Segunda Unidad: Desarrollo API JAVA SPRING
GUÍA DE LABORATORIO N° 10:
CREACIÓN DEL MODELO MVC - JAVA

Docente: Jaime Suasnabar Terrel

Fecha: …….. **Duración:** 90 minutos

Instrucciones: Contar con JDK Jakarta, Intelli J Idea y Tomcat.

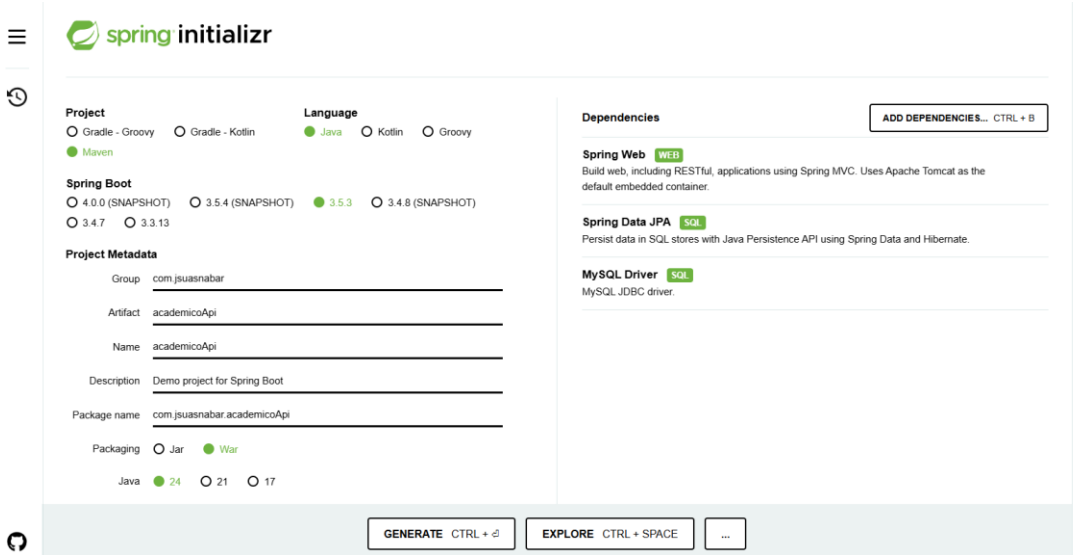
METAS

- Creación del Proyecto SPRING con SpringBoot
- Abrir el Proyecto con IntelliJ Idea y Configurar Ejecución de SpringBoot
- Configurar properties para MySQL y Crear los paquetes controllers, models, repositories y services
- Creación del modelo Estudiante
- Creación del EstudianteRepository
- Creación del EstudianteController
- Comprobar CRUD API con Posman

PROCEDIMIENTO

EJERCICIO 01. CREACIÓN DEL PROYECTO CON SPRING-BOOT

1. En https://start.spring.io/ configurar un proyecto Maven con la siguiente estructura



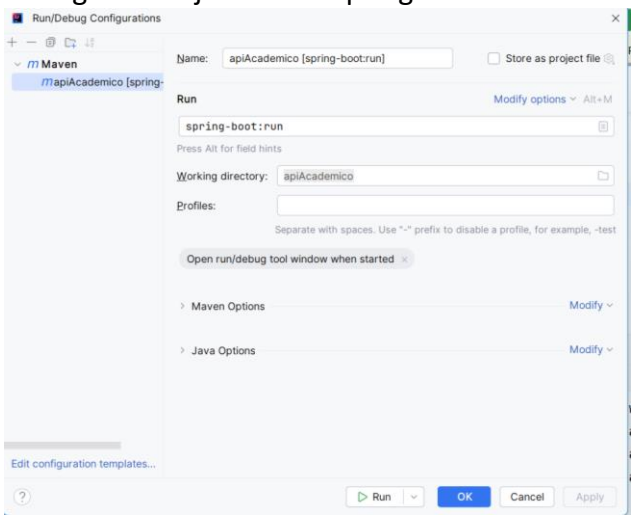
Descargar el proyecto con el botón GENERATE en un archivo *.zip

Descargar

Se descomprime el archivo zip descargado.

EJERCICIO 02. ABRIR EL PROYECTO CON INTELLIJ IDEA Y CONFIGURAR EJECUCIÓN DE SPRINGBOOT

1. Abrir en IntelliJ Idea el Proyecto Maven.
2. Verificar la instalación de librerías
3. Configurar la ejecicón de spring-boot

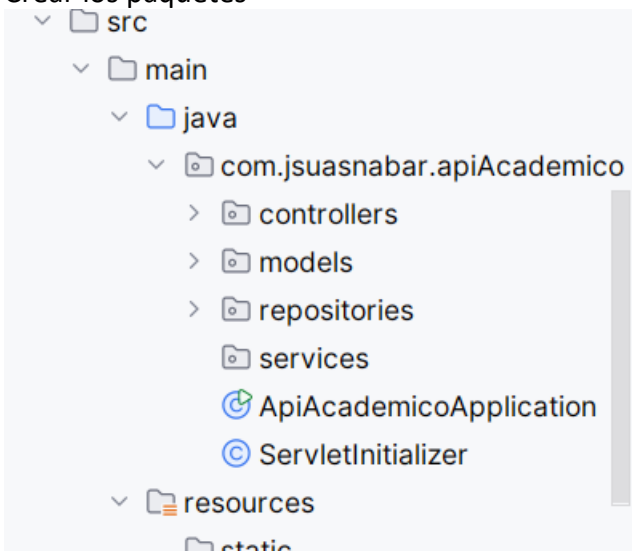


EJERCICIO 03. CONFIGURAR PROPERTIES PARA MYSQL Y CREAR LOS PAQUETES CONTROLLERS, MODELS, REPOSITORIES Y SERVICES

1. Confiigurar Properties del Proyecto.

```
spring.datasource.url=jdbc:mysql://localhost:3306/academico
spring.datasource.username=root
spring.datasource.password=123456
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

2. Crear los paquetes



EJERCICIO 04. CREACIÓN DEL MODELO ESTUDIANTE

```
package com.jsuasnabar.apiAcademico.models;

import jakarta.persistence.*;

@Entity
@Table(name = "estudiante")
public class Estudiante {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long idEstudiante;

   private String nomEstudiante;
   private String dirEstudiante;
   private String ciuEstudiante;

   // Getters y setters

   public String getNomEstudiante() {
      return nomEstudiante;
   }

   public void setNomEstudiante(String nomEstudiante) {
      this.nomEstudiante = nomEstudiante;
   }

   public String getDirEstudiante() {
      return dirEstudiante;
   }

   public void setDirEstudiante(String dirEstudiante) {
      this.dirEstudiante = dirEstudiante;
   }

   public String getCiuEstudiante() {
      return ciuEstudiante;
   }

   public void setCiuEstudiante(String ciuEstudiante) {
      this.ciuEstudiante = ciuEstudiante;
   }
}
```

EJERCICIO 05. CREACIÓN DEL ESTUDIANTEREPOSITORY

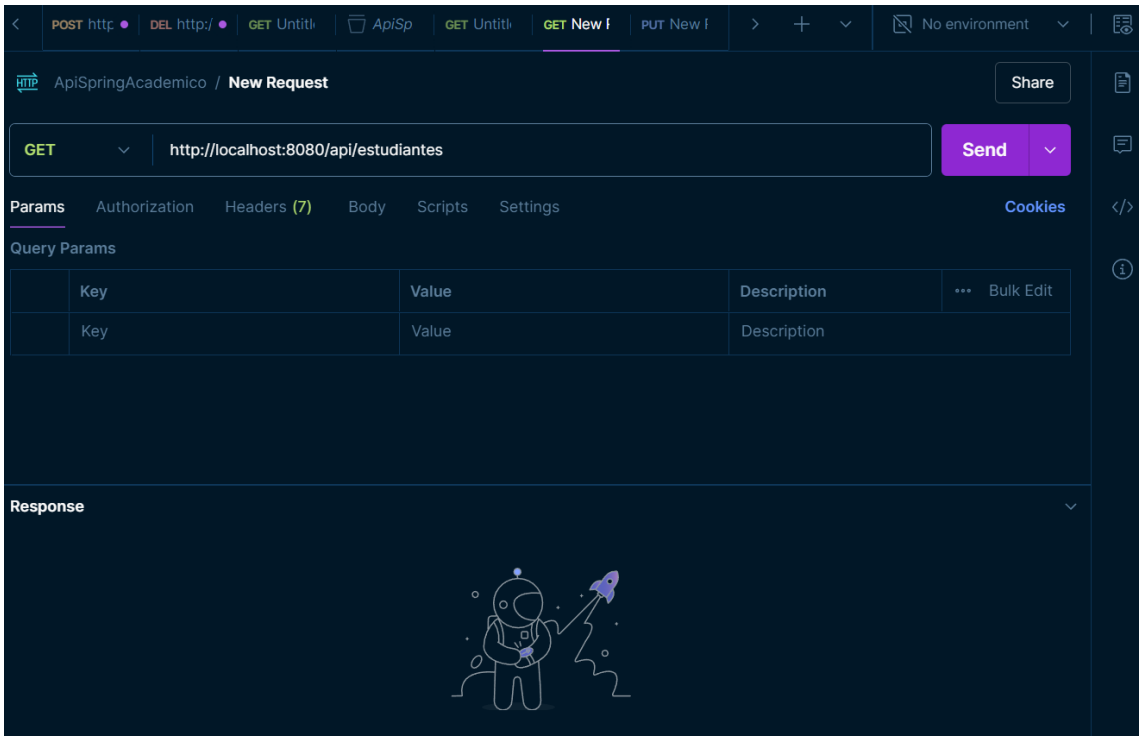
```
package com.jsuasnabar.apiAcademico.repositories;

import com.jsuasnabar.apiAcademico.models.Estudiante;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EstudianteRepository extends JpaRepository<Estudiante, Long>{
}
```

EJERCICIO 06. CREACIÓN DEL ESTUDIANTECONTROLLER

```
package com.jsuasnabar.apiAcademico.controllers;

import com.jsuasnabar.apiAcademico.models.Estudiante;
import com.jsuasnabar.apiAcademico.repositories.EstudianteRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/estudiantes")
public class EstudianteController {

   @Autowired
   private EstudianteRepository repo;

   @PostMapping
   public Estudiante agregar(@RequestBody Estudiante estudiante) {
      return repo.save(estudiante);
   }

   @PutMapping("/{id}")
   public ResponseEntity<Estudiante> modificar(@PathVariable Long id, @RequestBody
Estudiante datos) {
      return repo.findById(id)
            .map(est -> {
               est.setNomEstudiante(datos.getNomEstudiante());
               est.setDirEstudiante(datos.getDirEstudiante());
               est.setCiuEstudiante(datos.getCiuEstudiante());
               return ResponseEntity.ok(repo.save(est));
            })
            .orElse(ResponseEntity.notFound().build());
   }

   @DeleteMapping("/{id}")
   public ResponseEntity<Void> eliminar(@PathVariable Long id) {
      if (repo.existsById(id)) {
```

```
            repo.deleteById(id);
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }

    @GetMapping
    public List<Estudiante> listarTodos() {
        return repo.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Estudiante> buscarPorId(@PathVariable Long id) {
        return repo.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
}
```

EJERCICIO 07. COMPROBAR EN POSMAN



Practica Calificada 10

Desarrollar una API RESTful para la gestión de docentes utilizando Jakarta EE *y* Spring. La aplicación permitirá realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar y Otras) sobre una entidad llamada **Docente**, que estará asociada a una base de datos MySQL.

REQUISITOS DE INFORMACIÓN

1. **Entidad Docente:**
   - idDocente
   - nomDocente
   - dirDocente
   - ciuDocente
   - emailDocente
   - fecNacimiento
   - tiempoServicio

REQUISITOS FUNCIONALES

2. **Operaciones a implementar en la API:**
   - Lista todos los docentes
   - Obtiene un docente por ID
   - Crea un nuevo docente
   - Actualiza los datos de un docente
   - Elimina un docente
   - Listar todos los docentes que residen en una ciudad específica (EndPoint: /api/docentes/ciudad/Cusco) (2 PUNTOS)
   - Listar los docentes con al menos cierta cantidad de años de servicio (EndPoint: /api/docentes/experiencia/10. (2 PUNTOS)
   - Calcula y devuelve la edad promedio de todos los docentes registrados (EndPoint: /api/docentes/edad-promedio. (2 PUNTOS)
   - Agregar documentación Swagger. (2 PUNTOS)
   - Manejo de excepciones globales. (2 PUNTOS)
   - Listar los docentes para paginación en el listado de docentes (EndPoint: GET /api/docentes?page=0&size=10) page=0 es la primera página. size=10 indica que se devolverán 10 docentes por página. (8 PUNTOS)
3. **Validaciones mínimas:**
   - El emailDocente debe tener el formato de correo (2 PUNTOS)
   - El tiempoServicio no puede ser negativo. (2 PUNTOS)
   - La fecNacimiento debe ser anterior a la fecha actual. (2 PUNTOS)
4. **Framework:**
   Utiliza Jakarta versión 24, SPRING y Spring Data y MySQL.

5. **Demostración:**
   POSMAN.

Ejercicios Propuestos

1. Desarrolle un controlador de API que compruebe que el id y nota de la ruta "/student/{id}/{nota}" son números. En caso de certeza mostrar un json que contiene el id, nota, un mensaje de ruta válida y el estado 202.
2. Desarrolle un controlador de API con los id y notas según la siguiente ruta "/student/{id}?nota1=xx&nota2=xx&nota3=xx". Que muestre un json que contiene el id, promedio, un mensaje de ruta válida y el estado 202.
3. Diseñar y desarrolle una API REST para que devuelva datos desde una base de datos de 3 tablas departamento(id, name), provincia(id, name) y distrito(id, name).
4. Diseñar y desarrolle una API REST para realizar las operaciones CRUD desde y hacia una base de datos de 3 tablas departamento(id, name), provincia(id, name) y distrito(id, name).