ANALISIS Y DESARROLLO DE SOFTWARE

Módulos de software codificados y probados GA7-220501096-AA2-EV02

Tutor
ALVARO ESTEBAN BETANCOURT MATOMA
Docente

Servicio Nacional de Aprendizaje – SENA
No Ficha: (2885262)

Integrante:
David Alejandro Rodríguez Rodríguez
Bogotá D.C
2023

Introducción

En el presente trabajo se desarrollará un módulo de software siguiendo estándares de codificación y pruebas para asegurar su calidad y funcionalidad. Para ello, utilizaremos el framework Spring Boot, el entorno de desarrollo integrado (IDE) NetBeans y MySQL Workbench para la gestión de la base de datos. El objetivo principal es implementar un CRUD (Create, Read, Update, Delete) básico que integra los artefactos del ciclo de vida del software, cuentos como diagramas de clases, diagramas de casos de uso, historias de usuario, diseños y prototipos.

Objetivos del proyecto

1. **Codificación del módulo** : Implementar el módulo utilizando Spring Boot, asegurando que el código esté bien comentado y siga los estándares de codificación.
2. **Integración con Base de Datos** : Utilizar MySQL Workbench para gestionar la base de datos y asegurar una correcta conexión e interacción desde la aplicación.
3. **Uso de Herramientas de Versionamiento** : Crear y mantener el proyecto utilizando un sistema de control de versiones (como Git), facilitando la colaboración y seguimiento de cambios.

Desarrollo

Con base en la selección del proyecto a desarrollar móvil o web realice la codificación del módulo del proyecto aplicando alguno de los framework vistos en el componente formativo "Frameworks para construcción de aplicaciones con JAVA.".

Elementos para tener en cuenta:

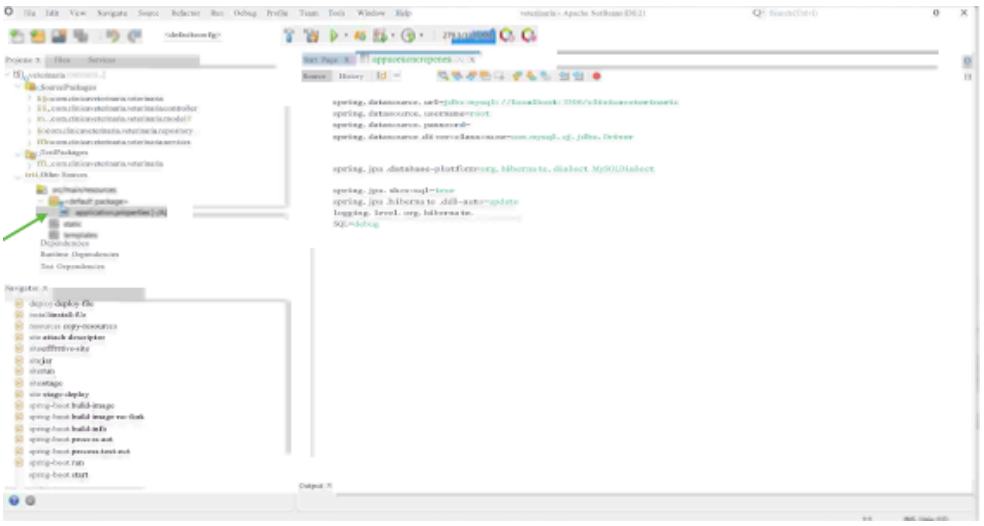
● Para la codificación del módulo debe tener en cuenta los artefactos del ciclo del software realizados con anterioridad: diagrama de clases, diagramas de casos de uso, historias de usuario, diseños, prototipos, Informe técnico de plan de trabajo para construcción de software con tecnologías seleccionadas etc.

● Se debe crear el proyecto utilizando herramientas de versionamiento. ● El código debe contener: formularios HTML con servlets
● Utilizar métodos get y pos

● Utilizar elementos de JS

1. Archivo de Propiedades de la Aplicación

El primer paso en el desarrollo del módulo es establecer la conexión con la base de datos en el archivo de propiedades de la aplicación. En este archivo se especifican las credenciales de acceso y la URL de la base de datos:



propiedades — Copiar código

```
spring.datasource.url=jdbc:mysql://localhost:3306/clinicaveterinaria
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
logging.level.org.hibernate.SQL=debug
```

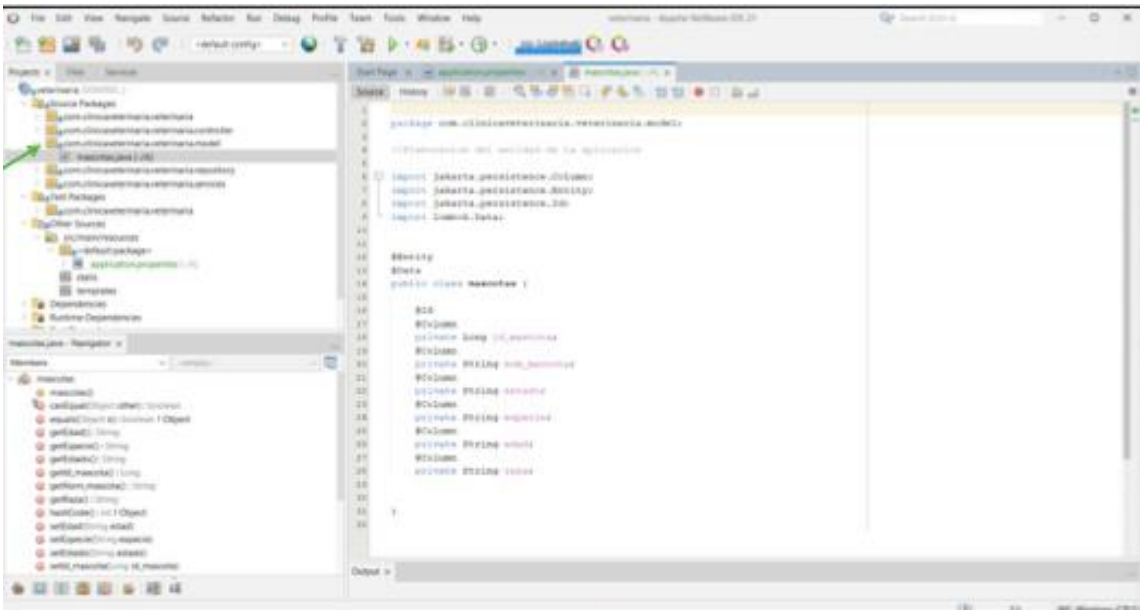
2. Estructuración del Proyecto

- **Creación de paquetes Java:**
  - **Modelo:** Contiene las clases de entidad de la aplicación.
  - **Repositorio:** Definir interfaces para la interacción con la base de datos.
  - **Servicios:** Definir la lógica de negocio y los métodos CRUD.
  - **Controlador:** Maneja las peticiones HTTP y utiliza los servicios para realizar las operaciones necesarias.

CREAMOS LOS 4 PAQUETES JAVA: modelo, repositorio, servicios, controlador

Desarrollo del CRUD

- **Modelo:** //clase para la elaboración de la entidad de la aplicación

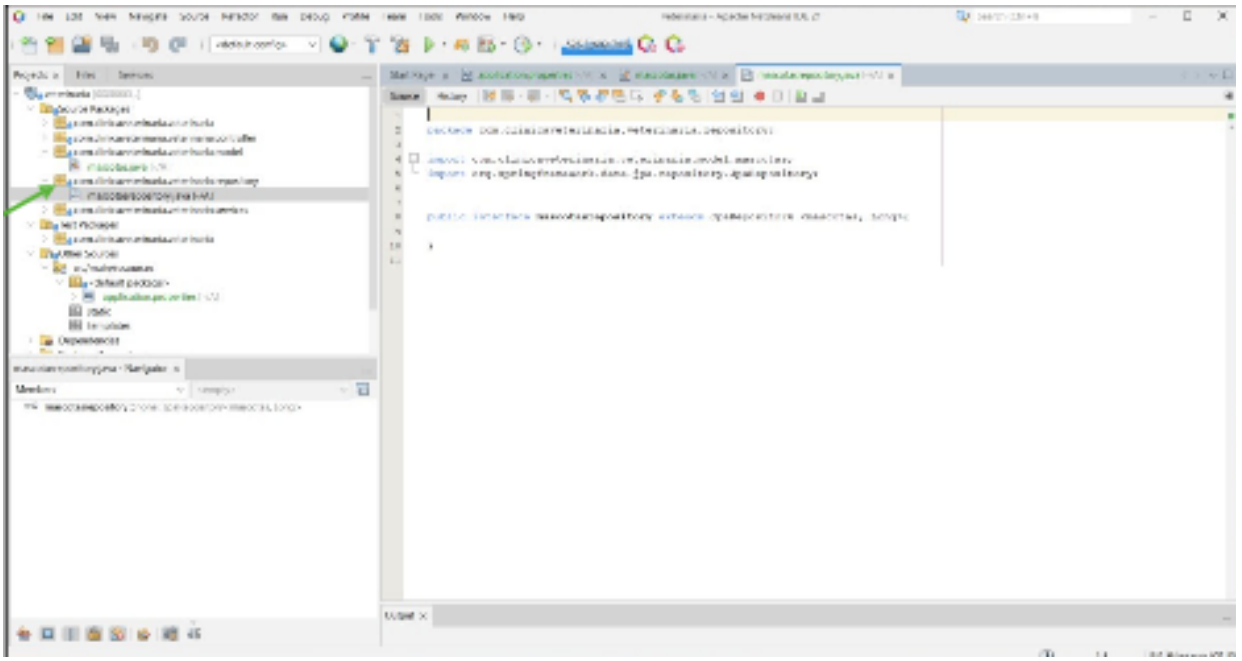


- **Modelo:**

Java — Copiar código

```java
@Entity
public class Mascota {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private String especie;
    private String raza;

    // Getters y setters
}
```

- **PAQUETE REPOSITORIO:** interfaz del repositorio, se llamar unas conexiones con la clase que se generó de mascota

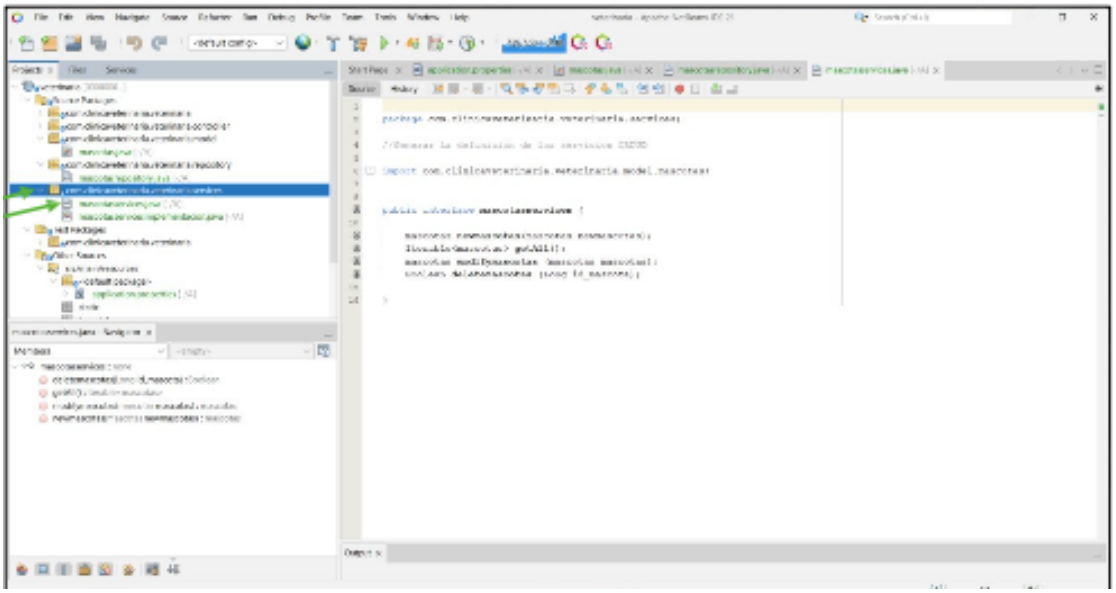


- **Repositorio:**

Java — Copiar código

```java
public interface MascotaRepository extends JpaRepository<Mascota, Long> {
}
```

4. **PAQUETE SERVICIOS**
   5.1. **Interfaz de servicio;** //Generar la definición de los servicios CRDUD



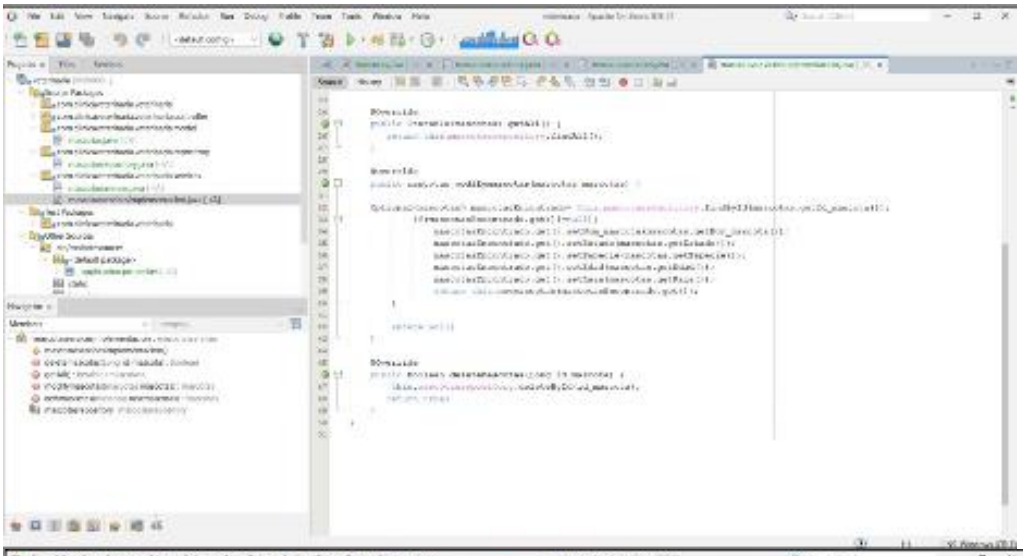
- **Servicios:**

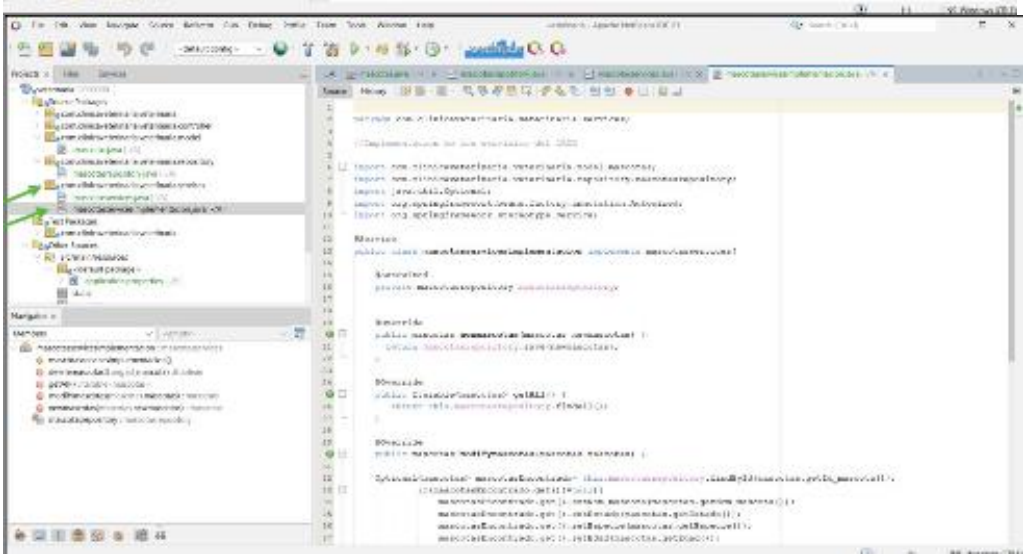
Java — Copiar código

```java
public interface MascotaService {
    List<Mascota> findAll();
    Mascota findById(Long id);
    Mascota save(Mascota mascota);
    void deleteById(Long id);
}

@Service
public class MascotaServiceImpl implements MascotaService {
    @Autowired
    private MascotaRepository mascotaRepository;

    @Override
    public List<Mascota> findAll() {
        return mascotaRepository.findAll();
    }
```

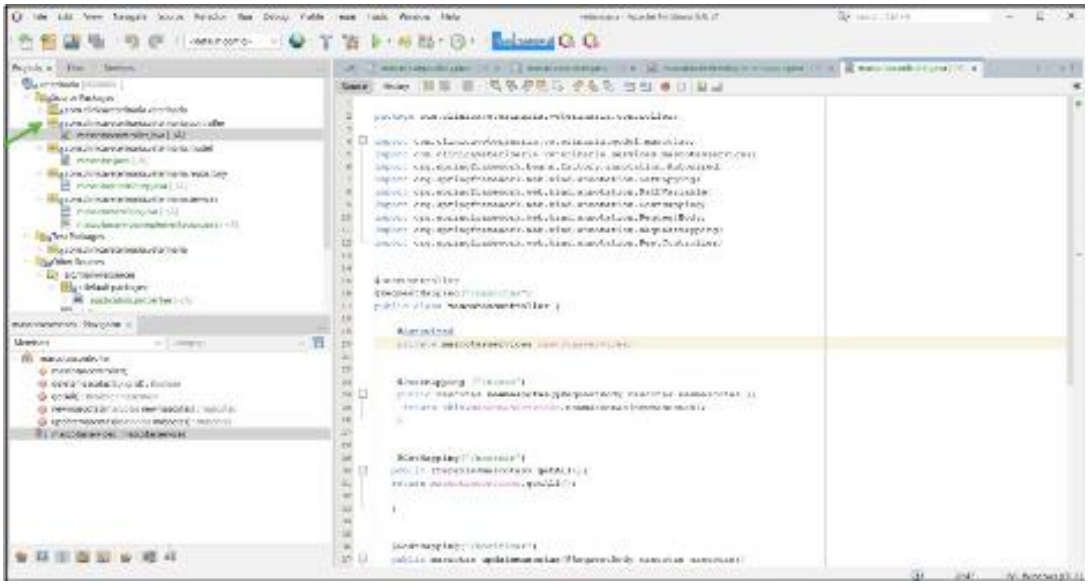
```java
    @Override
    public List<Mascota> findAll() {
        return mascotaRepository.findAll();
    }

    @Override
    public Mascota findById(Long id) {
        return mascotaRepository.findById(id).orElse(null);
    }

    @Override
    public Mascota save(Mascota mascota) {
        return mascotaRepository.save(mascota);
    }

    @Override
    public void deleteById(Long id) {
        mascotaRepository.deleteById(id);
    }
}
```

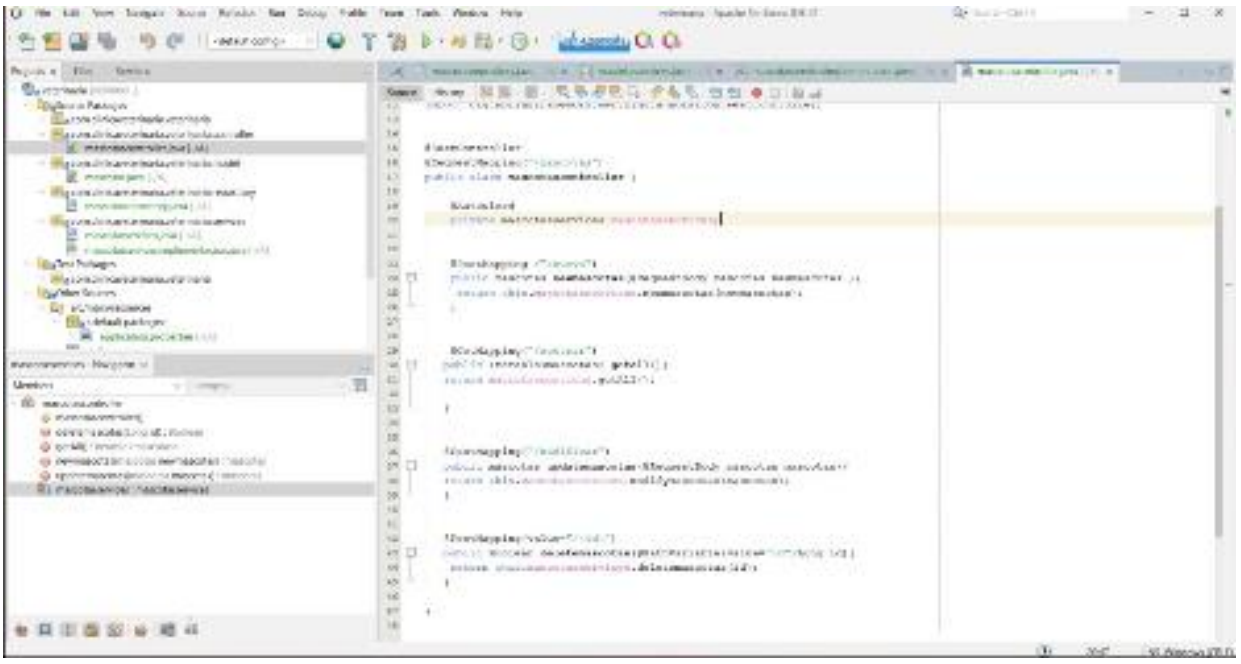
5.2. **Clase servicios:** implementación crud, definir cada uno de los elementos del crud





6. **PAQUETE CONTROLADOR:** clase denominada mascotas, utiliza para llamar los métodos





- **Controlador:**

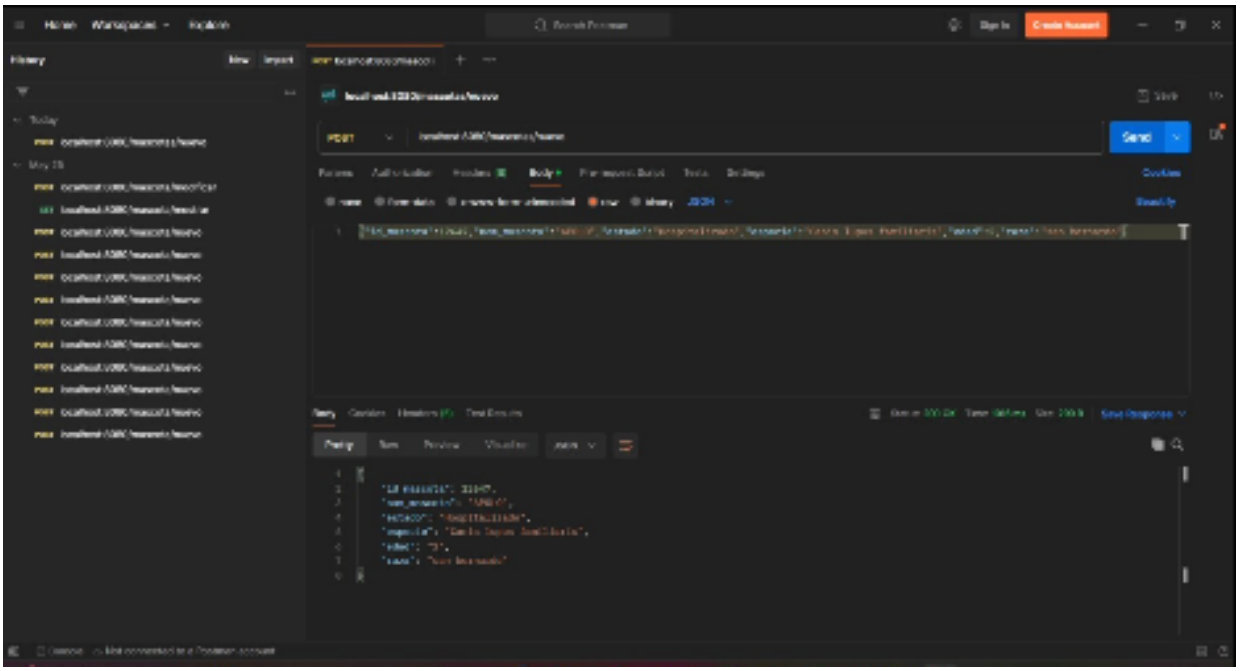
Java — Copiar código

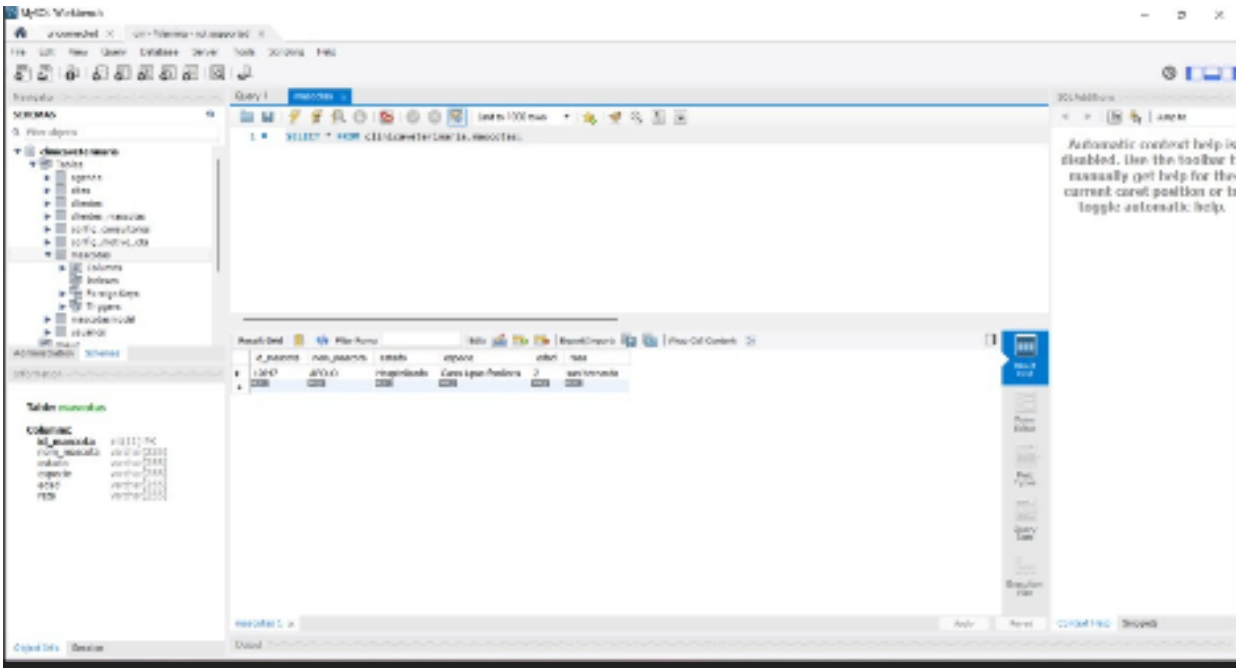
```java
@RestController
@RequestMapping("/mascotas")
public class MascotaController {
    @Autowired
    private MascotaService mascotaService;

    @GetMapping
    public List<Mascota> getAllMascotas() {
        return mascotaService.findAll();
    }

    @GetMapping("/{id}")
    public Mascota getMascotaById(@PathVariable Long id) {
        return mascotaService.findById(id);
    }

    @PostMapping
    public Mascota createMascota(@RequestBody Mascota mascota) {
        return mascotaService.save(mascota);
```

```
    }

    @PutMapping("/{id}")
    public Mascota updateMascota(@PathVariable Long id, @RequestBody Mascota ma
        Mascota existingMascota = mascotaService.findById(id);
        if (existingMascota != null) {
            mascota.setId(id);
            return mascotaService.save(mascota);
        }
        return null;
    }

    @DeleteMapping("/{id}")
    public void deleteMascota(@PathVariable Long id) {
        mascotaService.deleteById(id);
    }
}
```

7. UTILIZAMOS POSMAN PARA HACER LOS GET Y LOS POST

7.1. PARA CREAR UN REGISTRO



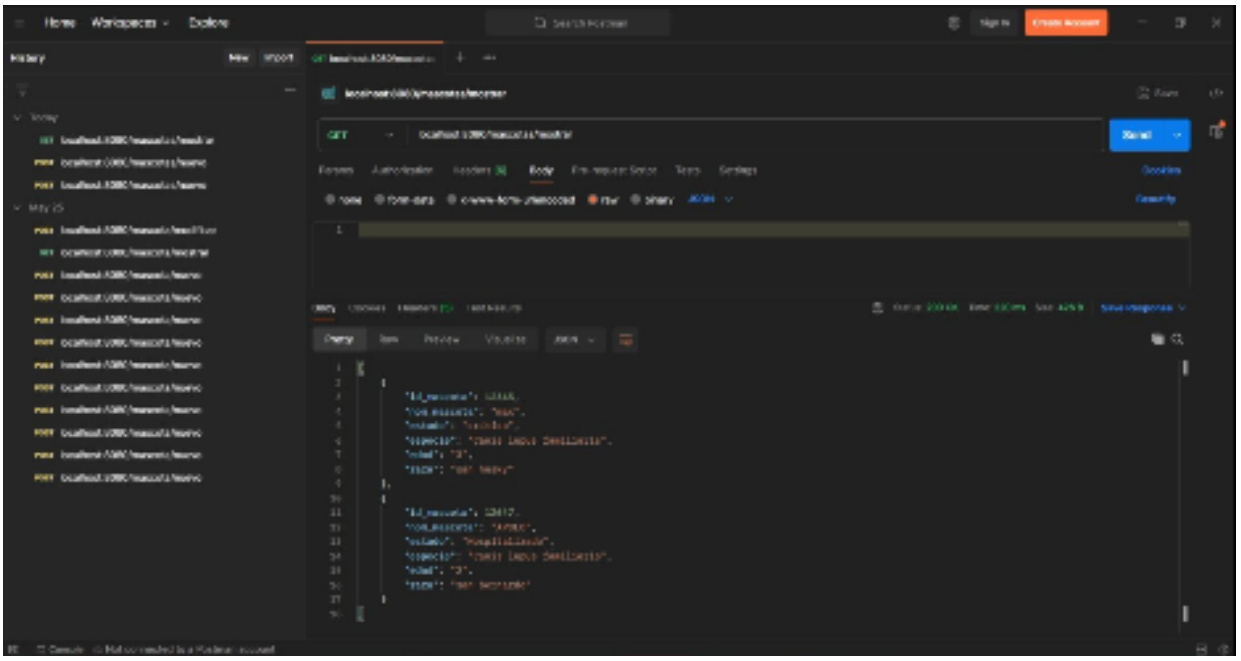


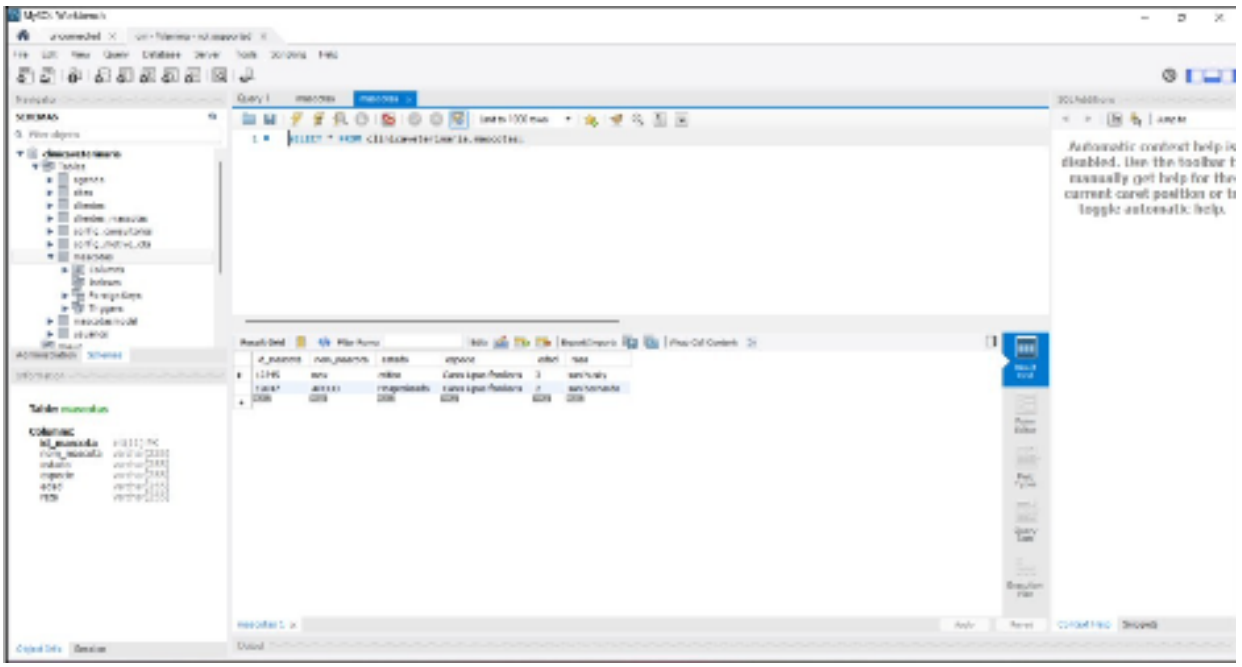
- Crear un registro:

json Copiar código

```
POST /mascotas
{
    "nombre": "Firulais",
    "especie": "Perro",
    "raza": "Labrador"
}
```

7.2 PARA MOSTRAR TODOS LOS REGISTROS



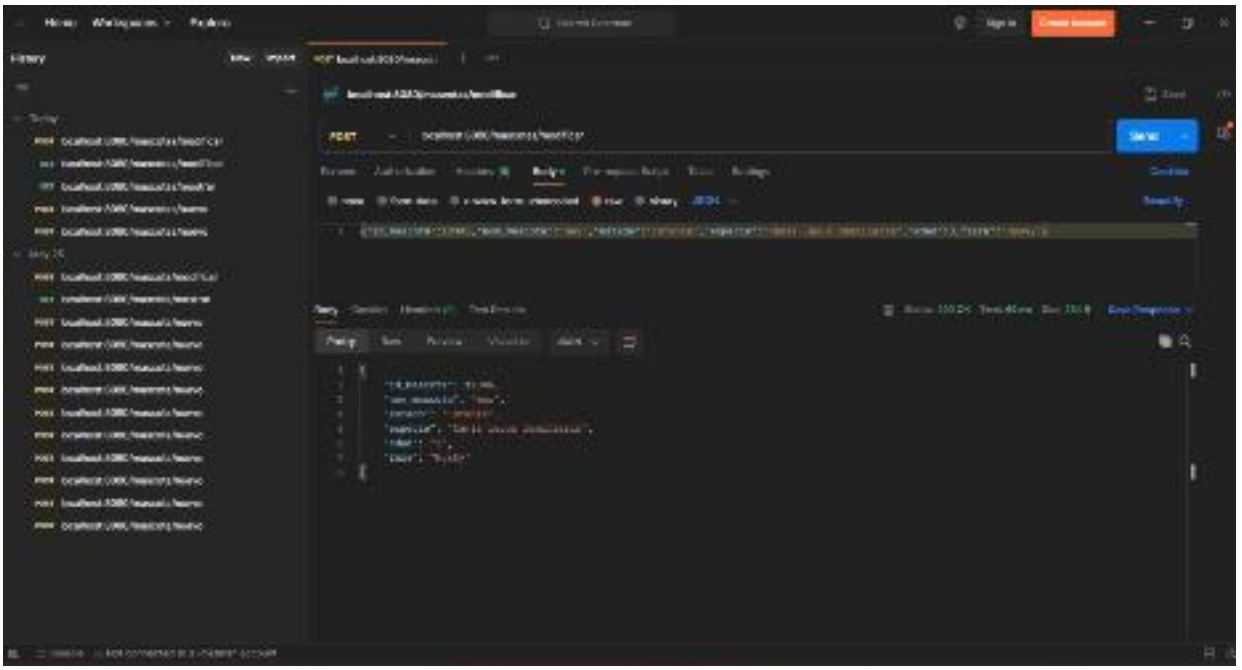


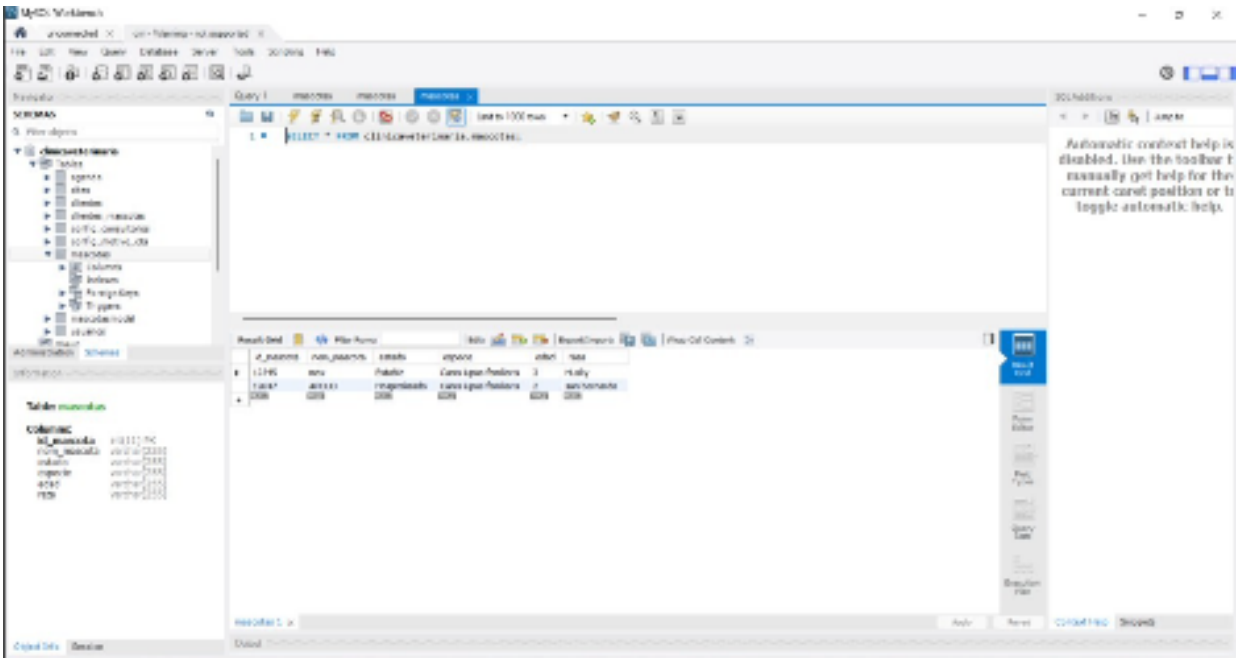
- Mostrar todos los registros:

json Copiar código

```
GET /mascotas
```

7.3. MODIFICAR UN REGISTRO





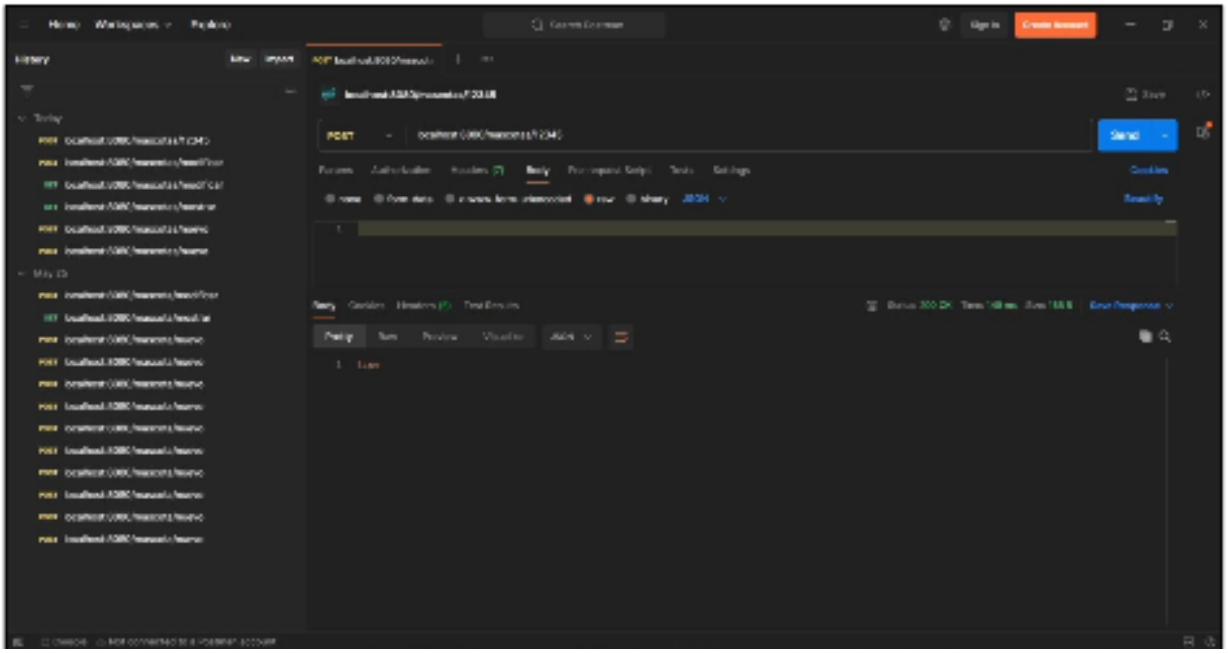
- Modificar un Registro:

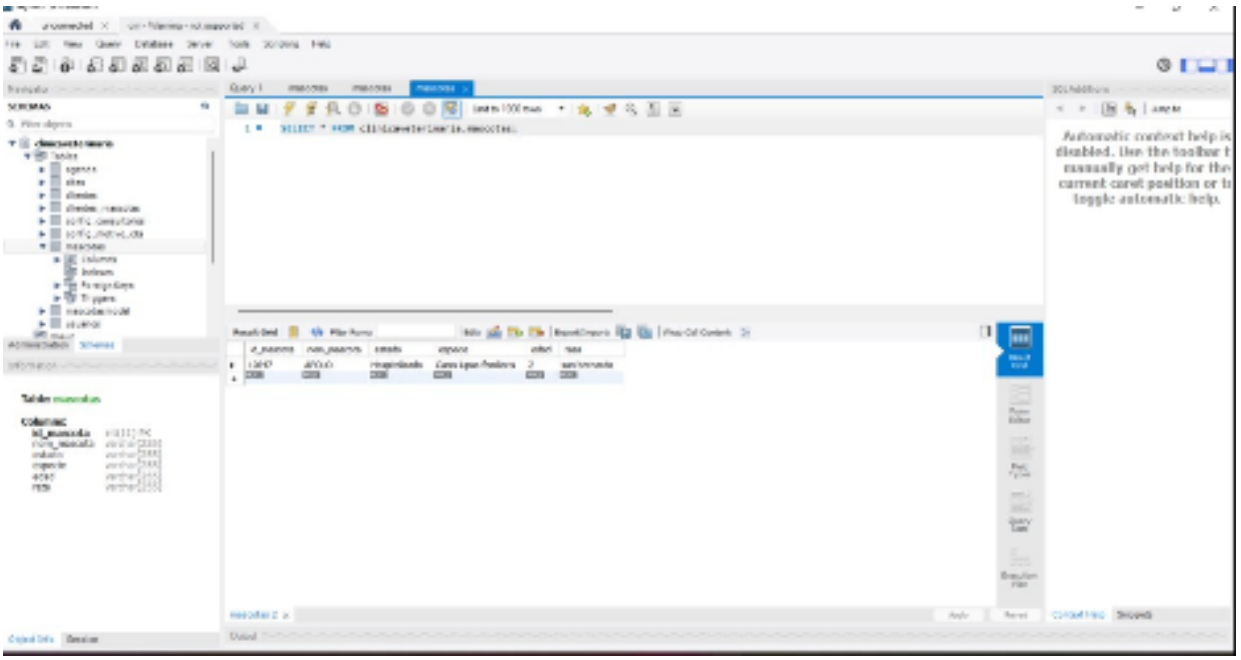
json

Copiar código

```
PUT /mascotas/1
{
    "nombre": "Fido",
    "especie": "Perro",
    "raza": "Beagle"
}
```

7.4. PARA ELIMINAR UN REGISTRO





- **Eliminar un registro:**

```json
DELETE /mascotas/1
```

Conclusión

El desarrollo de este módulo de la aplicación de gestión de mascotas utilizando Spring Boot, NetBeans y MySQL Workbench ha demostrado ser una tarea integral que combina diversas herramientas y prácticas recomendadas en la ingeniería de software. La implementación de un CRUD básico ha permitido reforzar la importancia de seguir un ciclo de vida de desarrollo de software bien estructurado, apoyado en artefactos como diagramas de clases, casos de uso, historias de usuario y prototipos.

Además, el uso de un sistema de control de versiones como Git ha sido crucial para el seguimiento detallado de los cambios y la colaboración efectiva en el proyecto, elementos esenciales en entornos de desarrollo modernos. En resumen, este proyecto ha permitido la consolidación de conocimientos y habilidades fundamentales en el desarrollo de software, garantizando un producto de calidad y alineado con las expectativas del usuario final.