2022

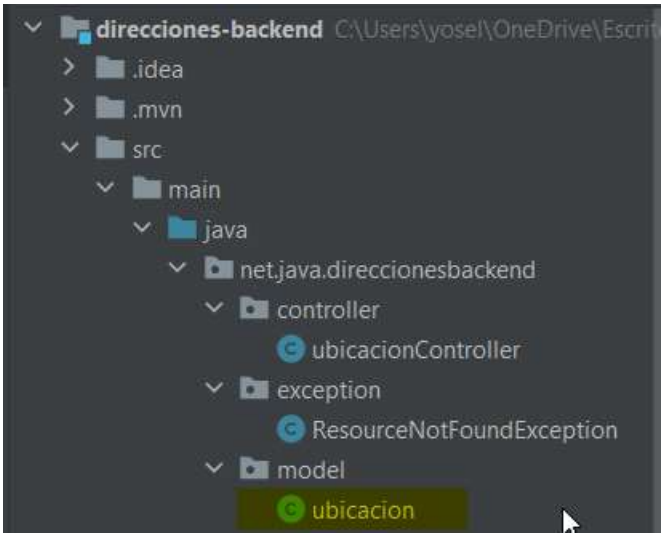
Manual Técnico Backend

Tabla de contenido

Backend

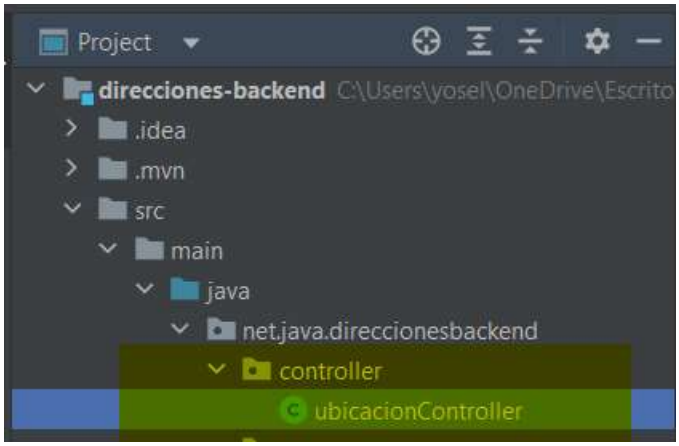
Mapeo de Tablas

En la carpeta MODEL/Ubicación está el mapeo de la tabla DIRECCION



```java
16 usages
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "direccion")

public class ubicacion {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id_direccion;

    @Column(name = "Nombre_dep")
    private String Nombre_dep;

    @Column(name = "Departamento")
    private String Departamento;

    @Column(name ="Municipio")
```

Instancias de servicios web



Métodos Web Service

1. *getAllUbicacion()*

Obtiene todos los registros de la tabla, que se encuentren activos

```
@GetMapping
public List<ubicacion> getAllubicacion(){
    //return UbicacionRepository.findAll();
     List<ubicacion> todas = UbicacionRepository.findAll();
     List<ubicacion> activas = new ArrayList<>();
     for (int i = 0; i< todas.size();i++){
         if(todas.get(i).getActivo()==1){
             activas.add(todas.get(i));
         }
     }
     return activas;
}
```

2. *createUbicacion(@RequestBody ubicacion Ubicacion)*

Se encarga de crear un registro nuevo en la tabla, recibiendo como parámetro un objeto de tipo Json para el insert del mismo.

```
/crear api rest de ubicacion
   @PostMapping
   public ubicacion createUbicacion(@RequestBody ubicacion Ubicacion){
       Ubicacion.setActivo(1);
       return UbicacionRepository.save(Ubicacion);
   }
```

3. *getUbicacionById(@PathVariable long id)*

Se encarga de obtener, por medio de un parámetro (id), el registro que corresponda en la tabla al id que recibe como argumento.

```
@GetMapping("{id}")
public ResponseEntity<ubicacion> getUbicacionById(@PathVariable long id){
ubicacion Ubicacion =UbicacionRepository.findById(id)
        .orElseThrow(()-> new ResourceNotFoundException("No existe la dependencia con el id:"+ id));

return ResponseEntity.ok(Ubicacion);
}
```

4. *updateUbicacion(@PathVariable long id,@RequestBody ubicacion ubicacionDetails)*

Este método, se encarga de actualizar un registro específico en la tabla, con respecto al Json recibido en el segundo parámetro, tomando los valores como nuevos del parámetro obtenido.

```java
@PutMapping("{id}")
public ResponseEntity<ubicacion> updateUbicacion(@PathVariable long id,@RequestBody ubicacion ubicacionDetails ){
    ubicacion updateUbicacion = UbicacionRepository.findById(id).
            orElseThrow(()-> new ResourceNotFoundException("No existe la dependencia con el id:"+ id));
    updateUbicacion.setNombre_dep(ubicacionDetails.getNombre_dep());
    updateUbicacion.setDepartamento(ubicacionDetails.getDepartamento());
    updateUbicacion.setMunicipio(ubicacionDetails.getMunicipio());
    updateUbicacion.setDireccion(ubicacionDetails.getDireccion());
    updateUbicacion.setTelefono(ubicacionDetails.getTelefono());
    updateUbicacion.setDescripcion(ubicacionDetails.getDescripcion());

    UbicacionRepository.save(updateUbicacion);

    return ResponseEntity.ok(updateUbicacion);

}
```

5. *deleteUbicacion(@PathVariable long id)*

Este método, a pesar que su nombre lleva la palabra “delete”, se encarga de colocar en estatus inactivo el registro con id recibido como argumento en el mismo, por lo que internamente no se elimina físicamente ningún registro.

```java
@DeleteMapping("{id}")
public ResponseEntity<ubicacion> deleteUbicacion(@PathVariable long id ){
    ubicacion deleteUbicacion = UbicacionRepository.findById(id).
            orElseThrow(()-> new ResourceNotFoundException("No existe la dependencia con el id:"+ id));
    deleteUbicacion.setActivo(0);


    UbicacionRepository.save(deleteUbicacion);

    return ResponseEntity.ok(deleteUbicacion);

}
```

6. *getUbicacionLike(@PathVariable String nombre)*

Este método se realiza la búsqueda por nombre de Dependencia, verifica en el listado ya existente que está en list<ubicación>.

```java
//realizar busqueda por nombre dependencia

@RequestMapping(path="/search/{nombre}")
public List<ubicacion> getUbicacionLike(@PathVariable String nombre){
    List<ubicacion> todas = UbicacionRepository.findByName(nombre);
    return todas;
}
```

En la carpeta de REPOSITORY/UbicaciónrRepository esta la interface JPArepository en la cual se encuentra el método que hace la consulta con respecto al parámetro que se recibe

```java
public interface ubicacionRepository extends JpaRepository <ubicacion, Long>{
    1 usage
    @Query("SELECT u FROM ubicacion u WHERE u.Nombre_dep LIKE concat('%',?1,'%') and u.Activo = 1")
    List<ubicacion> findByName(String nombreDep);
}
```

Conexión a MySQL server

En la conexión se deben de agregar el host en donde se encuentra corriendo la base de datos, el nombre de la base datos, el usuario y contraseña así como el del auto que controla el comportamiento de la conexión

```
spring.datasource.url=jdbc:mysql://localhost:3306/direccion_fisica?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyD
spring.datasource.username=root
spring.datasource.password=admin

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto = update
```