CREACIÓN DE UN SENCILLO API Restful con operaciones CRUD

NOTA: ANTES DE EMPEZAR, SIGA LAS INSTRUCCIONES DE INSTALACIÓN DE LOMBOK en https://projectlombok.org/setup/eclipse (aplican las mismas instrucciones para STS. Si no usa STS, también hay instrucciones para IntelliJ)

1. Creamos un nuevo proyecto, llamado ContactsAPI (New → Spring Starter Project)
    a. Name: ContactsAPI
    b. Group: com.cenfotec.contacts
    c. Artifact: ContactsAPI
    d. Package: com.cenfotec.contacts.
2. Agregue las siguientes dependencias: Web (Spring Web Starter), JPA, Lombok y la dependecia de MariaSQL o la de MYSQL que usó en laboratorios anteriores.

    En dicho servidor de BD, haga una BD llamada contacts, y en ella crea la tabla contact.

    El script de la tabla es:

```
CREATE TABLE contact
( id INT(11) NOT NULL AUTO_INCREMENT,
name VARCHAR(100) NOT NULL,
email VARCHAR(100),
phone VARCHAR(12),
CONSTRAINT sites_pk PRIMARY KEY (id)
);
```

3. Creamos el paquete com.cenfotec.contacts.model

4. En el paquete recién creado creamos la clase Contact. Con el siguiente código

```
@AllArgsConstructor
@NoArgsConstructor
@Data
@Entity
public class Contact {

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

  private String name;
  private String email;
  private String phone;
}
```

(import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;)

5. Ahora creamos el paquet com.cenfotec.contacts.repository
6. En el paquete recién creado creamos la clase ContactRepository. Con el siguiente código.

```
@Repository
public interface ContactRepository
    extends JpaRepository<Contact, Long> { }
```

(
```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.cenfotec.contacts.model.Contact;
```
)

7. Creamos el paquete com.cenfotec.contacts.controllers
8. En el paquete recién creado, creamos la clase ContactController, con el siguiente código

```
@RestController
@RequestMapping({"/contacts"})
public class ContactController {
  private ContactRepository repository;

  ContactController(ContactRepository contactRepository) {
      this.repository = contactRepository;
  }

}
```

Como verá, solo agregamos el constructor.

(
```
import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.cenfotec.contacts.model.Contact;
import com.cenfotec.contacts.repository.ContactRepository;
```
)

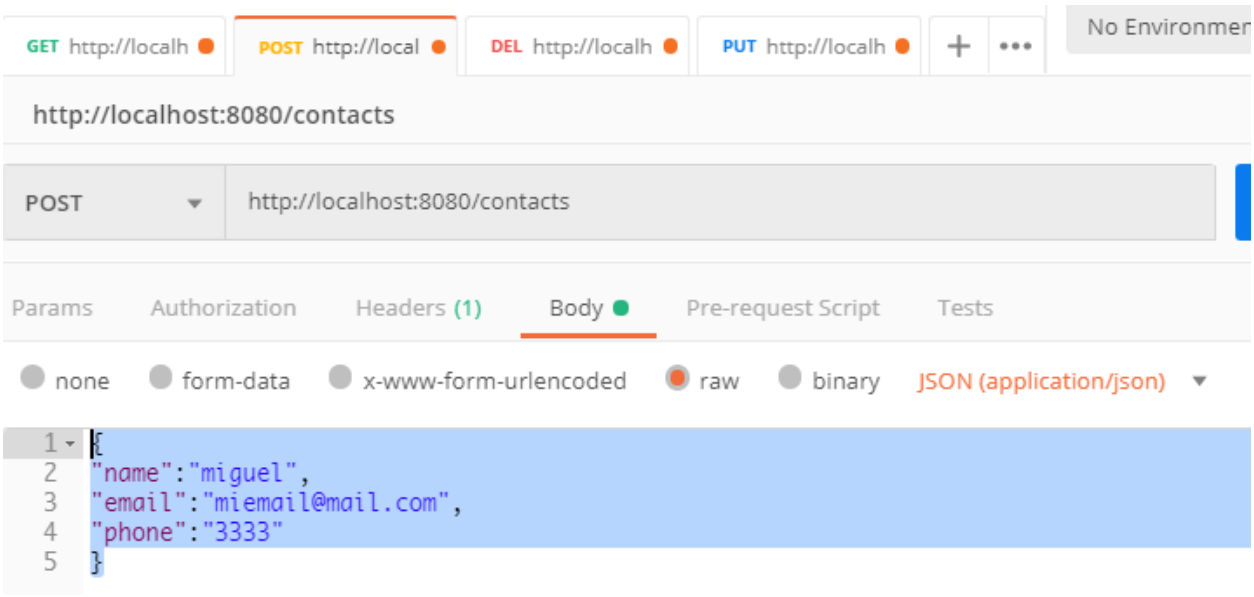
9. Ahora en esa misma clase, agregue el siguiente código (consejo, agregue 1 método a la vez):

```
@GetMapping
public List findAll(){
  return repository.findAll();
}

@GetMapping(path = {"/{id}"})
public ResponseEntity<Contact> findById(@PathVariable
long id){
  return repository.findById(id)
          .map(record ->
ResponseEntity.ok().body(record))
          .orElse(ResponseEntity.notFound().build());
}

@PostMapping
public Contact create(@RequestBody Contact contact){
    return repository.save(contact);
}

@PutMapping(value="/{id}")
  public ResponseEntity<Contact>
update(@PathVariable("id") long id,
       @RequestBody Contact contact){
    return repository.findById(id)
        .map(record -> {
            record.setName(contact.getName());
            record.setEmail(contact.getEmail());
            record.setPhone(contact.getPhone());
            Contact updated = repository.save(record);
            return ResponseEntity.ok().body(updated);
        }).orElse(ResponseEntity.notFound().build());
  }
```

```
@DeleteMapping(path ={"/{id}"})
  public ResponseEntity<?> delete(@PathVariable("id")
long id) {
    return repository.findById(id)
        .map(record -> {
            repository.deleteById(id);
            return ResponseEntity.ok().build();
        }).orElse(ResponseEntity.notFound().build());
  }
```

10. Ahora instale POSTMAN (https://www.getpostman.com/downloads/) o INSOMNIA (https://insomnia.rest/) (en este laboratorio se hará referencia a POSTMAN).
11. Haga la configuración necesaria de la conexión de su aplicación a la BD (url, password y username en application.properties).
12. Una vez instalado, ábralo registrese. Luego haga un nuevo request tipo GET (selecciónelo). Dele send. Obtendrá por resultado código 200 y en el body algo como []
13. Oprima el + que esta junto al tab del request. Estará listo para crear uno nuevo. Haga que ese request sea tipo POST. En la configuración de BODY seleccione RAW y JSON(application/json). Ponga en el BODY el siguiente texto:

```
{
"name":"yayo",
"email":"miemail@mail.com",
"phone":"3333"
}
```



Dele Send.

Observe el resultado y revise la BD usando Heidy.

Ahora vuelva al get y repita el send. Observe el resultado.

14. Repita el paso anterior unas 5 veces, modificando el body para agregar distintas personas. Y observe el listado.
15. Ahora prepare un nuevo request en otro tab de postman. Esta vez será de tipo DELETE. Con el url /contacts/1. Send Observe el resultado (200) y luego repita el get. Vea como se borró el dato con ese id.
16. Ahora haga un request tipo Update. Seleccione un id que no haya borrado. El tipo del request será PUT. El URL del recurso es /contacts/[idNoBorradoSeleccionadoPorUsted] y coloque en el Body una versión MODIFICADA para él. Oprima send. Si el resultado es 200, seleccione el GET y observe como cambiaron los datos.

Preguntas:

a. Cuál es la diferencia en la práctica entre @Controller y @RestController? Cuál es mejor usar en qué casos?
b. Que tipo de inyección estamos usando en el ContactController?
c. Por qué en el findAll no se especifica el URI para la operación?

d. Qué obtiene si trata de borrar 2 veces un recurso?
e. Comente la anotación de PutMapping del método que hace el update. Reinicie la aplicación. Observe que pasa si lo trata de hacer.
f. Rehabilite la anotación en PutMapping. Pero ahora cambie el URL de POSTMAN para que apunte directamente a la colección /contacts. Qué error recibe?
g. Si no escribió los set/get de Contact… de donde salieron?
h. Que efecto y anotaciones de Lombok usó en la clase Contact?

RETO SAYAYIN:

Implemente la clase Travel, con los campos id (Long), Startdate, EndDate (LocalDate cada uno), destiny (string).

La relación entre Contact y Travel es 1-N (un contact, puede tener varios viajes).

Haga la relación.

Haga que en Contact aparezcan los viajes.

Haga el endpoint para agregar un viaje a un contacto

Haga el endpoint para listar los viajes de un Contacto.

RETO SUPER SAYAYIN DIOS(A) AZUL

Agregue varios contactos (al menos 100… puede hacerlo con un script a pie).

Usando como referencia https://www.baeldung.com/rest-api-pagination-in-spring implemente la PAGINACIÓN DE CONTACTOS.