

## **Hands-on Lab API REST utilizando Spring, Java y VSCode**

### **Duración:**

120 min

### **Contactos:**

Bryan Oscullo (+593 98380 3012) – bryanoscullo@gmail.com

### **Objetivo**

El objetivo de este laboratorio es poner en práctica los conocimientos obtenidos a lo largo de la capacitación en programación orientada a objetos y Java, desarrollando un API RESTFUL sencilla utilizando el framework Spring Boot.

### **Prerrequisitos**

Este laboratorio asume que se tiene:

- Entendimiento básico sobre el lenguaje de programación Java.
- Conocimientos básicos de bases de datos relacionales, JSON y el protocolo HTTP
- Un entorno de desarrollo configurado previamente (guía de laboratorio para configuración de entorno de desarrollo).

### **Requerimientos de software y sistema**

- Windows 10 home
- Java – JDK 8
- Visual Studio Code
- Paquete de extensiones de Java y Spring Boot para Visual Studio Code.

## Ejercicio 0: Creación del proyecto

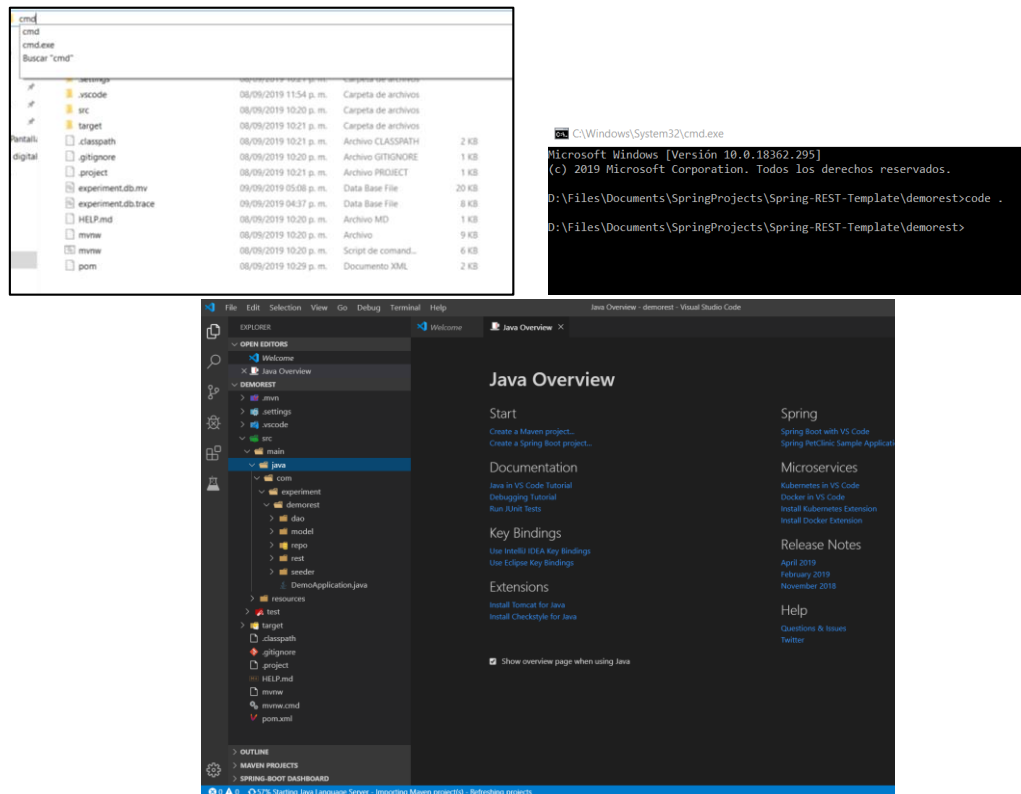
**Duración estimada:** 10 minutos

1. Copiar en la ubicación de su preferencia el proyecto plantilla que se encuentra en el siguiente repositorio:

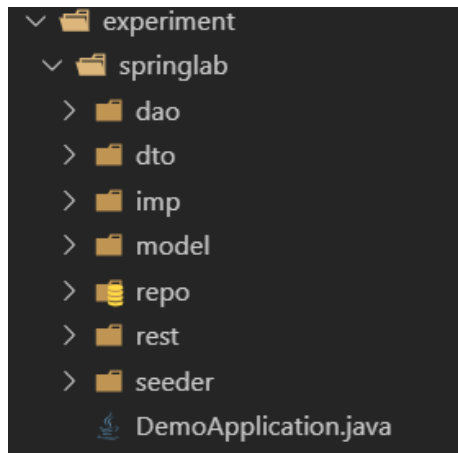
[https://github.com/BryanWladimir/Experiment\\_POO\\_PP/tree/master/Templates\\_Lab/Spring-REST-Template](https://github.com/BryanWladimir/Experiment_POO_PP/tree/master/Templates_Lab/Spring-REST-Template)

Nombre	Fecha de modifica...	Tipo	Tamaño
.vscode	30/08/2019 10:03 ...	Carpeta de archivos	
dbpizza	31/08/2019 10:56 ...	Carpeta de archivos	
demo	17/08/2019 01:45 ...	Carpeta de archivos	
demo_test	31/08/2019 08:40 ...	Carpeta de archivos	
pizzarest	27/08/2019 04:54 ...	Carpeta de archivos	
<input checked="" type="checkbox"/> Spring-REST-Template	08/09/2019 10:20 ...	Carpeta de archivos	
testpizzarest	01/09/2019 03:45 ...	Carpeta de archivos	
demo_test.7z	30/08/2019 11:04 ...	Archivo 7Z	50 KB

2. Abrir el proyecto **springlab** con el Visual Studio Code. Ejecutar una consola de comandos dentro del proyecto y el comando **code** .



- 3. Encontrará la siguiente estructura de paquetes:**



- **DAO:** Paquete de interfaces de objetos de acceso a datos.
- **IMP:** Paquete de las implementaciones de DAO.
- **DTO:** Paquete de objetos de transferencia de datos, importante para la comunicación entre los datos que maneja el servicio REST y la base de datos.
- **MODEL:** Paquete de las entidades de JPA para la persistencia de la base de datos. Representación de las tablas de base de datos.
- **REPO:** Paquete de interfaces para la creación del repositorio JPA.
- **REST:** Paquete de servicios REST
- **SEEDER:** Paquete de inicialización de información en la base de datos.

## Ejercicio 1: Creación de modelos de base de datos

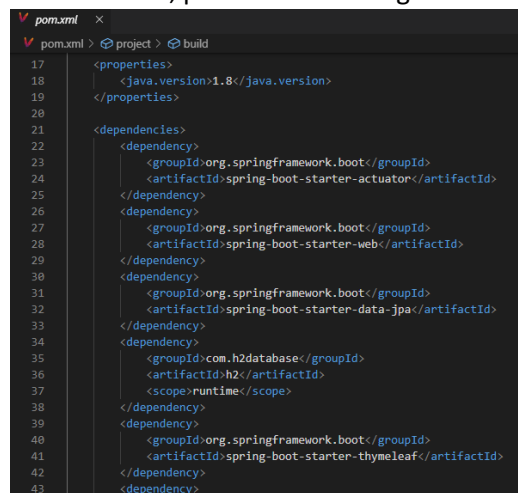
Duración estimada: 30 minutos

1. Configuración de la base de datos integrada (H2) donde se almacenará la información. Editar el archivo ***application.properties*** que se encuentra en el paquete ***src/main/resources***, de la siguiente manera:

```
server.port = 5000
spring.datasource.url=jdbc:h2:file:./experiment.db
```

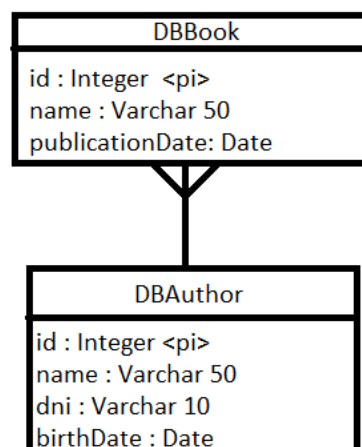
El número de puerto para el servicio y el nombre de la base de datos puede ser de su preferencia

2. La plantilla de proyecto ya viene configurada con las dependencias necesarias para el funcionamiento de los servicios REST, puede ver la configuración en el archivo ***pom.xml***.



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>experiment</artifactId>
  <version>1.0.0</version>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
  </dependencies>
</project>
```

3. Se utilizará el siguiente modelo de base de datos, en base al cual se crearán las entidades en el paquete ***model***.



4. Crear una clase **DBAuthor.java** en el paquete **model**, agregar la etiqueta **@Entity** para reconocerlo como clase de persistencia.

```
@Entity
public class DBAuthor{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Size(max = 50)
    @NotNull
    @Column(name = "name")
    private String name;

    @Size(max = 10)
    @NotNull
    @Column(name = "dni")
    private String dni;

    @Column(name = "birthDate")
    @Temporal(TemporalType.DATE)
    private Date birthDate;

    @OneToMany(mappedBy = "author")
    private List<DBBook> books;
}
```

Los atributos son los mismos de la base de datos, las etiquetas que se utilizan en cada uno sirven para identificar su tipo de dato como si de SQL se tratara, adicionalmente establecemos la relación de uno a muchos con **DBBook** mediante una lista de los mismos.

5. Agregamos constructores, setters y getters de los atributos.

```
public DBAuthor(){
}

public DBAuthor(Integer id){
    this.id = id;
}

public DBAuthor(String name, String dni, Date birthDate){
    this.name = name;
    this.dni = dni;
    this.birthDate = birthDate;
}

//Setters & Getters
```

6. No olvidarse del paquete y las importaciones en la clase.

```
package ec.edu.espe.experiment.springlab.model;
import java.util.Date;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
```

7. El mismo proceso anterior repetirlo para la entidad **DBBook**, creando la clase **DBBook.java** en el paquete **model** con la siguiente estructura.

```
package ec.edu.espe.experiment.springlab.model;

//Importaciones

@Entity
public class DBBook{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Size(max = 50)
    @NotNull
    @Column(name = "name")
    private String name;

    @Column(name = "publicationDate")
    @Temporal(TemporalType.DATE)
    private Date publicationDate;

    @JoinColumn(name = "idAuthor")
    @ManyToOne
    private DBAuthor author;

    public DBBook(){
```

```

    }

    public DBBook(Integer id){
        this.id = id;
    }

    public DBBook(String name, Date publicationDate, DBAuthor
author){
        this.name = name;
        this.publicationDate = publicationDate;
        this.author = author;
    }
    //Setters & Getters
}

```

8. Una vez creado el modelo de base de datos configurar el repositorio JPA, en el paquete **repo** crear las interfaces **IAuthorRepo.java** y **IBookRepo.java**, de la siguiente forma:

#### **IBookRepo.java**

```

package ec.edu.espe.experiment.springlab.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import ec.edu.espe.experiment.springlab.model.DBBook;
public interface IBookRepo extends JpaRepository<DBBook, Integer>{

}

```

#### **IAuthorRepo.java**

```

package ec.edu.espe.experiment.springlab.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import ec.edu.espe.experiment.springlab.model.DBAuthor;
public interface IAuthorRepo extends JpaRepository<DBAuthor, Integer>{

}

```

En cada repositorio se pasan como parámetros la entidad y el tipo de dato de la clave primaria de cada una. Estos repositorios serán inyectados como dependencias más adelante para el manejo de la base de datos.

## Ejercicio 2: Creación de DTO y DAO (Objetos de transferencia de datos y Objetos de acceso a datos)

Duración estimada: 40 minutos

1. Las entidades de repositorio no se pueden utilizar para la estructura de datos de los servicios REST, esto debido a los Join de referencia de una a otro, al parsear estos datos a JSON se crearía un loop infinito en donde un autor muestre una lista de libros, pero a su vez cada libro mostraría su autor repitiendo el ciclo una y otra vez. Para solucionar esto se utilizan los **DTO** como estructura de datos a nivel de servicio.
2. En el paquete **dto** crear las clases de estructura de dato: **Author.java** y **Book.java**, con la siguiente estructura:

### **Author.java**

```
package ec.edu.espe.experiment.springlab.dto;

import java.util.Date;
import java.util.List;

public class Author{

    private Integer id;
    private String name;
    private String dni;
    private Date birthDate;
    private List<Book> books;

    public Author(){

    }

    public Author(Integer id){
        this.id = id;
    }

    public Author(Integer id, String name, String dni, Date birthDate, List<Book> books){
        this.id = id;
        this.name = name;
        this.dni = dni;
        this.birthDate = birthDate;
        this.books = books;
    }

    //Setters & Getters
}
```



**Book.java**

```
package ec.edu.espe.experiment.springlab.dto;

import java.util.Date;

public class Book{

    private Integer id;
    private String name;
    private Date publicationDate;
    private Integer idAuthor;

    public Book(){

    }

    public Book(Integer id){
        this.id = id;
    }

    public Book(Integer id, String name, Date publicationDate, Integer idAuthor){
        this.id = id;
        this.name = name;
        this.publicationDate = publicationDate;
        this.idAuthor = idAuthor;
    }

    //Setters & Getters

}
```

3. Crear los objetos de acceso a datos **DAO**, en el paquete **dao** agregar dos interfaces, una por cada entidad de base de datos: **IAuthorDAO.java** y **IBookDAO.java**, con la siguiente estructura:

**IAuthorDAO.java**

```
package ec.edu.espe.experiment.springlab.dao;

import java.util.List;

import ec.edu.espe.experiment.springlab.dto.Author;
```

```
import ec.edu.espe.experiment.springlab.model.DBAuthor;

public interface IAuthorDAO{
    public List<Author> getAll();
    public Author get(Integer id);
    public Author post(Author author);
    public Author toAuthor(DBAuthor dbAuthor);
}
```

#### ***IBookDAO.java***

```
package ec.edu.espe.experiment.springlab.dao;

import java.util.List;

import ec.edu.espe.experiment.springlab.dto.Book;
import ec.edu.espe.experiment.springlab.model.DBBook;

public interface IBookDAO{
    public List<Book> getAll();
    public Book get(Integer id);
    public Book post(Book book);
    public Book toBook(DBBook dbBook);
}
```

Estas interfaces luego serán inyectadas como dependencias para su uso en los servicios, primero hay que implementarlas. Como se puede observar se manejarán funcionalidades básicas como: obtener todos los registros, registro por identificador y agregar un nuevo registro.

4. En el paquete ***imp*** crear una clase ***BookDAO.java*** que implemente ***IBookDAO*** para sobre escribir sus métodos, agregar la etiqueta ***@Repository*** a la clase para identificar que tendrá acceso a los repositorios.

```
package ec.edu.espe.experiment.springlab.imp;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import ec.edu.espe.experiment.springlab.dao.IBookDAO;
import ec.edu.espe.experiment.springlab.dto.Book;
import ec.edu.espe.experiment.springlab.model.DBAuthor;
import ec.edu.espe.experiment.springlab.model.DBBook;
import ec.edu.espe.experiment.springlab.repo.IAuthorRepo;
import ec.edu.espe.experiment.springlab.repo.IBookRepo;

@Repository
public class BookDAO implements IBookDAO{

    @Autowired
    private IBookRepo repo;

    @Autowired
    private IAuthorRepo repoAuthor;

}

```

Con la etiqueta **@Autowired** inyectamos los repositorios como dependencias, esto nos permite hacer usos de los métodos de repositorio sin la necesidad de instanciar un objeto.

```

@Override
public List<Book> getAll(){
    List<Book> list = new ArrayList<>();
    try {
        List<DBBook> list_dbBook = repo.findAll();
        if (list_dbBook != null) {
            for (DBBook dbBook : list_dbBook) {
                list.add(toBook(dbBook));
            }
        }
    } catch (Exception e) {
        list = new ArrayList<>();
    }
    return list;
}

```

Sobre escribimos el método **getAll** en donde se recupera todos los registros de tipo **DBBook** de la base de datos y son parseados al tipo **Book** para ser enviados por el servicio REST, el método que realiza el parseo es el siguiente:

```
@Override
    public Book toBook(DBBook dbBook){
        return new Book(dbBook.getId(),
            dbBook.getName(),
            dbBook.getPublicationDate(),
            dbBook.getAuthor().getId());
    }
```

De la misma manera sobre escribimos los dos métodos restantes: **get** para encontrar un registro por su **id** y **post** para almacenar un registro en la base de datos.

```
@Override
    public Book get(Integer id){
        Book book = null;
        try {
            Optional<DBBook> dbBook = repo.findById(id);
            if(dbBook != null){
                book = toBook(dbBook.get());
            }
        } catch (Exception e) {
            book = null;
        }
        return book;
    }

    @Override
    public Book post(Book book){
        Book response;
        try{
            Optional<DBAuthor> dbAuthor = repoAuthor.findById(book.getIdAuthor());
            DBBook dbBook = new DBBook(book.getName(),
                book.getPublicationDate(),
                dbAuthor.get());
            repo.save(dbBook);
            repo.flush();
            response = toBook(dbBook);
        }
    }
```

```

        catch(Exception e){
            response = null;
        }
        return response;
    }

```

5. Repetimos el mismo proceso para autor, creando la clase **AuthorDAO.java**, con el siguiente código:

```

package ec.edu.espe.experiment.springlab.imp;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
//imports
@Repository
public class AuthorDAO implements IAuthorDAO {
    @Autowired
    private IAuthorRepo repo;

    @Autowired
    private IBookDAO daoBook;

    @Override
    public List<Author> getAll() {
        List<Author> list = new ArrayList<>();
        try {
            List<DBAuthor> list_dbAuthor = repo.findAll();
            if (list_dbAuthor != null) {
                for (DBAuthor dbAuthor : list_dbAuthor) {

                    list.add(toAuthor(dbAuthor));
                }
            }
        } catch (Exception e) {
            list = new ArrayList<>();
        }
        return list;
    }

    @Override

```

```

public Author get(Integer id) {
    Author author = null;
    try {
        Optional<DBAuthor> dbAuthor = repo.findById(id);
        if(dbAuthor != null){
            author = toAuthor(dbAuthor.get());
        }
    } catch (Exception e) {
        author = null;
    }
    return author;
}

@Override
public Author post(Author author) {
    Author response = null;
    try {
        DBAuthor dbAuthor = new DBAuthor(author.getName(),
            author.getDni(),
            author.getBirthDate());
        repo.save(dbAuthor);
        repo.flush();
        response = toAuthor(dbAuthor);
    } catch (Exception e) {
        response = null;
    }
    return response;
}

@Override
public Author toAuthor(DBAuthor dbAuthor){
    List<Book> books = new ArrayList<>();
    if(dbAuthor.getBooks() != null){
        for(DBBook dbBook : dbAuthor.getBooks()){
            books.add(daoBook.toBook(dbBook));
        }
    }
    return new Author(dbAuthor.getId(),
        dbAuthor.getName(),
        dbAuthor.getDni(),
        dbAuthor.getBirthDate(),
        books);
}
}

```

### Ejercicio 3: Creación de rutas y semilla de datos

Duración estimada: 20 minutos

1. Con el repositorio y el acceso a datos implementado solo resta crear el servicio REST. En el paquete **rest** crear la clase **RestAuthorController.java** para definir los servicios que manejen los datos de autor. Son necesarias agregar dos etiquetas a la clase: **@RestController** para identificar que esta clase será un controlador de servicios y **@RequestMapping("/api/author")** para definir una ruta de acceso a estos servicios.
2. Inyectar la dependencia de **IAuthorDAO** para poder utilizar los métodos de acceso a datos, mediante:

```
@Autowired
private IAuthorDAO dao;
```

3. Primer servicio a implementar es el de recuperar todos los datos de autor, de la siguiente manera:

```
@GetMapping
public ResponseEntity<List<Author>> getAll() {
    List<Author> response = dao.getAll();
    if (response != null) {
        return ResponseEntity.status(HttpStatus.OK).body(response);
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
    }
}
```

La etiqueta **@GetMapping** identifica que este servicio es de tipo **GET**, el **ResponseEntity** permite responder al cliente del servicio un estatus de su petición más la respuesta

4. De la misma forma para implementar el **getById** y el **post**:

```
@GetMapping(value =("/{id}")
public ResponseEntity<Author> get(@PathVariable("id") Integer id) {
    Author response = dao.get(id);
    if (response != null) {
        return ResponseEntity.status(HttpStatus.OK).body(response);
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
    }
}

@PostMapping
```

```

public ResponseEntity<Author> post(@RequestBody Author entity) {
    Author response = dao.post(entity);
    if (response != null) {
        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    } else {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }
}

```

5. Repetimos el proceso anterior para implementar los servicios de libro en una clase ***RestBookController.java***:

```

package ec.edu.espe.experiment.springlab.rest;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import ec.edu.espe.experiment.springlab.dao.IBookDAO;
import ec.edu.espe.experiment.springlab.dto.Book;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@RestController
@RequestMapping("/api/book")
public class RestBookController {

    @Autowired
    private IBookDAO dao;

    @GetMapping
    public ResponseEntity<List<Book>> getAll() {
        List<Book> response = dao.getAll();
        if (response != null) {
            return ResponseEntity.status(HttpStatus.OK).body(response);
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
        }
    }
}

```



```

    @GetMapping(value =("/{id}")
    public ResponseEntity<Book> get(@PathVariable("id") Integer id) {
        Book response = dao.get(id);
        if (response != null) {
            return ResponseEntity.status(HttpStatus.OK).body(response);
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
        }
    }

    @PostMapping
    public ResponseEntity<Book> post(@RequestBody Book entity) {
        Book response = dao.post(entity);
        if (response != null) {
            return ResponseEntity.status(HttpStatus.CREATED).body(response);
        } else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }
}

```

6. Como último paso antes de ejecutar los servicios se puede crear un componente semilla que inicialice datos en la base. En el paquete **seeder** crear la clase **DatabaseSeeder.java** con la siguiente estructura:

```

package ec.edu.espe.experiment.springlab.seeder;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import ec.edu.espe.experiment.springlab.model.DBAuthor;
import ec.edu.espe.experiment.springlab.model.DBBook;
import ec.edu.espe.experiment.springlab.repo.IAuthorRepo;
import ec.edu.espe.experiment.springlab.repo.IBookRepo;

@Component
public class DatabaseSeeder implements CommandLineRunner {

```

```
@Autowired
private IAuthorRepo repoAuthor;

@Autowired
private IBookRepo repoBook;

@Autowired
public DatabaseSeeder(IAuthorRepo authorRepo){
    this.repoAuthor = authorRepo;
}

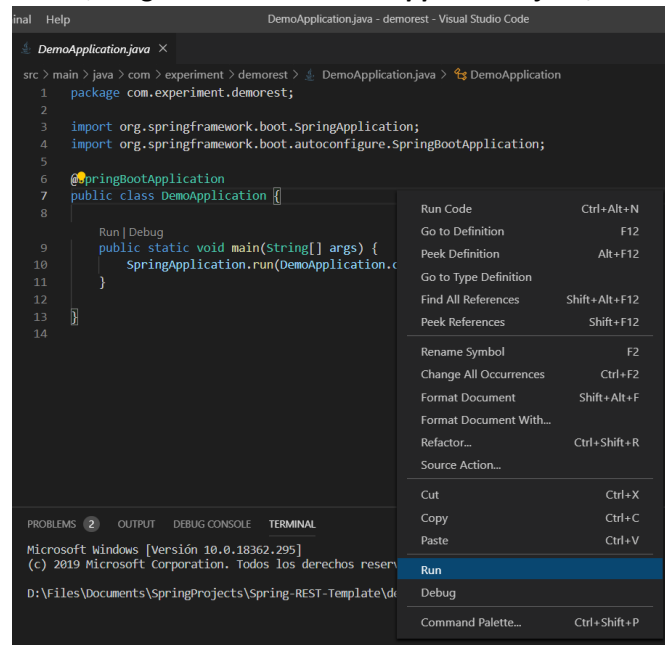
@Override
public void run(String... strings) throws Exception {
    List<DBAuthor> authors = new ArrayList<>();
    authors.add(new DBAuthor("Autor 1", "1718152269", new Date()));
    repoAuthor.saveAll(authors);

    List<DBBook> books = new ArrayList<>();
    books.add(new DBBook("Libro 1", new Date(), authors.get(0)));
    repoBook.saveAll(books);
}
}
```

## Ejercicio 4: Ejecución y prueba de los servicios

Duración estimada: 20 minutos

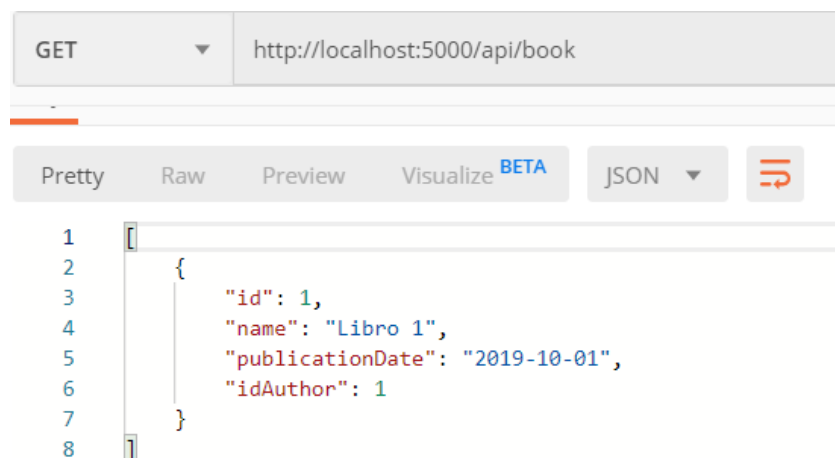
1. Para correr los servicios, dirigirse a la clase **DemoApplication.java**, click derecho y **Run**.



2. De ser exitoso nos mostrará un mensaje que el servidor ha sido iniciado en el puerto 5000

```
: Exposing 2 endpoint(s) beneath base path '/actuator'
: Tomcat started on port(s): 5000 (http) with context path=/
: Started DemoApplication in 9.6 seconds (JVM running)
```

## Consultar libros



## Consultar autores

GET http://localhost:5000/api/author

Pretty Raw Preview Visualize BETA JSON ↺

```
1  [
2    {
3      "id": 1,
4      "name": "Autor 1",
5      "dni": "1718152269",
6      "birthDate": "2019-10-01",
7      "books": [
8        {
9          "id": 1,
10         "name": "Libro 1",
11         "publicationDate": "2019-10-01",
12         "idAuthor": 1
13       }
14     ]
15   }
16 ]
```

## Agregar autor

POST http://localhost:5000/api/author

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL BETA **JSON**

```
1 {
2   "name": "Autor 2",
3   "dni": "1717196693",
4   "birthDate": "2019-10-01"
5 }
```

body Cookies Headers (3) Test Results **Status: 201 Created**

Pretty Raw Preview Visualize BETA JSON ↺

```
1 {
2   "id": 2,
3   "name": "Autor 2",
4   "dni": "1717196693",
5   "birthDate": "2019-10-01T00:00:00.000+0000",
6   "books": []
7 }
```

```
GET http://localhost:5000/api/author

Pretty Raw Preview Visualize BETA JSON

1 [
2   {
3     "id": 1,
4     "name": "Autor 1",
5     "dni": "1718152269",
6     "birthDate": "2019-10-01",
7     "books": [
8       {
9         "id": 1,
10        "name": "Libro 1",
11        "publicationDate": "2019-10-01",
12        "idAuthor": 1
13      }
14    ]
15  },
16  {
17    "id": 2,
18    "name": "Autor 2",
19    "dni": "1717196693",
20    "birthDate": "2019-09-30",
21    "books": []
22  }
23 ]
```

## Agregar libro

```
POST http://localhost:5000/api/book

Params Authorization Headers (9) Body Pre-request Script Tests Settings
● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL BETA JSON

1 {
2   "name": "Libro 2",
3   "publicationDate": "2019-10-01",
4   "idAuthor": 1
5 }

Body Cookies Headers (3) Test Results Status: 201 Created

Pretty Raw Preview Visualize BETA JSON

1 {
2   "id": 2,
3   "name": "Libro 2",
4   "publicationDate": "2019-10-01T00:00:00.000+0000",
5   "idAuthor": 1
6 }
```

### Consultar autor por id

```
GET http://localhost:5000/api/author/1

Pretty Raw Preview Visualize BETA JSON ↺
```

```
1 {
2   "id": 1,
3   "name": "Autor 1",
4   "dni": "1718152269",
5   "birthDate": "2019-10-01",
6   "books": [
7     {
8       "id": 1,
9       "name": "Libro 1",
10      "publicationDate": "2019-10-01",
11      "idAuthor": 1
12    },
13    {
14      "id": 2,
15      "name": "Libro 2",
16      "publicationDate": "2019-09-30",
17      "idAuthor": 1
18    }
19  ]
20 }
```

### Consultar libro por id

```
GET http://localhost:5000/api/book/2

Pretty Raw Preview Visualize BETA JSON ↺
```

```
1 {
2   "id": 2,
3   "name": "Libro 2",
4   "publicationDate": "2019-09-30",
5   "idAuthor": 1
6 }
```