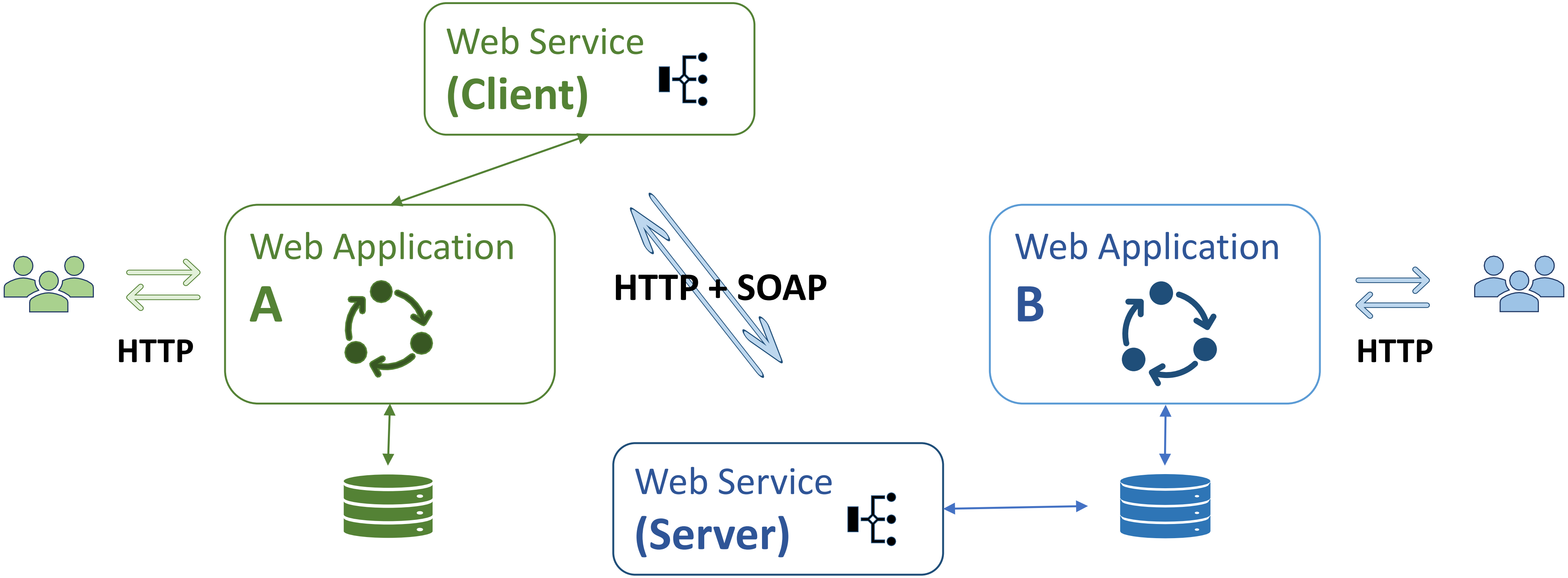
# Spring RESTful API

By

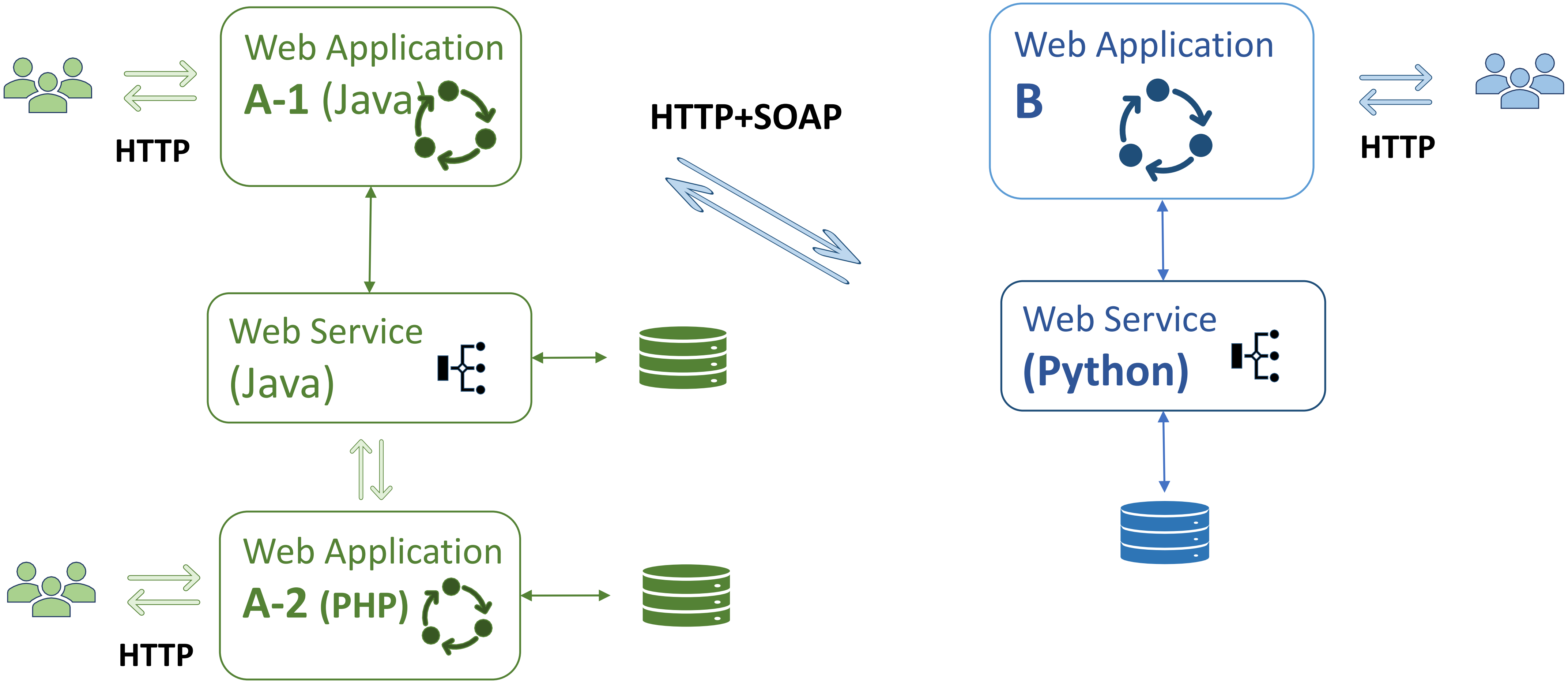
Pichet Limvajiranan

# Web Service (1)

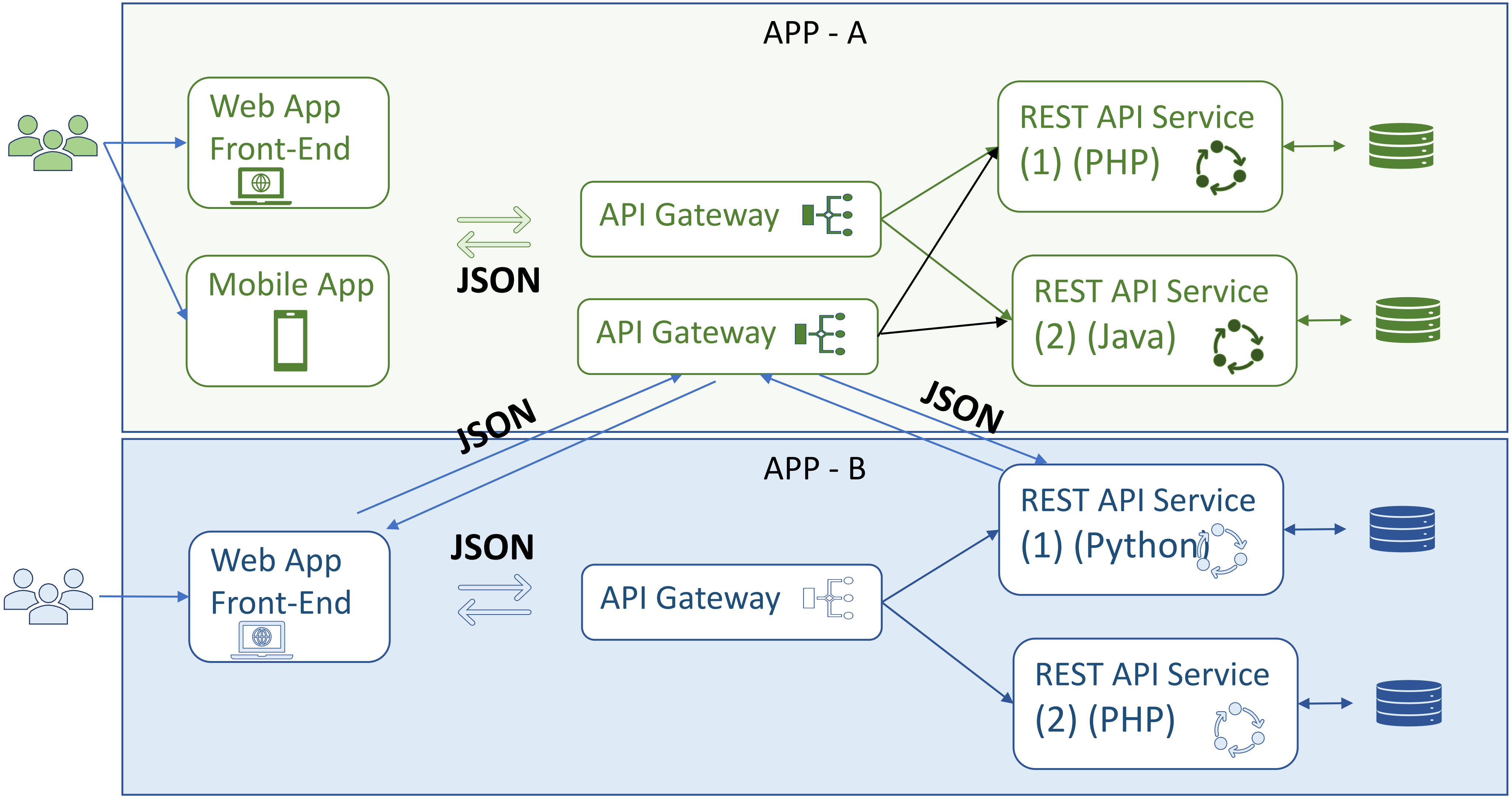
SOAP: Simple Object Access Protocol



# Web Service (2)



# Web Service (3)


APP - A
Web App Front-End
Mobile App
JSON
API Gateway
API Gateway
REST API Service (1) (PHP)
REST API Service (2) (Java)
JSON
JSON
APP - B
Web App Front-End
JSON
API Gateway
REST API Service (1) (Python)
REST API Service (2) (PHP)

# REST API (also known as RESTful API)

- REST stands for **RE**presentational **S**tate **T**ransfer and was created by computer scientist Roy Fielding.
- An application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.
- In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the **resources**.
- Each **resource** is **identified** by URIs/ global IDs.
- REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

# RESTFul Principles and Constraints

- RESTFul Client-Server
  - Server will have a RESTful web service which would provide the required functionality to the client.
  - The client send’s a request to the web service on the server. The server would either reject the request or comply and provide an adequate response to the client.
- Stateless
  - Statelessness mandates that each request from the client to the server must contain all of the information necessary to understand and complete the request.
  - The server cannot take advantage of any previously stored context information on the server.
  - For this reason, the client application must entirely keep the session state.

# RESTFul Principles and Constraints (2)

- Cacheable
  - The cacheable constraint requires that a response should implicitly or explicitly label itself as cacheable or non-cacheable.
  - If the response is cacheable, the client application gets the right to reuse the response data later for equivalent requests and a specified period.
- Layered system
  - The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior.
  - For example, in a layered system, each component cannot see beyond the immediate layer they are interacting with.

# RESTFul Principles and Constraints (3)

- Interface/Uniform Contract
  - This is the underlying technique of how RESTful web services should work. RESTful basically works on the HTTP web layer and uses the below key verbs to work with resources on the server.
    - POST – To create a resource on the server
    - GET – To retrieve a resource from the server
    - PUT – To change the state of a resource or to update it
    - DELETE – To remove or delete a resource from the server
- Code on demand (optional)
  - REST also allows client functionality to extend by downloading and executing code in the form of applets or scripts.
  - Servers can provide part of features delivered to the client in the form of code, and the client only needs to execute the code.

# Rules of REST API

- There are certain rules which should be kept in mind while creating REST **API endpoints**.
  - REST is based on the resource or **noun** instead of action or verb based. It means that a URI of a REST API should always end with a noun. Example: **/api/users** is a good example.
  - HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE.
  - A web application should be organized into resources like users and then uses HTTP verbs like – GET, PUT, POST, DELETE to modify those resources. And as a developer it should be clear that what needs to be done just by looking at the endpoint and HTTP method used.

# RESTful Resource Example

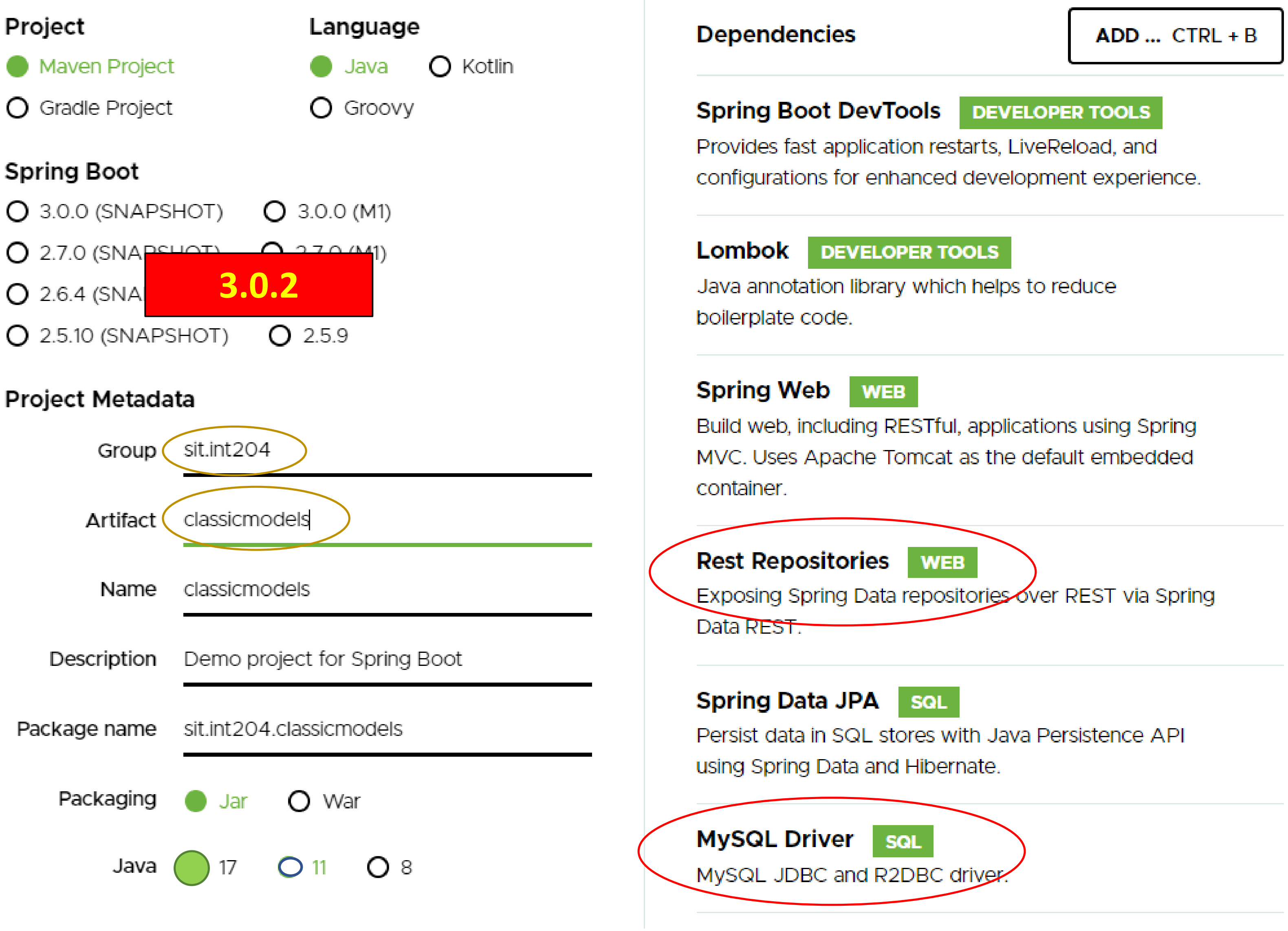
| URI | HTTP verb | Description |
|---|---|---|
| api/users | GET | Get all users |
| api/users/new | GET | Show form for adding new user |
| api/users | POST | Add a user |
| api/users/1 | PUT | Update a user with id = 1 |
| api/users/1/edit | GET | Show edit form for user with id = 1 |
| api/users/1 | DELETE | Delete a user with id = 1 |
| api/users/1 | GET | Get a user with id = 1 |

Always use plurals in URL to keep an API URI consistent throughout the application.
Send a proper HTTP code to indicate a success or error status.

# Building a Spring Boot REST API

- Step 1: Initializing a Spring Boot Project
- Step 2: Connecting Spring Boot to the Database
- Step 3: Creating a User Model
- Step 4: Creating Repository Classes (Persistence Layer)
- Step 5: Creating Service Classes (Business Layer)
- **Step 6: Creating a Rest Controller** (Presentation Layer)
- Step 7: Compile, Build and Run
- **Step 8: Testing the APIs** POSTMAN

# Step 1: Initializing a Spring Boot Project



**Project**

● Maven Project

○ Gradle Project

**Language**

● Java ○ Kotlin

○ Groovy

**Spring Boot**

○ 3.0.0 (SNAPSHOT) ○ 3.0.0 (M1)

○ 2.7.0 (SNAPSHOT) ○ 2.7.0 (M1)

○ 2.6.4 (SNA

3.0.2

○ 2.5.10 (SNAPSHOT) ○ 2.5.9

**Project Metadata**

Group sit.int204

Artifact classicmodels

Name classicmodels

Description Demo project for Spring Boot

Package name sit.int204.classicmodels

Packaging ● Jar ○ War

Java ● 17 ○ 11 ○ 8

**Dependencies** ADD ... CTRL + B

**Spring Boot DevTools** DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Lombok** DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

**Spring Web** WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Rest Repositories** WEB

Exposing Spring Data repositories over REST via Spring Data REST.

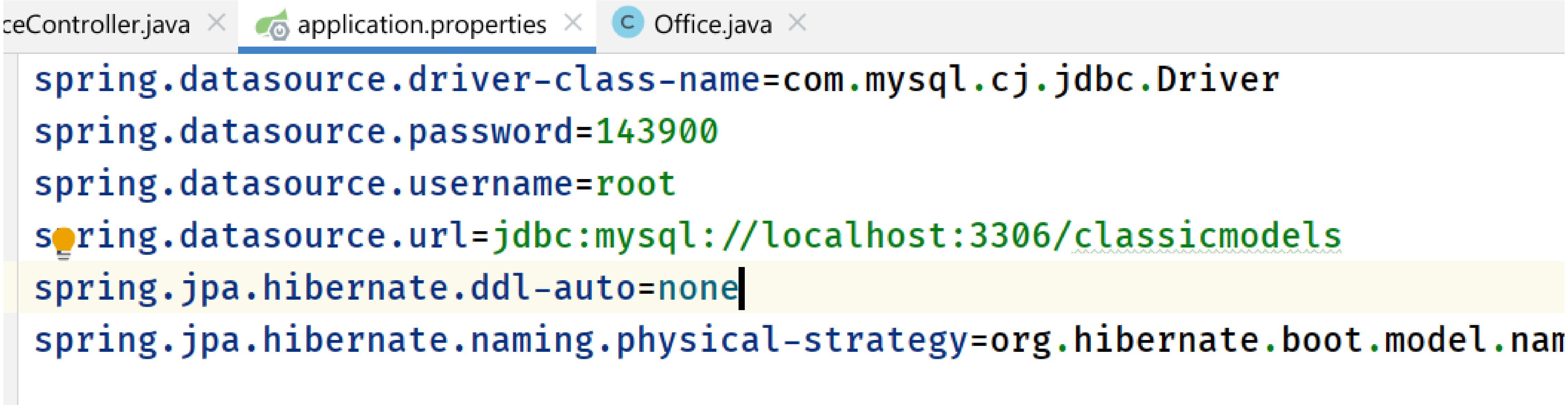
**Spring Data JPA** SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

**MySQL Driver** SQL

MySQL JDBC and R2DBC driver.

# Step 2: Connecting Spring Boot to the Database



```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.password=143900
spring.datasource.username=root
spring.datasource.url=jdbc:mysql://localhost:3306/classicmodels
spring.jpa.hibernate.ddl-auto=none
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.nam
```

# Step 3: Creating an Office Model

```
@Entity
@Table(name = "offices")
@JsonRootName("Office")
public class Office {
    @Id
    @Column(name = "officeCode", nullable = false, length = 10)
    private String id;
    @Column(name = "city", nullable = false, length = 50)
    private String city;
    @Column(name = "phone", nullable = false, length = 50)
    private String phone;
    @Column(name = "addressLine1", nullable = false, length = 50)
    private String addressLine1;
    @Column(name = "addressLine2", length = 50)
    private String addressLine2;
    @Column(name = "state", length = 50)
    private String state;
```

# Step 4: Creating Repository Classes

```
import org.springframework.data.jpa.repository.JpaRepository;
import sit.int204.demo.entities.Office;

public interface OfficeRepository extends
        JpaRepository<Office, String> {
}
```

# Step 5: Creating a Service (1)

```
@Service
public class OfficeService {
    @Autowired

    private OfficeRepository repository;
    public List<Office> getAllOffices() {
        return repository.findAll();
    }

    public Office getOffice(String officeCode) {
        return repository.findById(officeCode).orElseThrow(
                ()->new RuntimeException(officeCode+ " does not exist !!!"));
    }

    public Office addNewOffice(Office newOffice) {
          return service.create(newOffice);
     }
```

# Creating a Service (2)

```
public Office update(String officeCode , Office updateOffice) {
    Office office = repository.findById(officeCode).map(o->mapOffice(o, updateOffice))
            .orElseThrow( ()->new RuntimeException(officeCode+ " does not exist !!!"));
    return repository.saveAndFlush(office);
}

public void deleteOffice(String officeCode) {
    repository.findById(officeCode).orElseThrow(()->
            new RuntimeException(officeCode + " does not exist !!!"));
    repository.deleteById(officeCode);
}
```

# Step 6: Creating a Controller (1)

```
@RestController
@RequestMapping("/api/offices")
public class OfficeController {
   @Autowired
   private OfficeService service;

   @GetMapping("")
   public List<Office> getOffices() {
      return service.getAllOffices();
   }

   @GetMapping("/{officeCode}")
   public Office getOffice(@PathVariable String officeCode) {
      return service.getOffice(officeCode).orElseThrow(
            ()->new RuntimeException(officeCode+ " does not exist !!!"));
   }
```

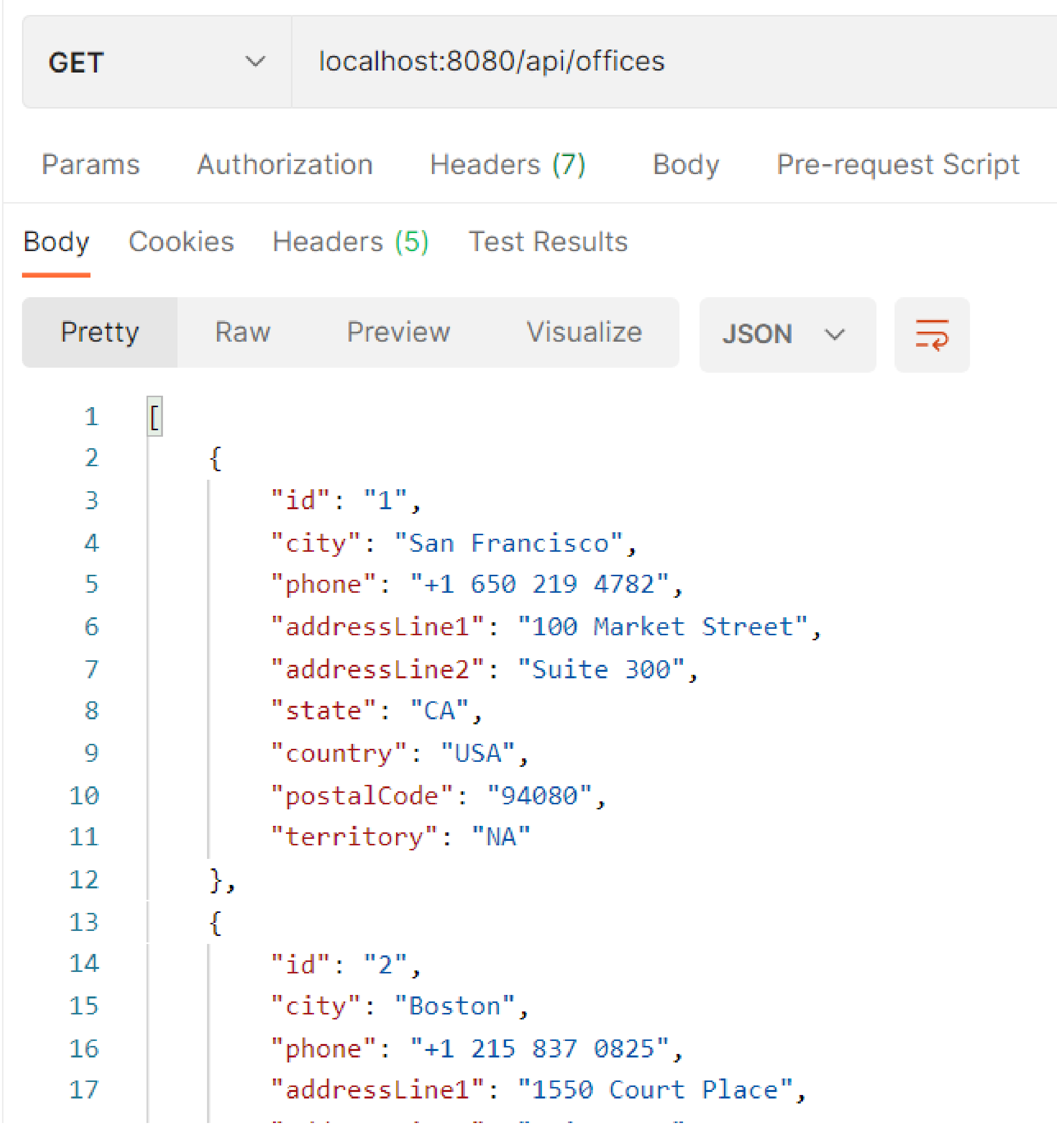
Path Variable **{ }**

{officeCode}

# Creating a Controller (2)

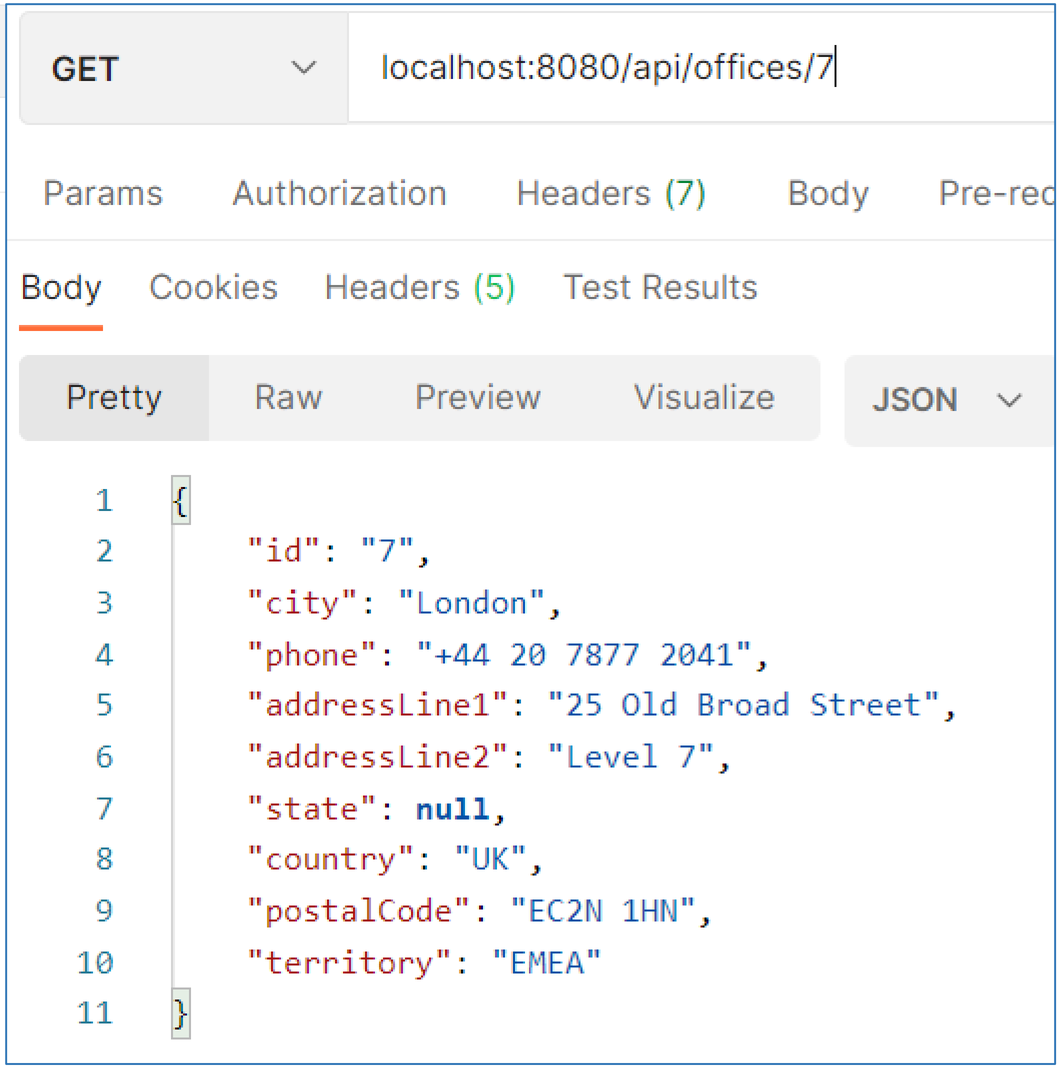
```
@PostMapping("")
@ResponseStatus(HttpStatus.CREATED)
public Office create(@RequestBody Office newOffice) {
   return service.addNewOffice(newOffice);
}

@PutMapping("/{officeCode}")
public Office update(@RequestBody Office updateOffice, @PathVariable String
officeCode) {
    return service.updateOffice(officeCode, updateOffice);
}

@DeleteMapping("/{officeCode}")
public void delete(@PathVariable String officeCode) {
   repository.deleteOffice(officeCode);
}
```

# Step 8: Testing the APIs (GET)



localhost:8080/api/offices

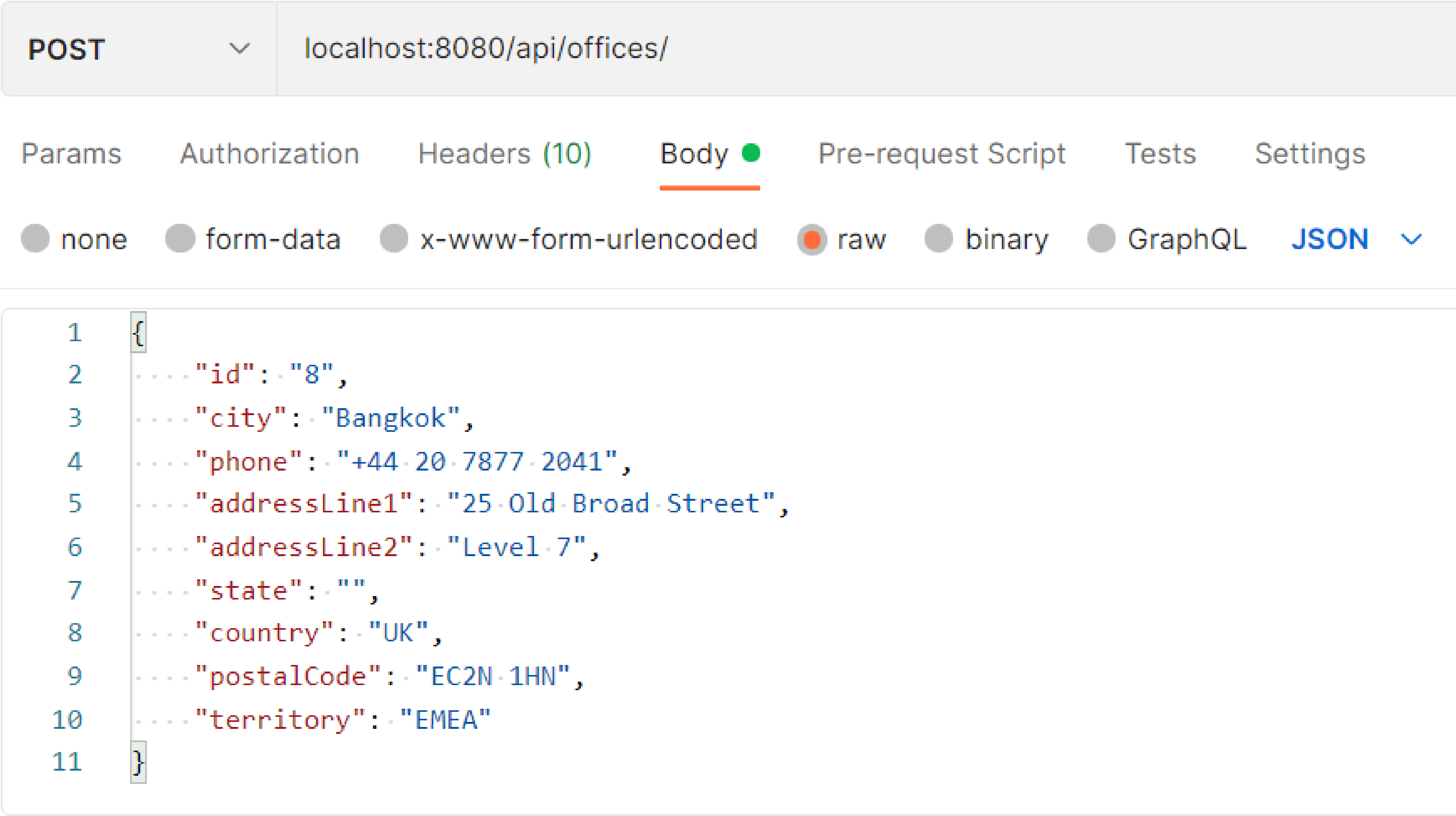
```
[
    {
        "id": "1",
        "city": "San Francisco",
        "phone": "+1 650 219 4782",
        "addressLine1": "100 Market Street",
        "addressLine2": "Suite 300",
        "state": "CA",
        "country": "USA",
        "postalCode": "94080",
        "territory": "NA"
    },
    {
        "id": "2",
        "city": "Boston",
        "phone": "+1 215 837 0825",
        "addressLine1": "1550 Court Place",
```



localhost:8080/api/offices/7

```
{
    "id": "7",
    "city": "London",
    "phone": "+44 20 7877 2041",
    "addressLine1": "25 Old Broad Street",
    "addressLine2": "Level 7",
    "state": null,
    "country": "UK",
    "postalCode": "EC2N 1HN",
    "territory": "EMEA"
}
```

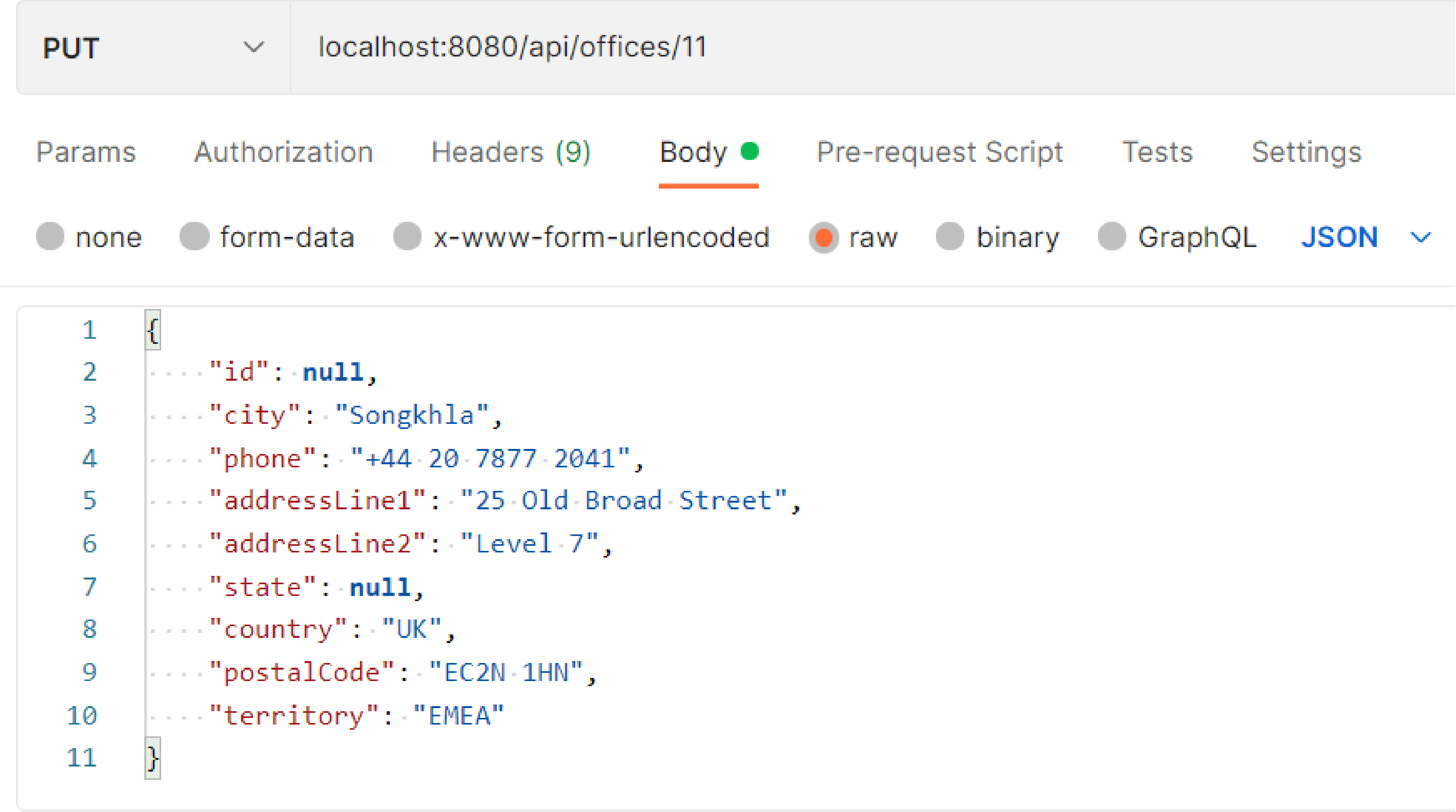
# Step 8: Testing the APIs (POST)



POST localhost:8080/api/offices/

```
{
    "id": "8",
    "city": "Bangkok",
    "phone": "+44 20 7877 2041",
    "addressLine1": "25 Old Broad Street",
    "addressLine2": "Level 7",
    "state": "",
    "country": "UK",
    "postalCode": "EC2N 1HN",
    "territory": "EMEA"
}
```

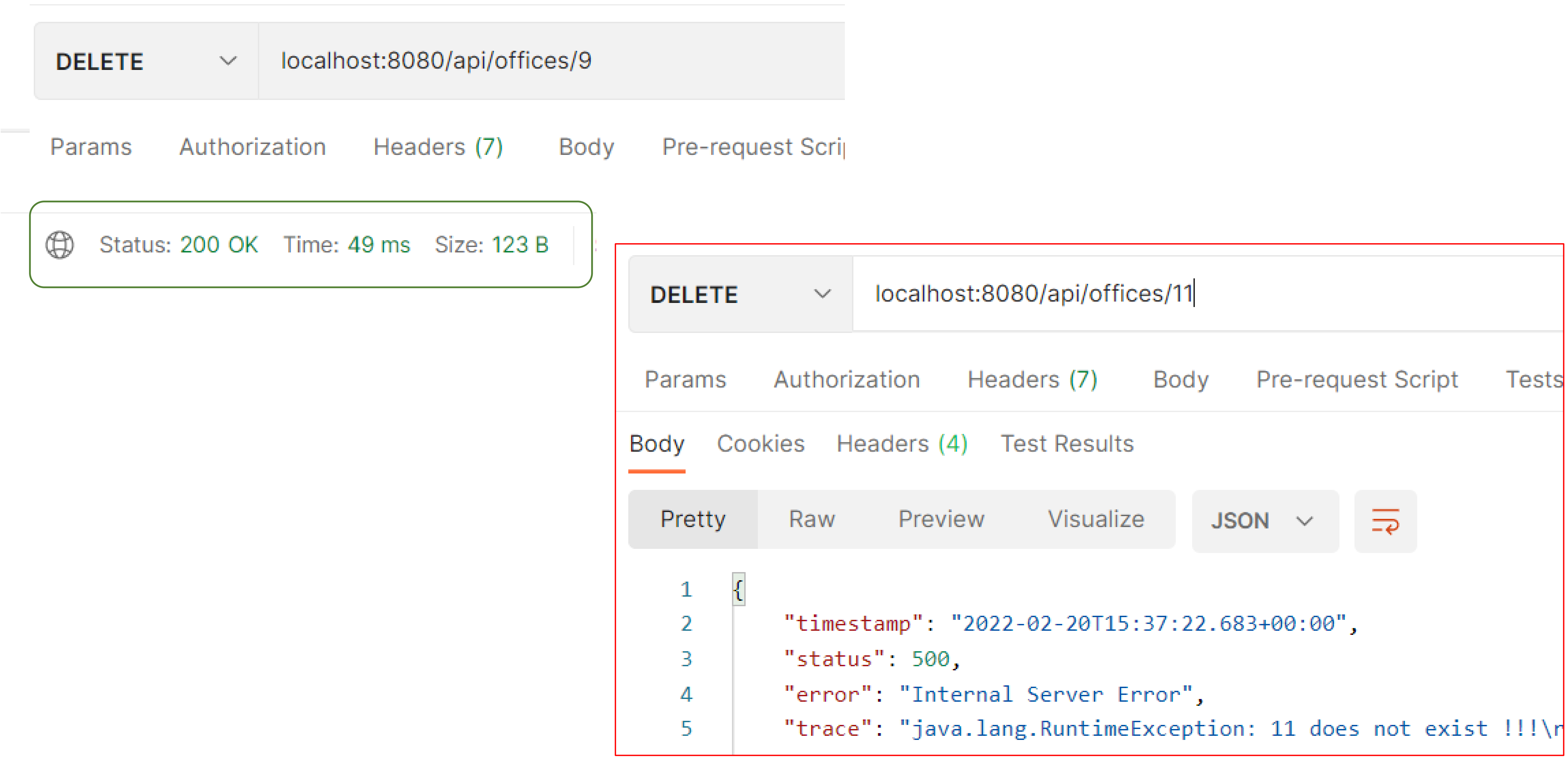
# Step 8: Testing the APIs (PUT)



PUT localhost:8080/api/offices/11

```json
{
    "id": null,
    "city": "Songkhla",
    "phone": "+44 20 7877 2041",
    "addressLine1": "25 Old Broad Street",
    "addressLine2": "Level 7",
    "state": null,
    "country": "UK",
    "postalCode": "EC2N 1HN",
    "territory": "EMEA"
}
```

# Step 8: Testing the APIs (DELETE)



DELETE localhost:8080/api/offices/9

Status: 200 OK Time: 49 ms Size: 123 B

DELETE localhost:8080/api/offices/11

```
{
    "timestamp": "2022-02-20T15:37:22.683+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "trace": "java.lang.RuntimeException: 11 does not exist !!!\r
```