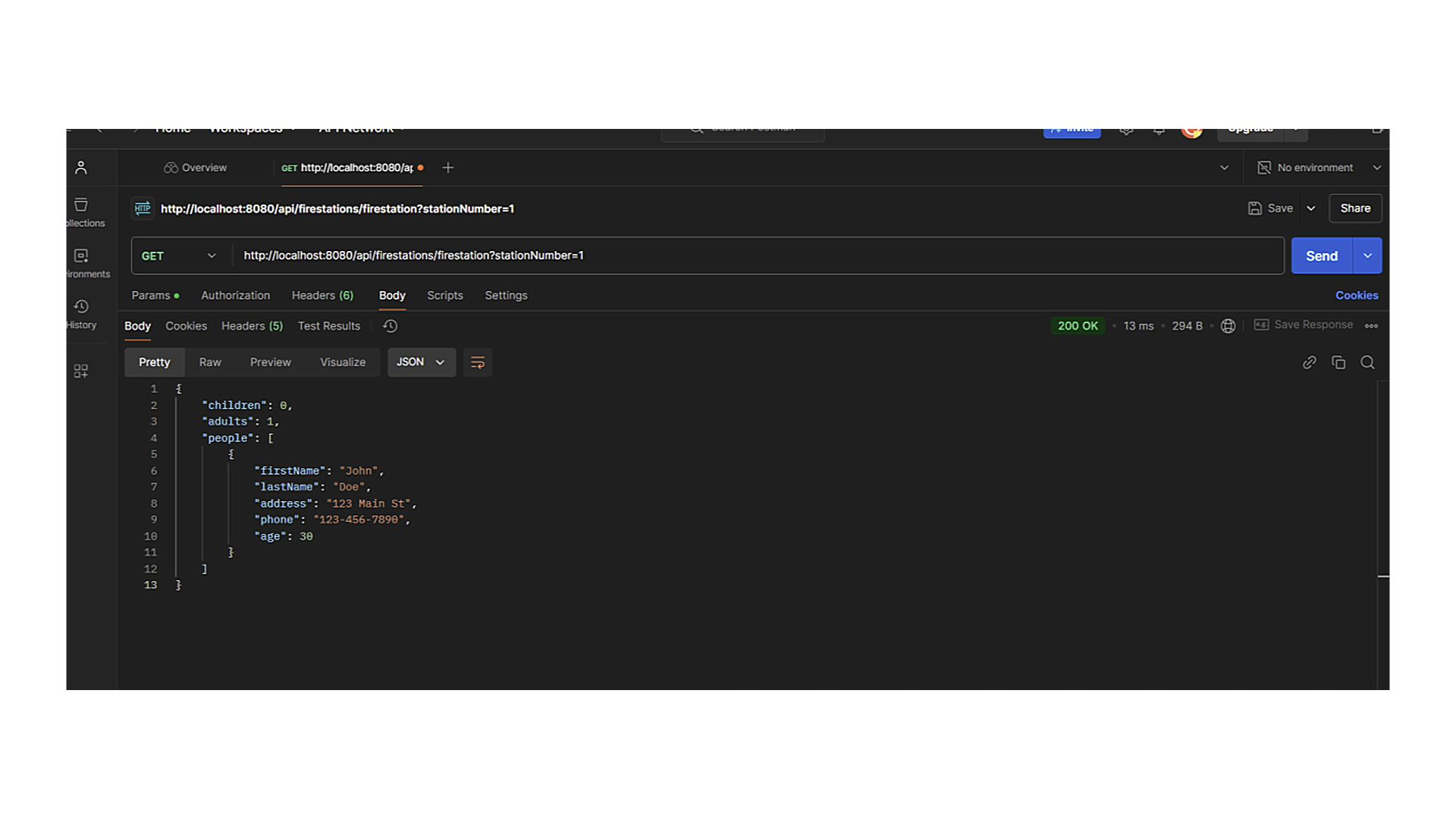
* Provides details about fire station 1, including associated addresses.



### **3. Retrieve Residents Served by Fire Station**

**URL**: <http://localhost:8080/api/firestations/firestation?stationNumber=1>

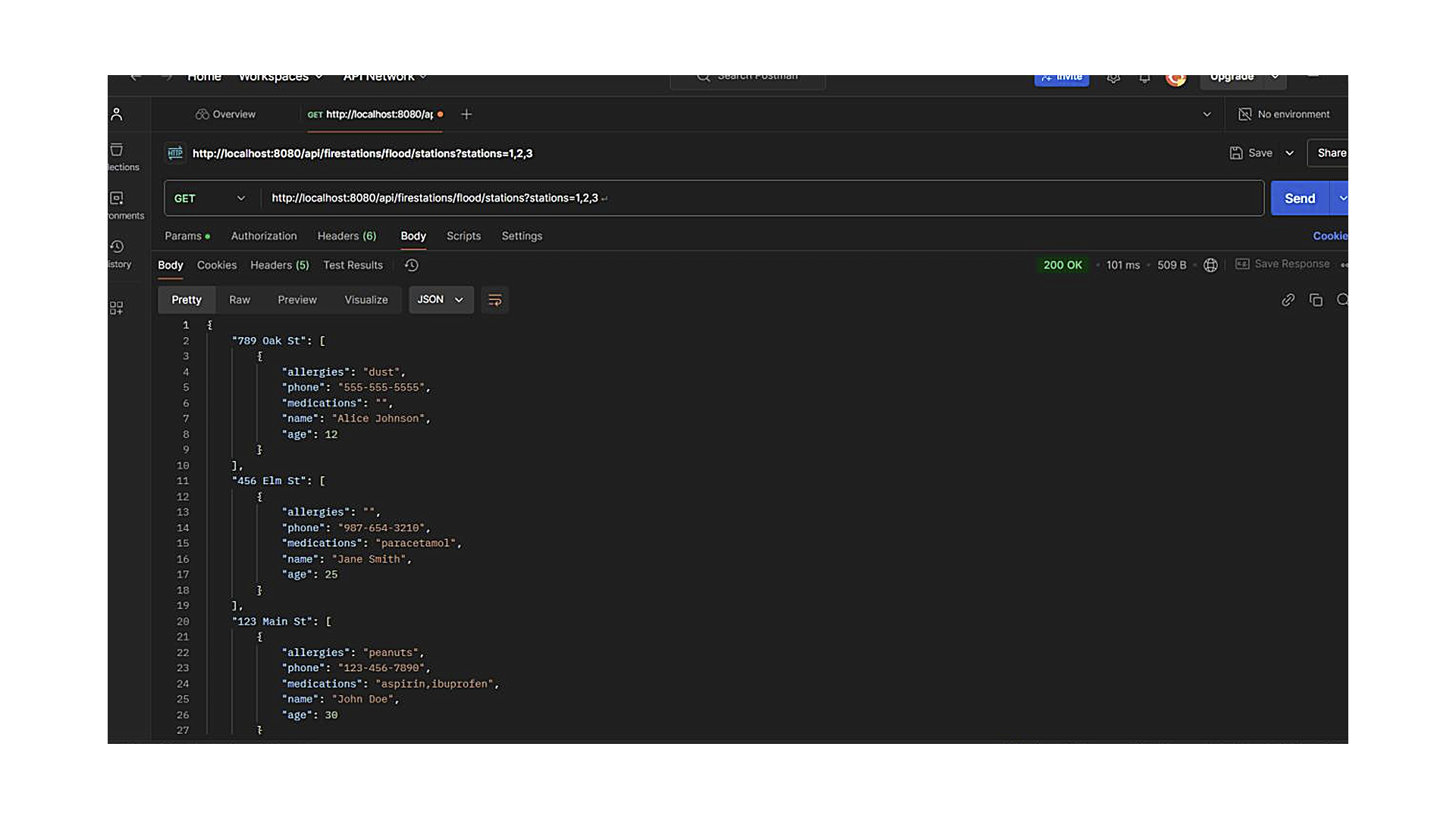
* Returns a list of individuals served by fire station 1, including:
  + Names
  + Addresses
  + Phone numbers
  + Summaries of adults and children served



### **4. Retrieve Flood Information by Fire Stations**

**URL**: <http://localhost:8080/api/flood/stations?stations=1,2,3>

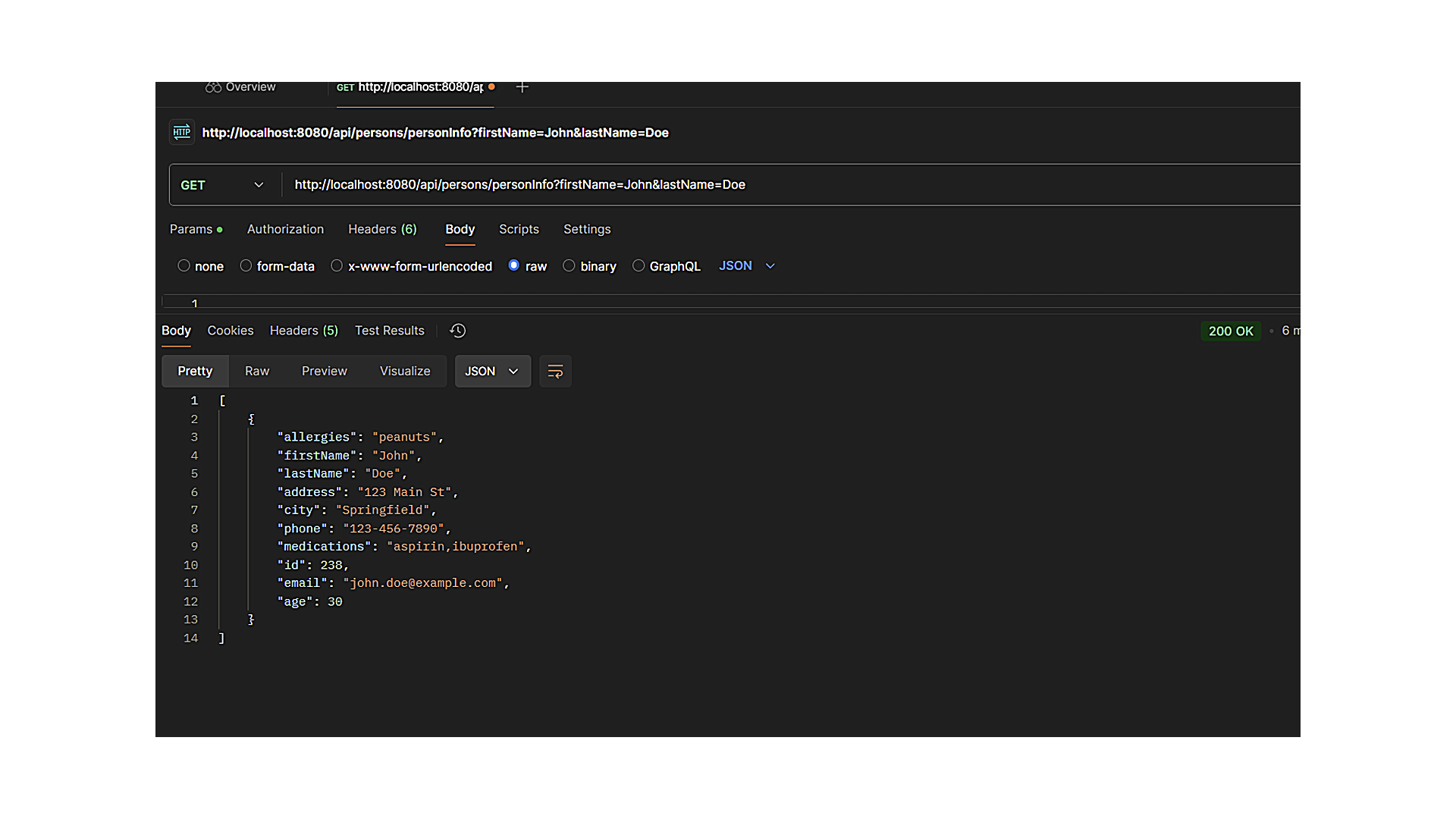
* Provides grouped household data for fire stations 1, 2, and 3, including:
  + Address
  + Names
  + Phone numbers
  + Ages
  + Medical details



### **5. Retrieve Individual Information by Name**

**URL**: <http://localhost:8080/api/persons/personInfo?firstName=John&lastName=Doe>

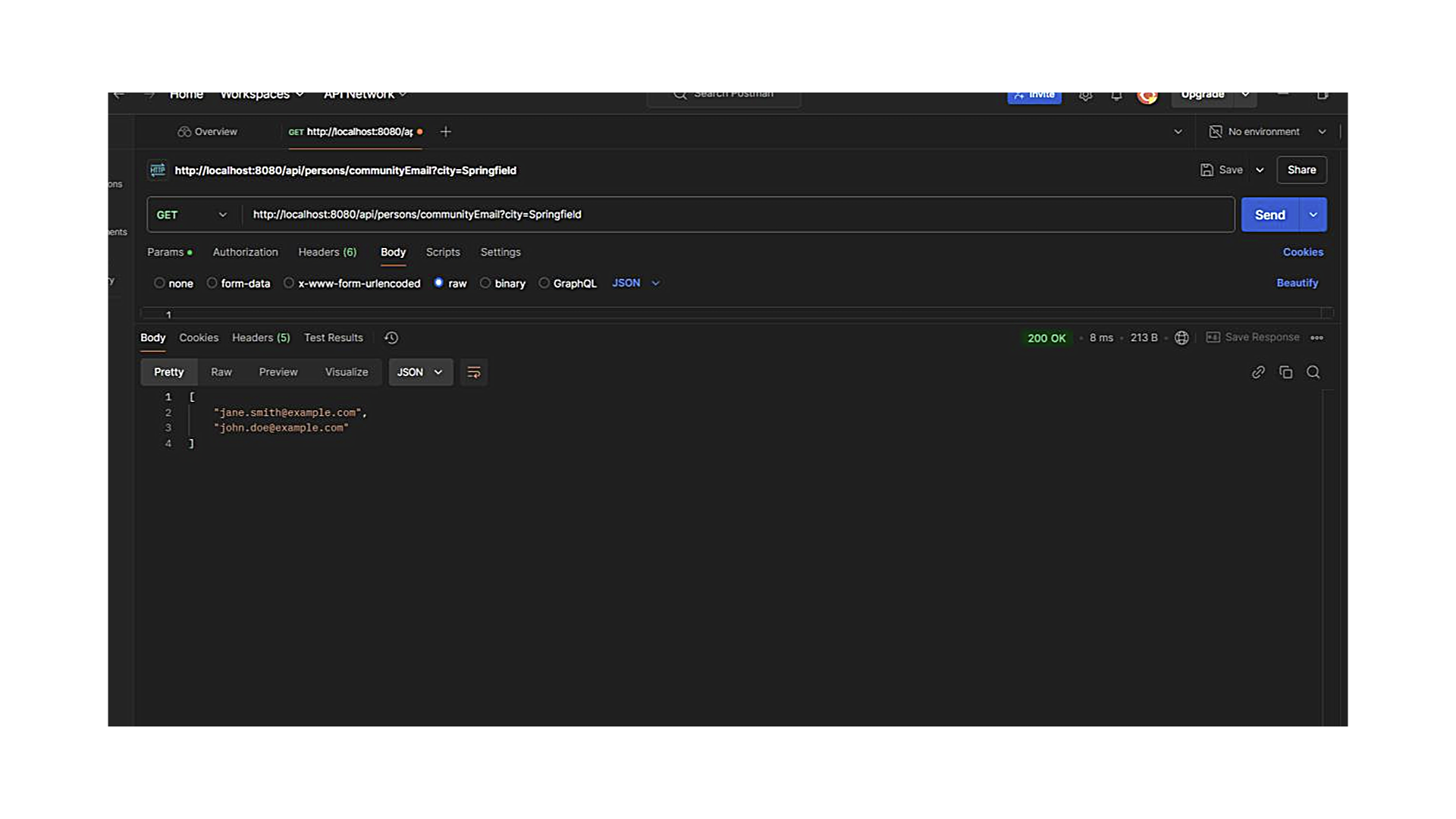
* Returns personal details for individuals named John Doe, including:
  + Address
  + Age
  + Email address
  + Medical data



### **6. Retrieve Community Emails by City**

**URL**: <http://localhost:8080/api/persons/communityEmail?city=Springfield>

* Provides email addresses for all residents of Springfield, facilitating city-wide notifications.



### **7. Retrieve Fire Station Details by Address**

**URL**: <http://localhost:8080/api/firestations/byAddress?address=456%20Elm%20St>

* Returns the fire station serving the address 456 Elm St, including resident information such as:
  + Names
  + Medical details

## 

## **Conclusion**

The *SafetyNet Alerts* system successfully addresses the challenge of delivering timely and detailed emergency notifications. By integrating key functionalities such as resident data retrieval, medical information access, and community-wide communication tools, the application ensures improved preparedness for first responders and affected individuals. The use of industry-standard design patterns and principles further supports the scalability and maintainability of the project.

|  |  |  |
| --- | --- | --- |
| **SOLID Principle** | **Description** | **Implementation in the Project** |
| **S - Single Responsibility Principle** | A class should have only one reason to change, meaning it should be responsible for a single specific task. | In the PersonService class, responsibilities were clearly defined by separating tasks into distinct methods, such as getPersonById for fetching a specific person's details and savePerson for saving data to the database. This ensures each method has a single purpose, maintaining a clear structure. |
| **O - Open/Closed Principle** | Software components should be open for extension but closed to modification, enabling the addition of new features without altering existing functionality. | Interfaces and inheritance were employed to extend functionality without modifying existing code. For instance, both FireStationService and PersonService are designed to be easily extendable for future features while preserving the integrity of the existing system. |
| **L - Liskov Substitution Principle** | Objects of a superclass should be replaceable with objects of its subclass without affecting the correctness of the program. | The inheritance hierarchy is structured to support flexible extensions. For example, should different types of fire stations or residents need to be added in the future, existing classes can be extended seamlessly without disrupting the overall system functionality. |
| **I - Interface Segregation Principle** | Clients should not be forced to depend on interfaces they do not use. Interfaces should be fine-grained and focused on specific functionalities. | Instead of designing a single, generic interface, interfaces were divided into smaller, purpose-specific ones. For example, the fire station repository and person repository have distinct methods tailored to their respective responsibilities, ensuring modularity and precision in data management. |
| **D - Dependency Inversion Principle** | High-level modules should not depend on low-level modules but rather on abstractions. Both high-level and low-level components should depend on shared abstractions. | Dependency injection was implemented throughout the services using Spring’s @Autowired annotation to inject necessary repository dependencies. This approach ensures that services depend on abstractions provided by the repository interfaces rather than concrete implementations, enhancing modularity and reducing coupling. |