**Hibernate: proyecto**

Crear un proyecto con spring boot que incorpore: Spring web, PosgreSQL, Spring Data JPA y Spring devtools.

**Crear las siguientes entidades:**

TagColor: una enumeración, podrá ser de tipo BLUE, YELLOW, GREEN, RED.

Tag: representa una entidad etiqueta con los atributos id, nombre, color (enum

TagColor, almacenada como string en base de datos)

User: representa una entidad usuario con los siguientes atributos id, nombre,

apellido, dni, si está activo sí o no, fecha de nacimiento

Task: representa una entidad tarea, con los atributos id, título, descripción,

finalizada (sí o no), fecha de entrega (LocalDate).

BillingInfo: representa la información de facturación de un usuario

**Asociaciones entre entidades**:

Un usuario tiene muchas tareas, una tarea sólo puede pertenecer a un mismo

usuario a la vez.

Una tarea puede tener muchas etiquetas, una etiqueta puede estar en más de

una tarea a la vez.

Un usuario tiene una información de facturación (BillingInfo) y una información

de facturación sólo puede pertenecer a un mismo usuario.

**Desarrollos:**

* Implementar controladores que permitan:

1. Crear entidades de cada tipo (POST) utilizando repositorios
2. Actualizar entidades de cada tipo (PUT) utilizando repositorios
3. Recuperar entidades de cada tipo: recuperar todos, recuperar uno,

recuperar todos filtrando por un atributo utilizando criteria con

EntityManager desde clases DAO.

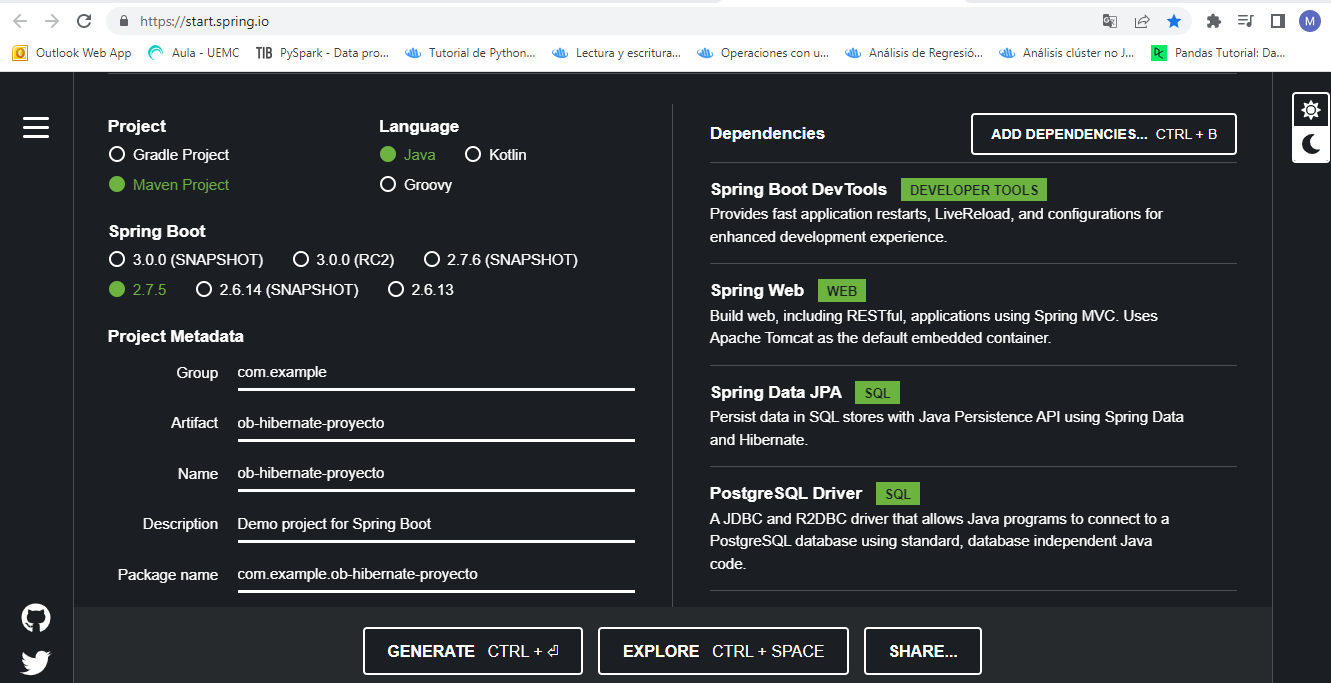
* Los controladores llaman a capa de servicio y el servicio llama a repositorios o

DAOs.

* Pruebas con Postman para cada entidad y exportar la colección postman para

agregarla al proyecto en la carpeta raíz.

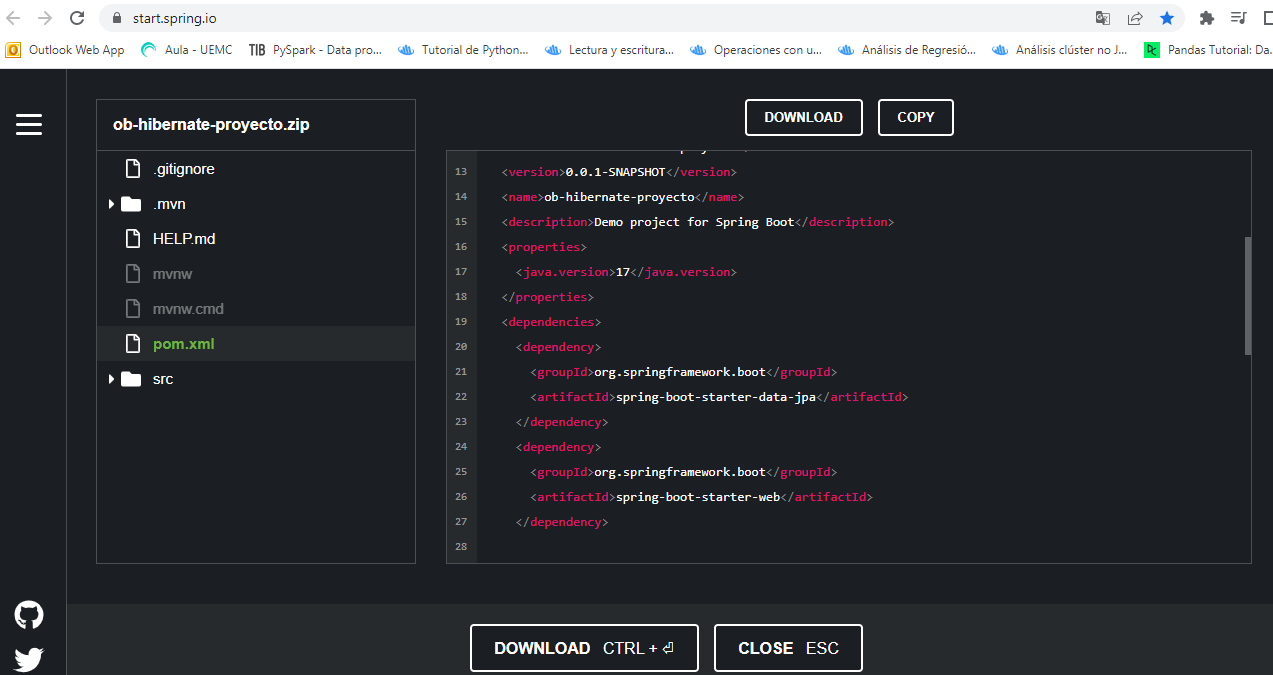
1. Creación del proyecto con Spring Inizializr <https://start.spring.io/>





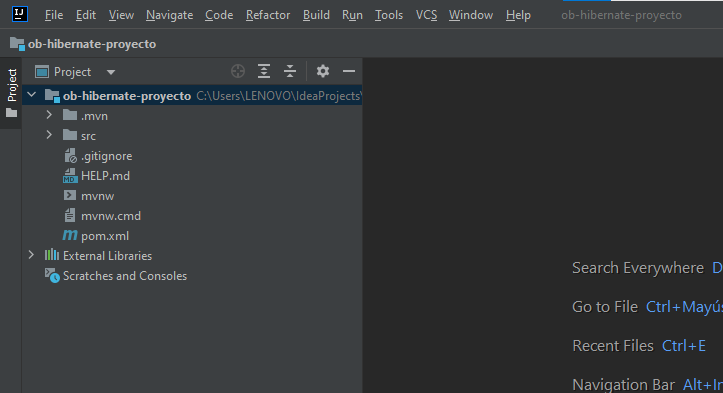


Con esto se observa la estructura del proyecto y archivo de configuración (POM) en el navegador.



Lo descargamos a una carpeta y lo abrimos en intellij

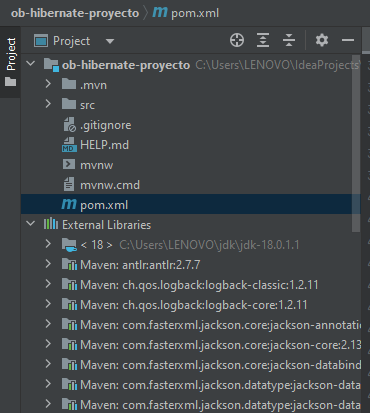




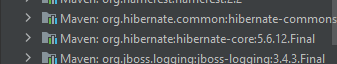
Por otro lado se utilizara Postman para analizar las peticiones (GET - POST – PUT - DELETE) a una url (que es nuestra aplicación con Sprint e hibernate).

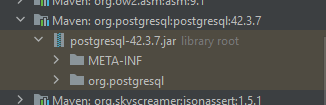
Aquí es donde ejecutamos nuestros métodos (con java) y los vemos.

Ya con el proyecto cargado en intellij en POM se ve el archivo de configuración y en External Libraries vemos las librerías cargadas para el proyecto

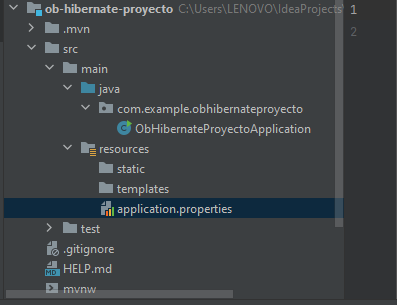


Deberiamos verificar que este cargado hibernate y postgres.



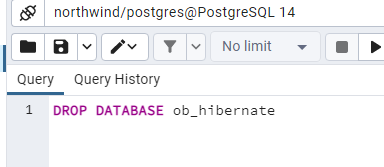


Estructura

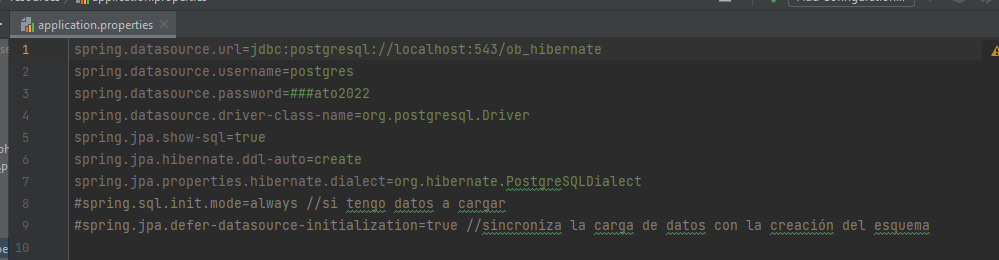


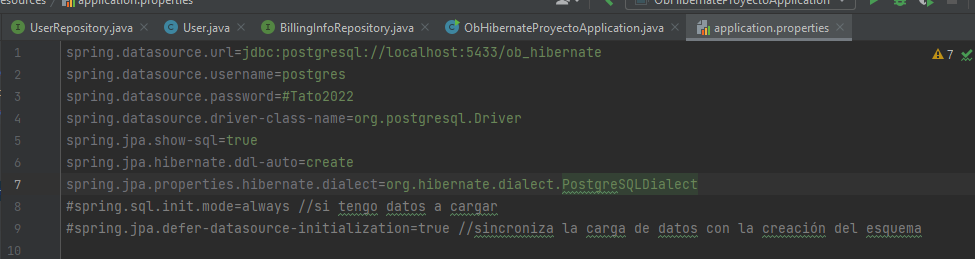
En application.properties vamos a realizar la configuración de la base de datos (Postgres en este caso). (Debemos estar conectados a una BD En este caso ob\_hibernate en postgres).

Si vamos a utilizar una BD existente conviene borrar y volverla a generar. Para borrar conectarse a otra BD y poner



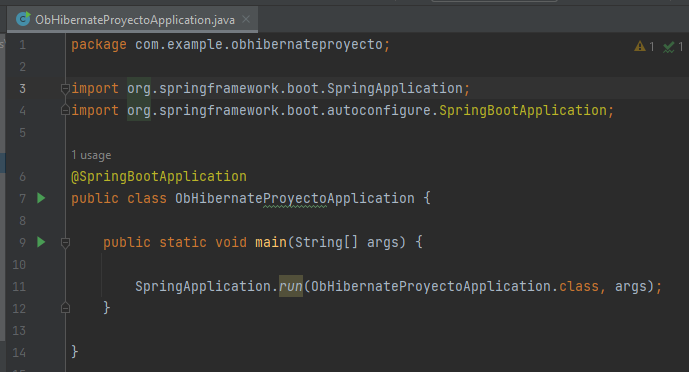
Configuramos la conexión a la BD



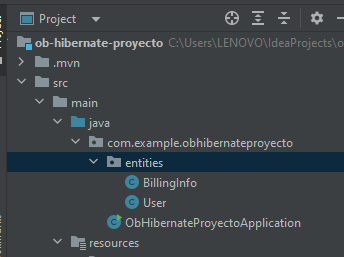


En ObHibernateProyectoApplication esta el **main** que ejecuta la app.

Aquí dentro se generan los paquetes y se crearan las entidades, controladores, servicios, DAOs, repositorios y se iran controlando desde postgres



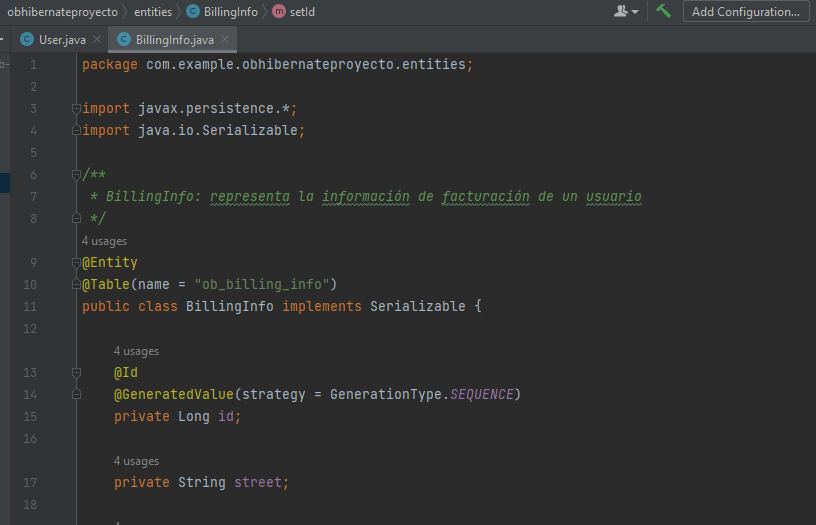
Comenzamos a generar objetos (Package - entities) (Usuarios)(BillingInfo)



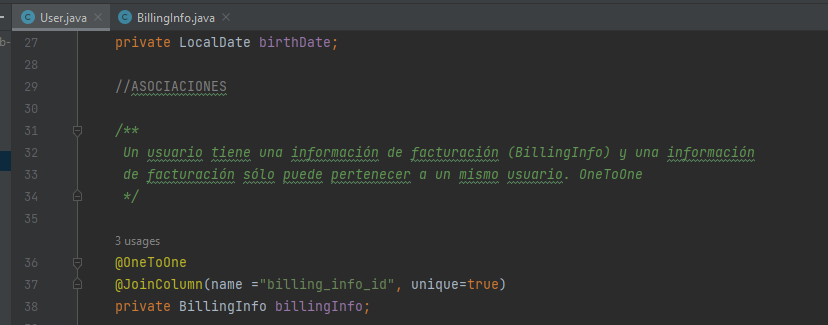
Java class (Usuarios): Definimos los atributos y generamos contructor, getter – setter y To-String



Java class (BillingInfo): Definimos los atributos y generamos contructor, getter – setter y To-String

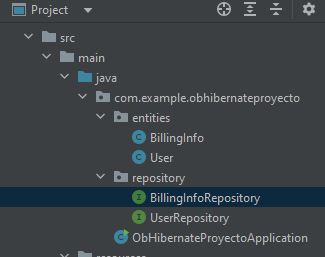


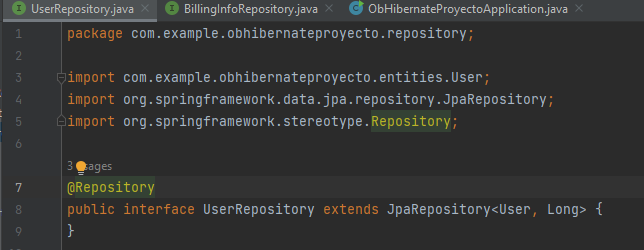
Generamos las Asociaciones

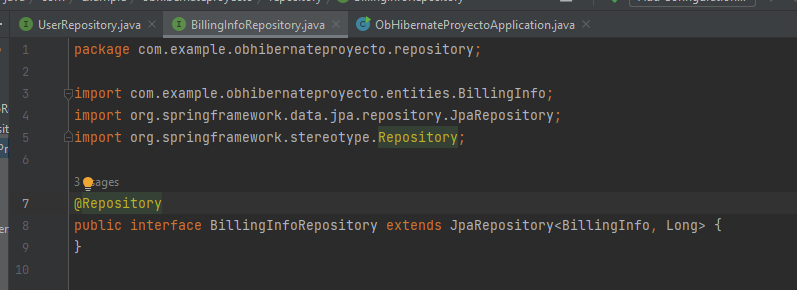




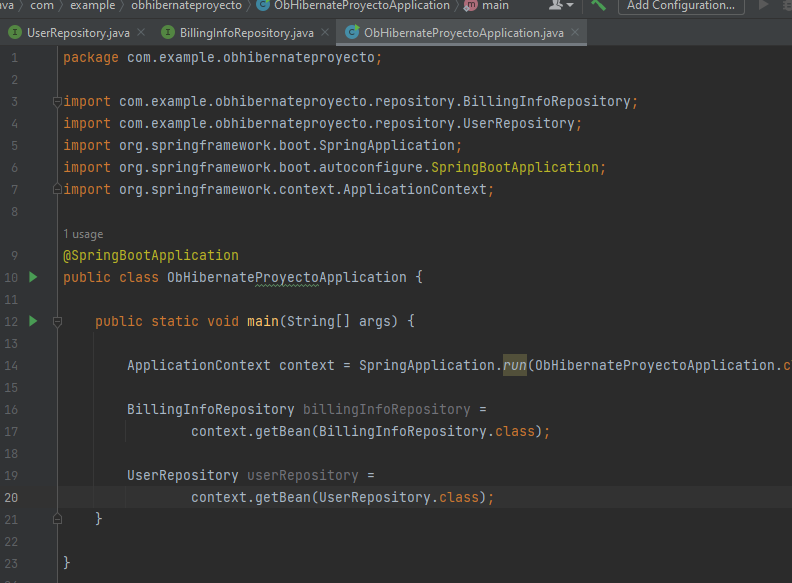
Generamos Repositorios (Interfaces UserRepository - BillingInfoRepository)

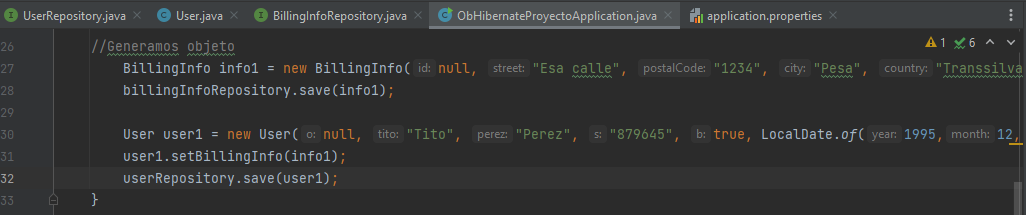




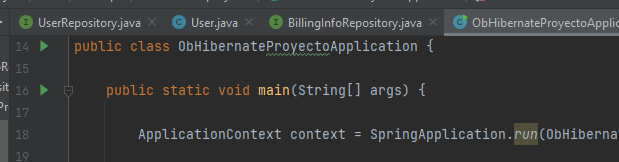


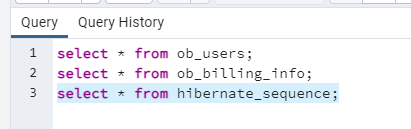
Y en el Main

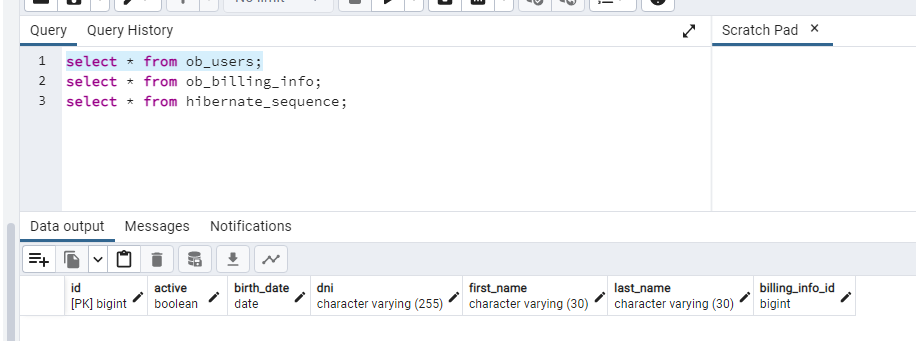


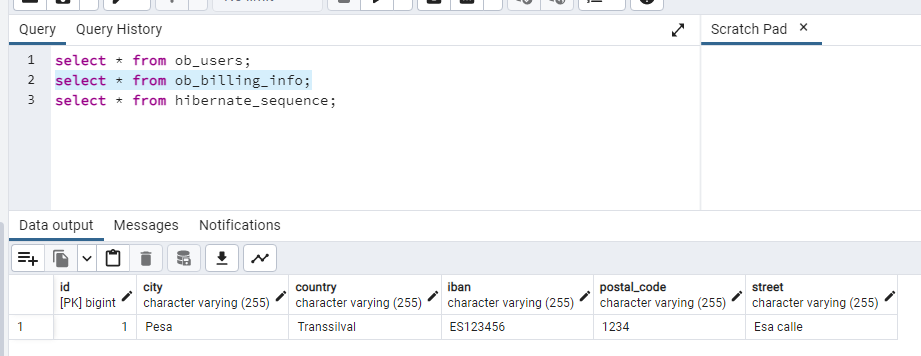


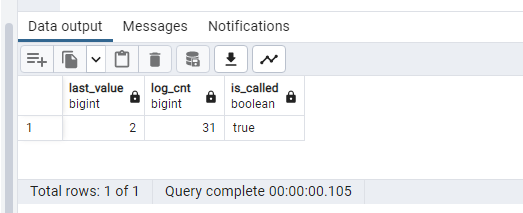
Y probamos….



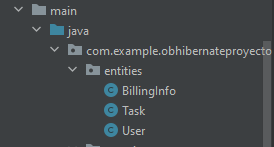


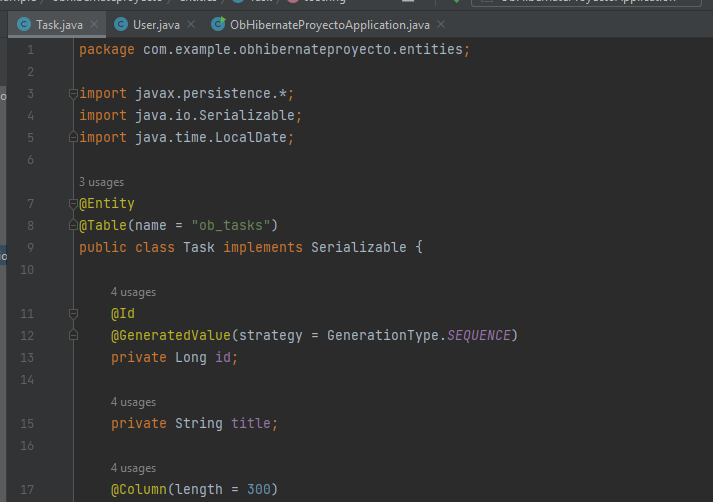






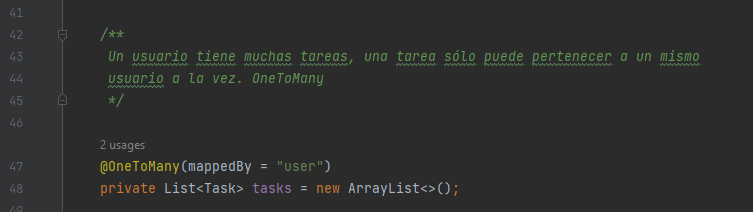
Generamos una nueva class Task y su asociación a User





Asociación

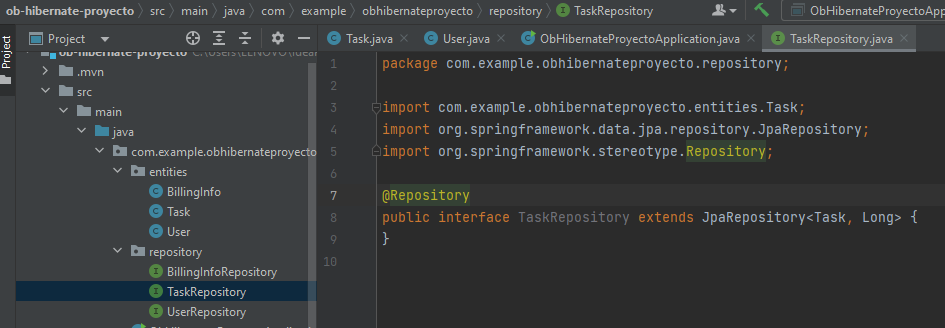
User



Task

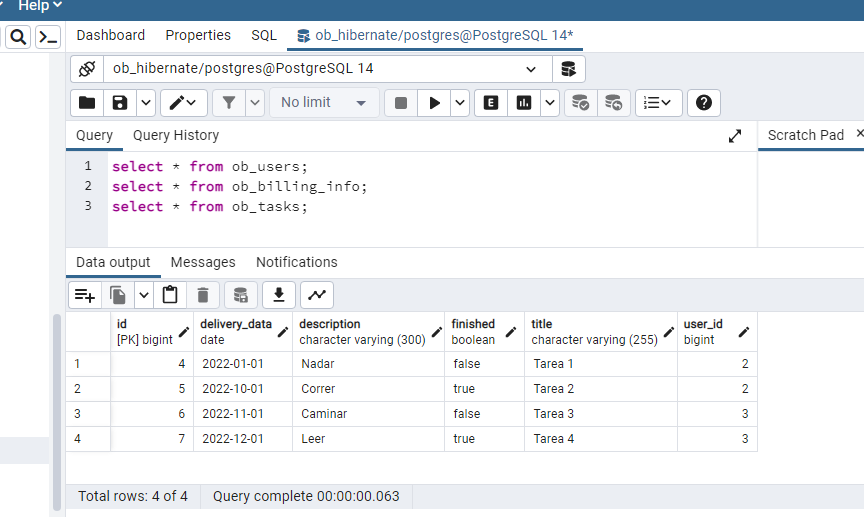


Grneramos el TaskRepository

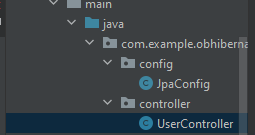


Corremos y vemos Postgre





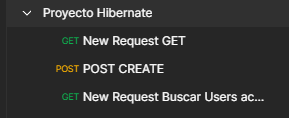
Agragamos una capa de controladores

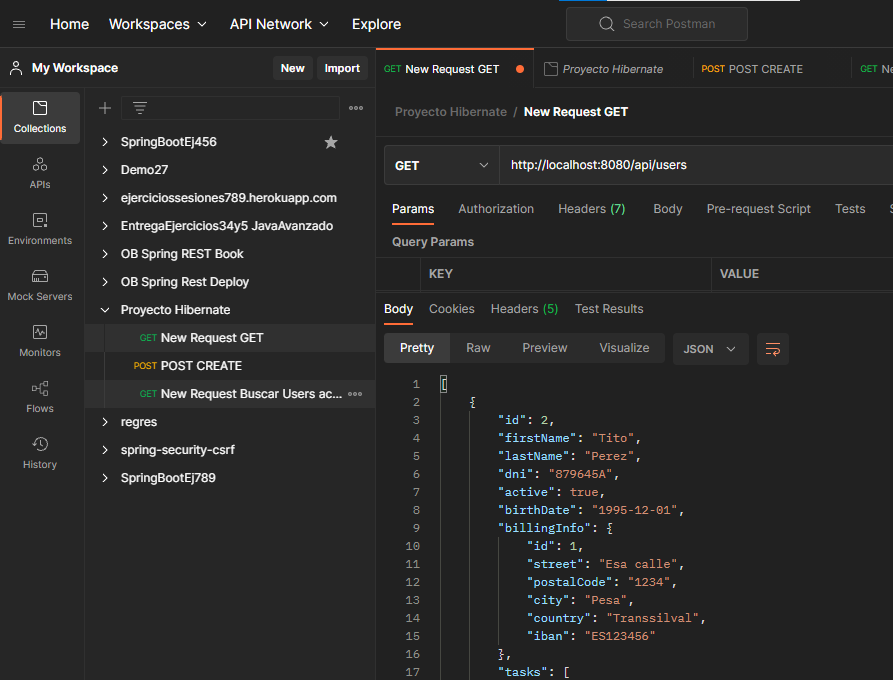


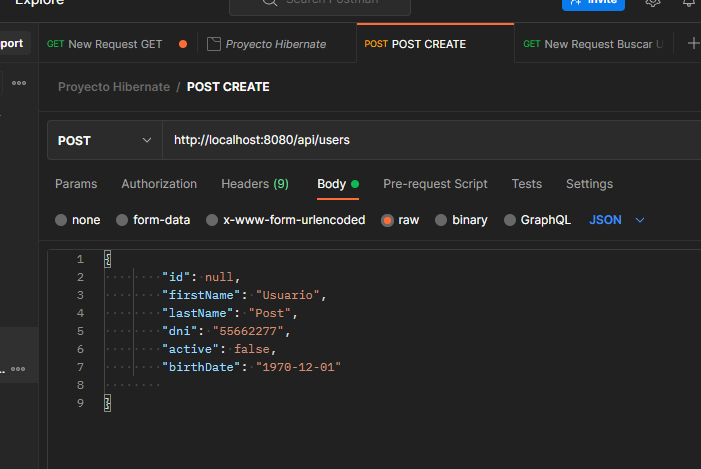


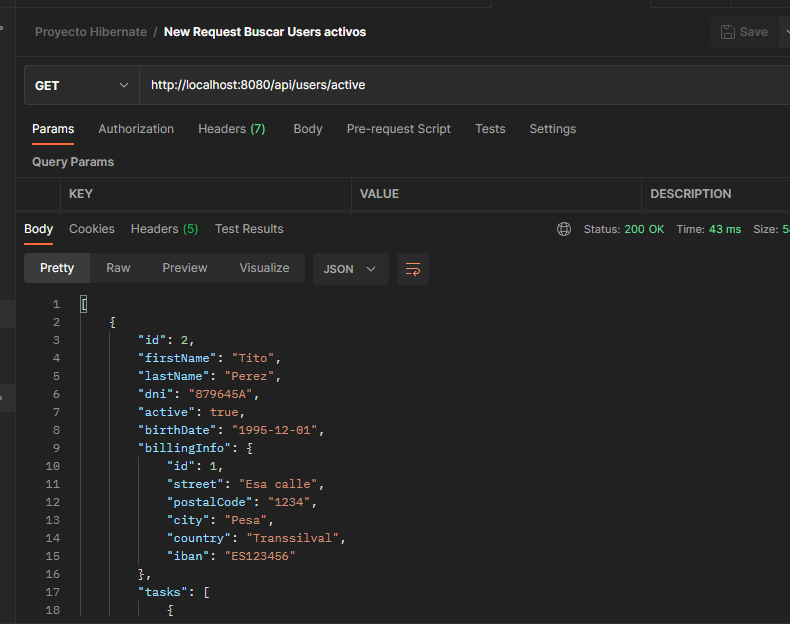
package com.example.obhibernateproyecto.controller;  
  
import com.example.obhibernateproyecto.dao.UserDAO;  
import com.example.obhibernateproyecto.entities.User;  
import com.example.obhibernateproyecto.repository.UserRepository;  
//import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
*/\*\*  
 \* TODO - crear servicio  
 \* Método actualizar  
 \* Pruebas de guardado cuando hay asociaciones  
 \* - la asociación billinginfo se guarda desde aquí  
 \* - la asociación tasks se guarda desde el controlador tasks  
 \*/*@RestController  
public class UserController {  
  
 private UserRepository userRepository;  
  
 private UserDAO userDao;  
  
 public UserController(UserRepository userRepository, UserDAO userDao){  
 this.userRepository = userRepository;  
 this.userDao = userDao;  
  
 }  
  
 @GetMapping("/api/users")  
 private List<User> findAll(){  
 return this.userRepository.findAll();  
 }  
  
 @GetMapping("/api/users/active")  
 private List<User> findAllActive(){  
 return this.userDao.findAllActive();  
 }  
  
 @PostMapping("/api/users")  
 private User create(@RequestBody User user){  
 return this.userRepository.save(user);  
 }  
  
 @PutMapping("/api/users")  
 private User update(@RequestBody User user){  
 return this.userRepository.save(user);  
 }  
  
 @DeleteMapping("/api/user/{id}")  
 private void delete(@PathVariable Long id){  
 if(this.userRepository.existsById(id))  
 this.userRepository.deleteById(id); // *TODO revisar integridad referencial* }

Hacemos las pruebas en postman

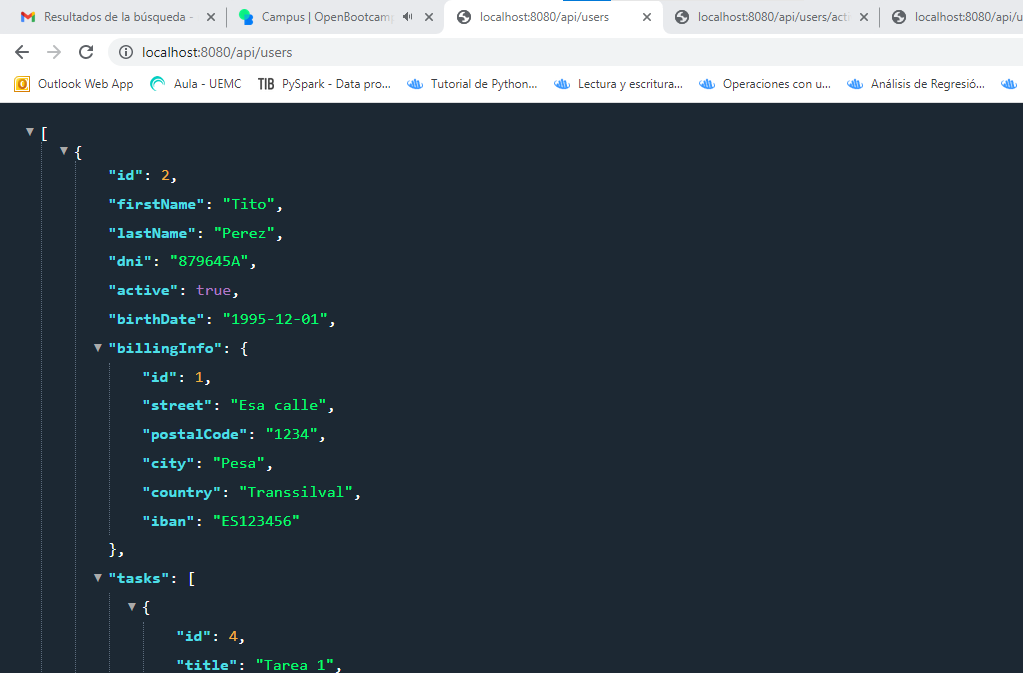








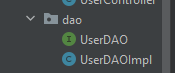
O en el navegador

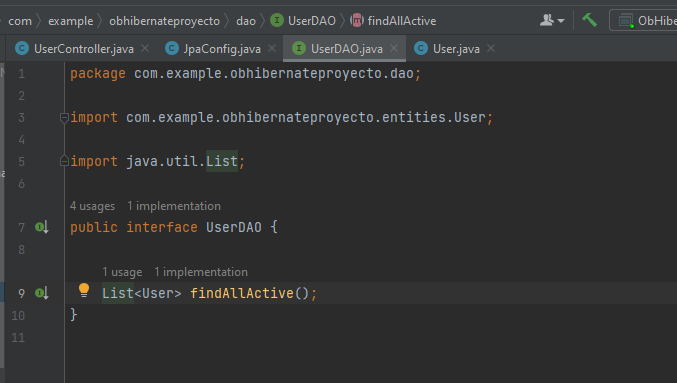


Para evitar los problemas de serializacion de campos (o atributo) con el json



Dao





package com.example.obhibernateproyecto.dao;  
  
import com.example.obhibernateproyecto.entities.User;  
import org.springframework.stereotype.Repository;  
  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Predicate;  
import javax.persistence.criteria.Root;  
import java.util.List;  
  
@Repository  
public class UserDAOImpl implements UserDAO {  
  
 @PersistenceContext  
 private EntityManager entityManager;  
  
 @Override  
 public List<User> findAllActive() {  
  
 CriteriaBuilder builder = entityManager.getCriteriaBuilder();  
 CriteriaQuery<User> criteria = builder.createQuery(User.class);  
 Root<User> root = criteria.from(User.class);  
  
 Predicate isActive = builder.isTrue(root.get("active"));  
  
 criteria.select(root).where(isActive);  
  
 return entityManager.createQuery(criteria).getResultList();  
 }  
}