**Step-by-Step Documentation**

**Step 1: Download JDK**

* **Install JDK 21**
* **Configure the JAVA\_HOME environment variable and test by running:**

**java -version**

**Step 2: Set Up Spring Boot and Add Dependencies**

1. **Initialize Spring Boot:**
   * **Use** [**Spring Initializr**](https://start.spring.io/) **to generate a new Spring Boot project.**
   * **Add dependencies:**
     + **Spring Web**
     + **Spring Data MongoDB**
     + **Lombok (optional, for reducing boilerplate code).**
2. **Add Dependencies:**

**<dependencies>**

**<dependency>**

**<groupId>org.projectlombok</groupId>**

**<artifactId>lombok</artifactId>**

**<optional>true</optional>**

**</dependency>**

**</dependencies>**

**Step 3: Import Project into Eclipse**

1. **Open Eclipse IDE.**
2. **Navigate to File > Import > Maven > Existing Maven Projects.**
3. **Select the project folder and click Finish.**
4. **Verify that dependencies load correctly.**

**Step 4: Create Required Files**

1. **Model Class: Customer  
   Define fields like id, employeeName, role, department, etc., and annotate with @Document(collection = "customers") for MongoDB. Example:**

**@Data**

**@Document(collection = "customers")**

**public class Customer {**

**@Id**

**private String id;**

**private String employeeName;**

**private String role;**

**private String department;**

**private double salary;**

**private String startDate;**

**private String endDate;**

**private String manager;**

**}**

1. **Repository Interface: CustomerRepo  
   Create a repository interface to interact with MongoDB:**

**public interface CustomerRepo extends MongoRepository<Customer, String> {**

**List<Customer> findByEmployeeNameContainingIgnoreCase(String employeeName);**

**}**

1. **Service Layer: CustomerService  
   Implement business logic for operations:**
   * **Create a customer**
   * **Get by ID**
   * **Update by ID**
   * **Delete by ID**
   * **List all customers**
   * **Search by name**
2. **Controller Layer: CustomerController  
   Define REST API endpoints and map them to service methods.**
3. **Exception Handling: GlobalExceptionHandler  
   Handle exceptions globally for consistent responses.**

**Step 5: Implement CRUD and Search Operations**

1. **Create Customer:**

**Endpoint: POST /customers**

**Description: Adds a new customer record.**

**Request Body:**

**{**

**"employeeName": "sathyaseelan001",**

**"role": "Software Engineer",**

**"startDate": "2024-01-01",**

**"endDate": "2024-01-02",**

**"salary": 80000.0,**

**"term": "Permanent",**

**"otherDetail": "N/A",**

**"department": "IT",**

**"manager": "Jane Smith"**

**}**

1. **List All Customers:**

**Endpoint: GET /customers**

**Description: Retrieves all customer records.**

1. **Get Customer by ID:  
   Endpoint: GET /customers/{id}  
   Description: Fetches a customer by their unique ID**

**Request Body:**

**{**

**"Id": "6751f0def195f214170dbf4f",**

**"employeeName": "sathyaseelan001",**

**"role": "Software Engineer",**

**"startDate": "2024-01-01",**

**"endDate": "2024-01-02",**

**"salary": 80000.0,**

**"term": "Permanent",**

**"otherDetail": "N/A",**

**"department": "IT",**

**"manager": "Jane Smith"**

**}**

1. **Update Customer by ID:**

**Endpoint: PUT /customers/{id}**

**Description: Updates a specific customer's details.**

**Request Body:**

**{**

**"Id": "6751f0def195f214170dbf4f",**

**"employeeName": "sathyaseelan001",**

**"role": "Software Engineer",**

**"startDate": "2024-01-01",**

**"endDate": "2024-01-02",**

**"salary": 80000.0,**

**"term": "Permanent",**

**"otherDetail": "N/A",**

**"department": "IT",**

**"manager": "Jane Smith"**

**}**

1. **Delete Customer by ID:  
   Endpoint: DELETE /customers/{id}  
   Description: Deletes a customer record by ID.**
2. **Search Customers by Name:  
   Endpoint: GET /customers/search?name={name}  
   Description: Searches for customers whose names contain the provided string.**

**Step 6: Add Logging**

1. **Logging helps track the flow of execution and system activity, making it easier to monitor how the application behaves under different conditions. It provides a record of what happens at runtime, including method calls, input data, and execution outcomes.**

For actions like creating, updating, deleting, or fetching employment agreements, the loader would typically be implemented on the client-side.

When an API call is triggered (e.g., via createEmploymentAgreement, updateEmploymentAgreement), the UI can display a spinner or loading overlay until the server responds with success or failure.

**Step 7: Implement Error Handling**

1. **Create GlobalExceptionHandler to handle exceptions centrally.**

When validation fails (e.g., salary is negative, or a mandatory field is missing), Spring automatically returns a 400 Bad Request response with detailed messages about the invalid fields.

A custom validation method ensures that the start date is earlier than the end date.

Fields like employeeName, role, startDate, endDate, and salary are marked as mandatory using annotations like @NotNull.

If a specific operation fails (e.g., trying to fetch a non-existent employment agreement), the GlobalExceptionHandler catches the error and returns:

**404 Not Found**: When an agreement with the specified ID is not found.

**Step 8: Run the Spring Boot Application**

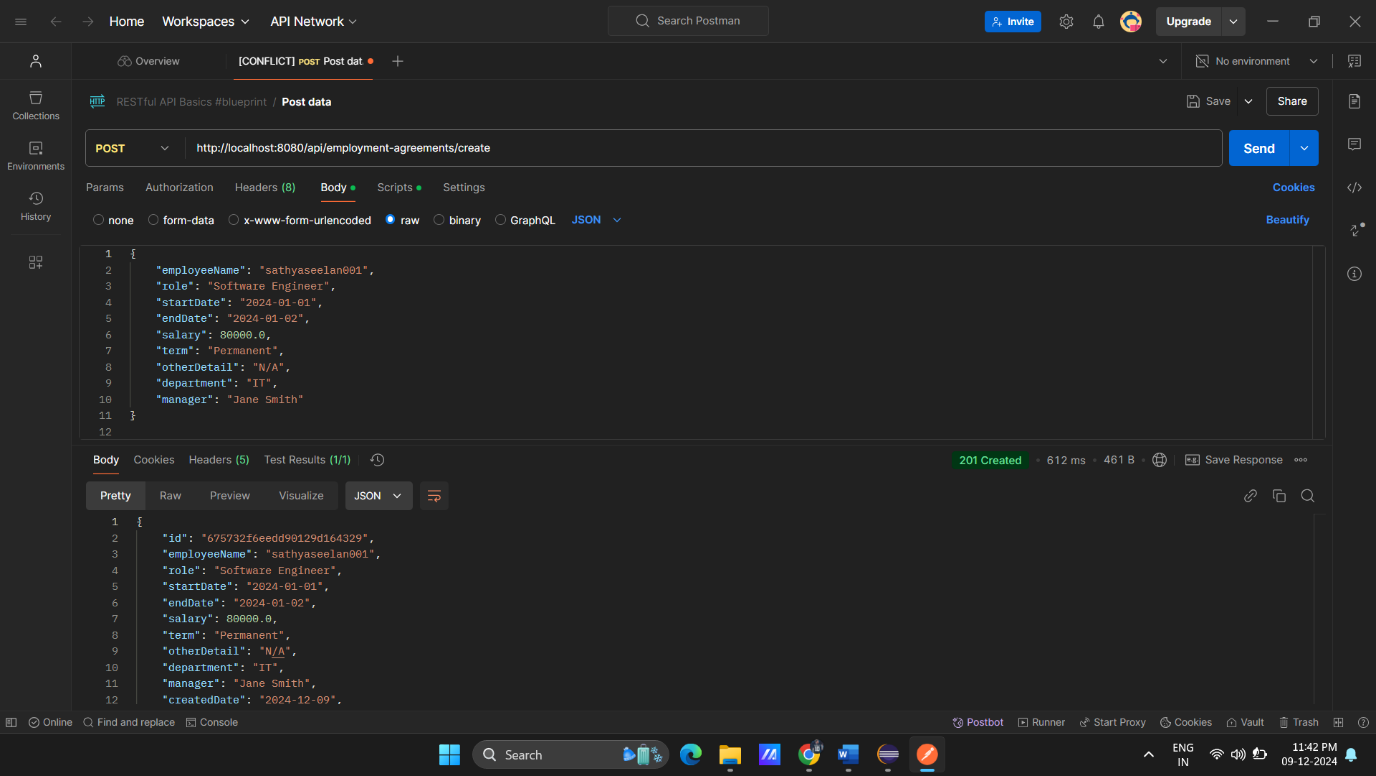
1. **Use Maven to start the application:**
2. **Ensure the service runs at http://localhost:8080.**

**Step 9: Perform Operations Using Postman**

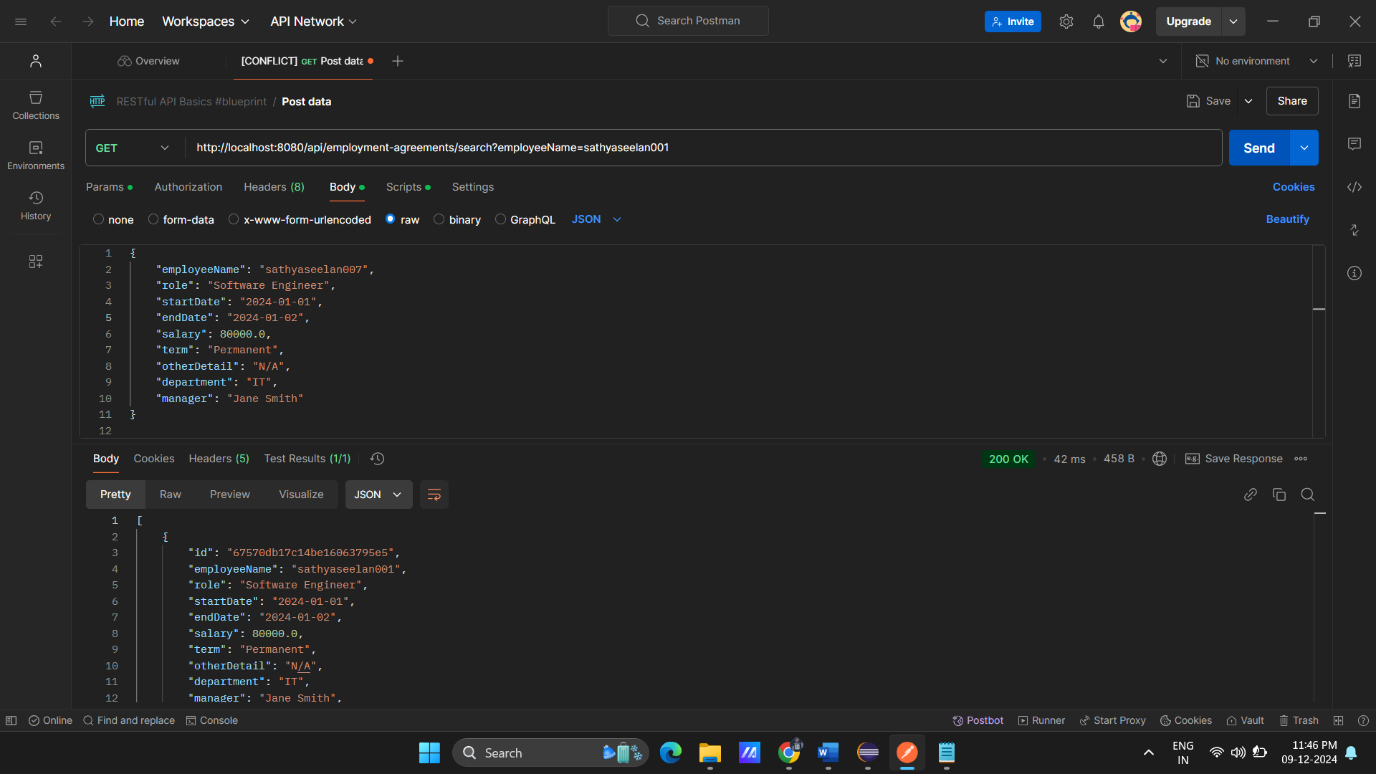
1. **Create Customer:**
   * **POST /customers with a valid JSON payload.**
2. **List All Customers:**
   * **GET /customers to fetch all customer records.**
3. **Get Customer by ID:**
   * **GET /customers/{id} to retrieve a specific record.**
4. **Update Customer by ID:**
   * **PUT /customers/{id} with updated fields in the request body.**
5. **Delete Customer by ID:**
   * **DELETE /customers/{id} to remove a record.**
6. **Search Customer by Name:**
   * **GET /customers/search?name=John to find customers matching the name.**

**Example Workflow**

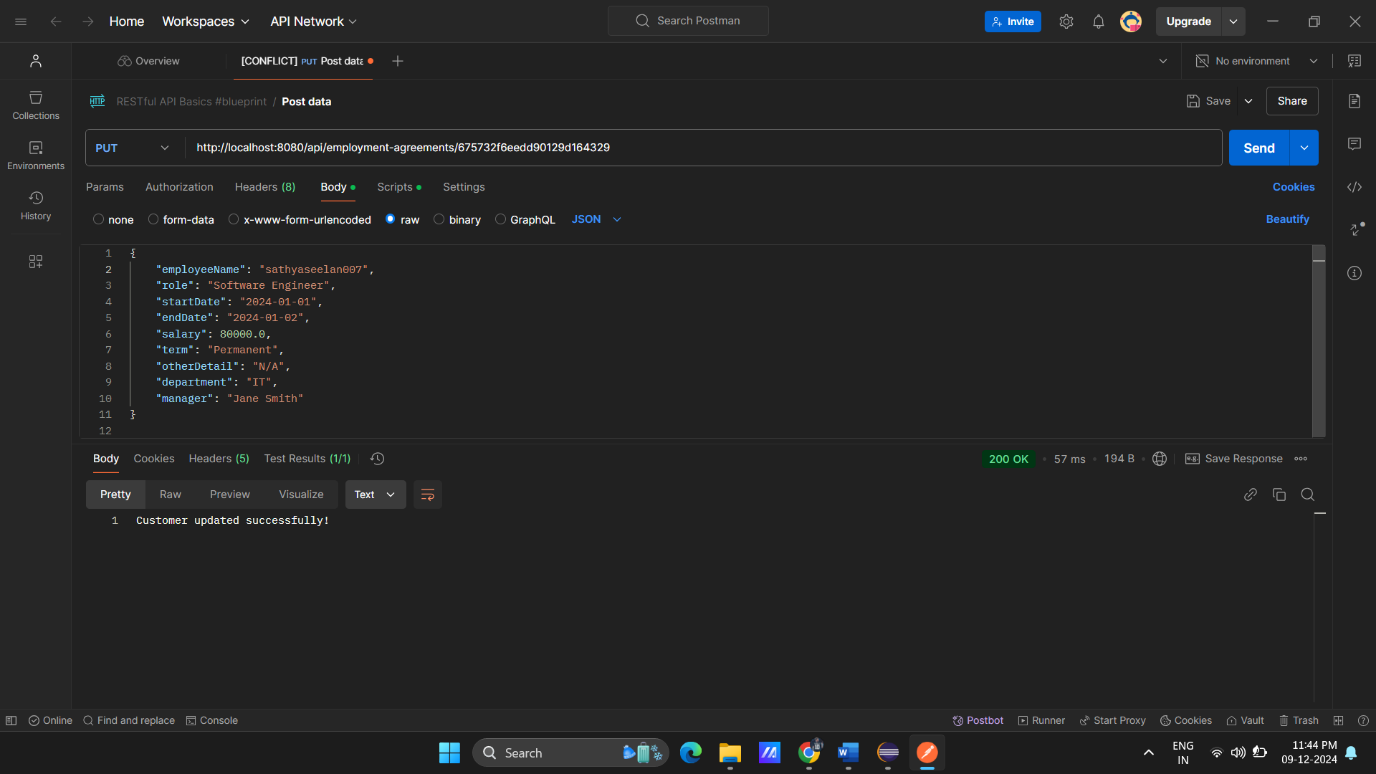
1. **Create:**
   * **POST /api/employment-agreements/create with customer data.**
   * **Returns a 201 Created status with the created customer details.**

****

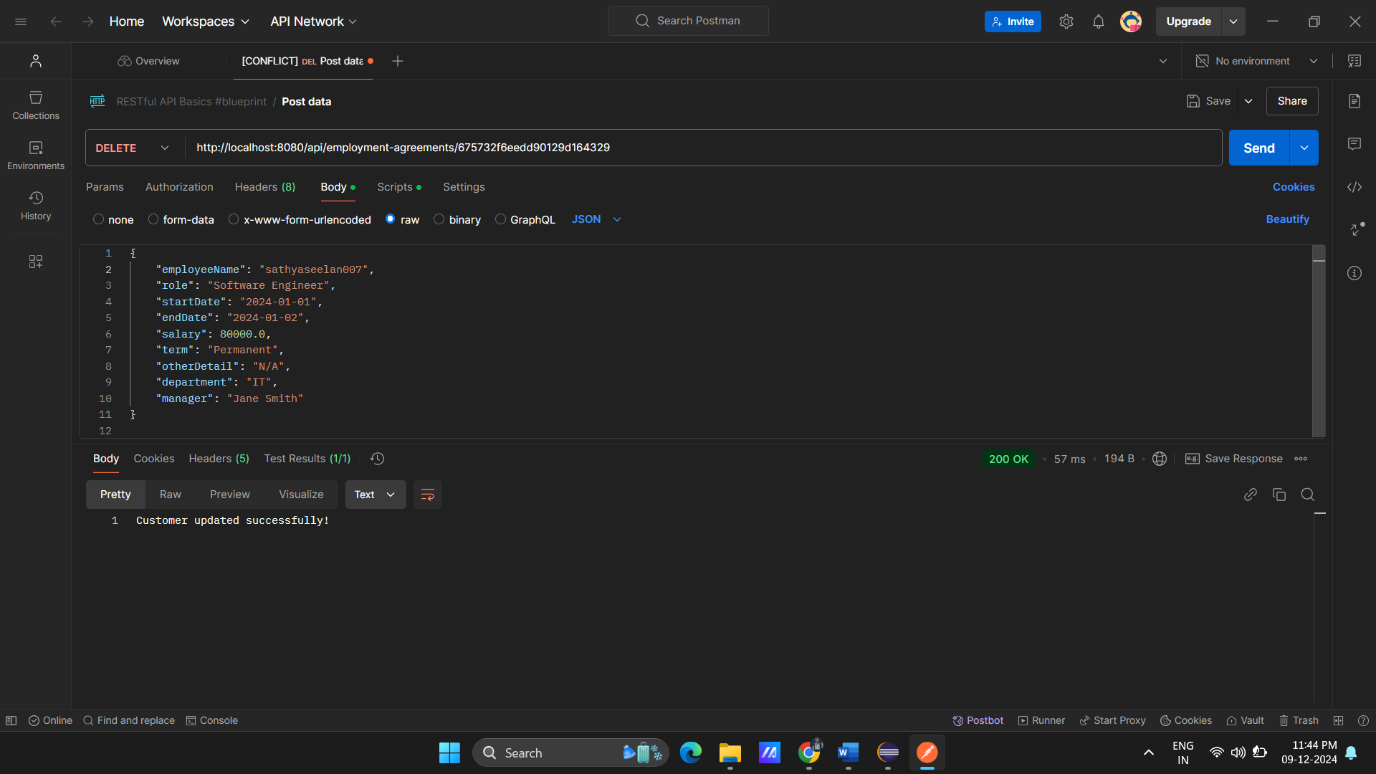
1. **Search:**
   * **GET /api/employment-agreements/search?employeeName=sathyaseelan001.**
   * **Returns customers whose names match "John".**

****

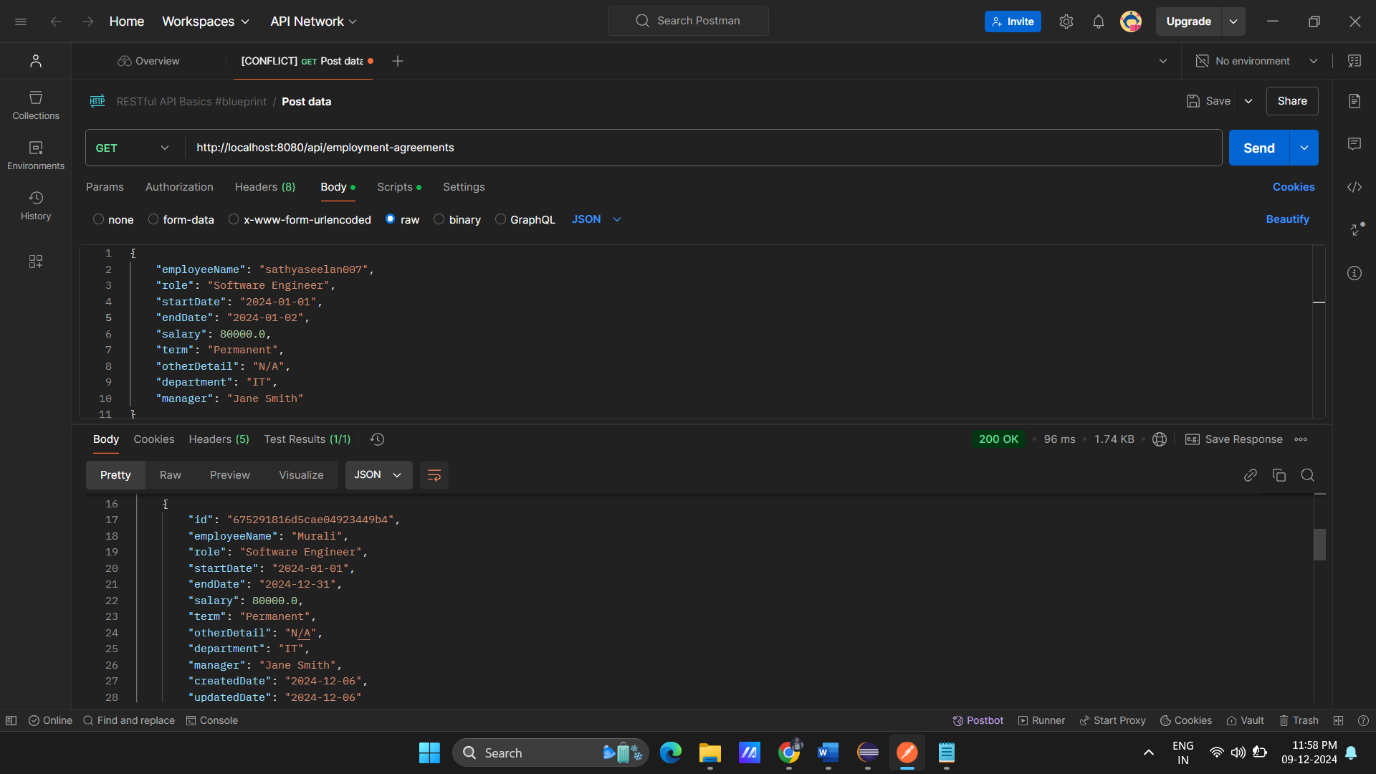
1. **Update:**
   * **PUT /api/employment-agreements/{id} to update details.**
   * **Returns a 200 OK with the updated record.**

****

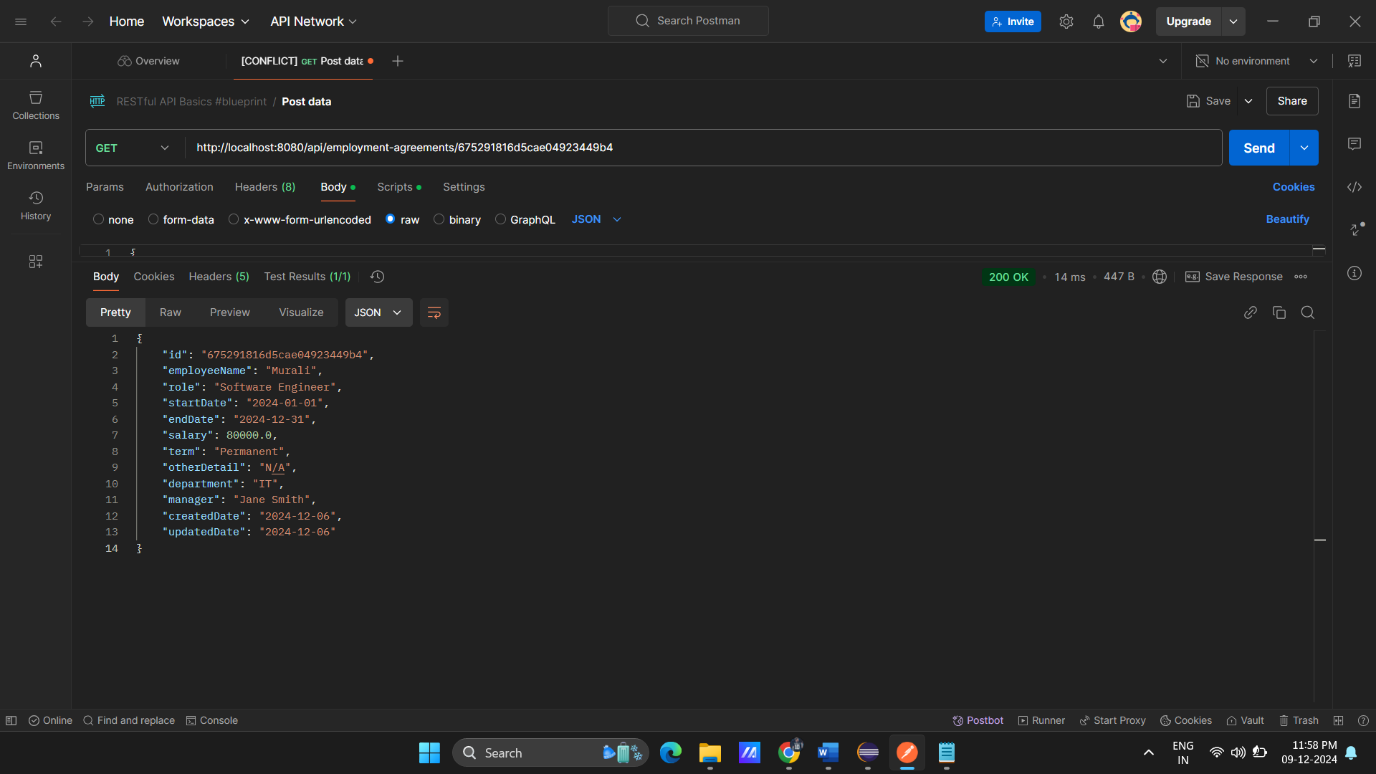
1. **Delete:**
   * **DELETE /customers/{id}.**
   * **Returns a 200 OK with a success message.**

****

1. **List:**
   * **GET /api/employment-agreements.**
   * **Returns a 200 OK with a success message.**

****

1. **Get**
   * **GET /api/employment-agreements/(id).**
   * **Returns a 200 OK with a success message.**

****