

# Creación de Api-Rest con Spring-boot

**Asignatura:** Despliegue/HLC  
**Curso:** 2 DAW (20/21)  
**Autor:** Jesús David Gutiérrez Delgado

# Índice

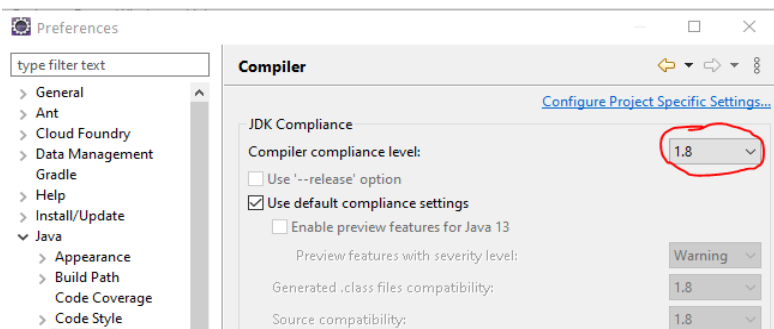
0. Introducción.....	3
1. Generación del proyecto.....	4
2. Desarrollo de objetos y servicios.....	6
2.0 Aplicación.....	6
2.1 Modelo.....	7
2.2 Objeto DAO.....	8
2.3 Servicio.....	9
2.4 Controlador.....	10
3. Persistencia.....	11
4. Puesta en marcha.....	12
5. Pruebas en local.....	13
6. Despliegue en GearHost de la BB.DD.....	13
7. Despliegue de la aplicación en Heroku.....	15
8. Pruebas en el servidor.....	18
9. Documentación.....	18

## 0. Introducción

El presente documento pretende describir el proceso de creación y despliegue de una pequeña API para la realización de un CRUD, que mantendrá una sola tabla. Será desarrollado en Java con el framework Springboot (JPA e Hibernate). Se hará inca pié en los procesos de desarrollo en Java (objetos, dependencias, anotaciones, etc.) así como en el de despliegue, que en este caso se hará en [Heroku](#) para el back-end y [GearHost](#) para la BB.DD.

# 1. Generación del proyecto

Comenzaremos por generar la estructura básica de nuestro proyecto para poder iniciar el desarrollo en el IDE de nuestra elección que, para el caso que nos ocupa, será Eclipse. Comprobaremos previamente cuál es la versión de Java con la que estamos trabajando en Eclipse, ya que necesitaremos saberla para los pasos siguientes. En nuestro caso es la versión 8.



Para la generación del proyecto entraremos en la página de [Springboot](#) y desde ella iniciaremos la configuración básica del proyecto antes de su generación. Señalar sólo que todos estos pasos se podrían haber realizado de forma manual desde el IDE, pero Spring nos los automatiza desde esta web que vamos a ver, siendo el resultado final un proyecto ya estructurado que sólo tendremos que importar desde el IDE.



**Project**  
☒ Maven Project ☐ Gradle Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 2.5.0 (SNAPSHOT) ☐ 2.5.0 (M2) ☐ 2.4.4 (SNAPSHOT) ☒ 2.4.3  
☐ 2.3.10 (SNAPSHOT) ☐ 2.3.9

**Project Metadata**  

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 15 ☐ 11 ☒ 8

**Dependencies**  

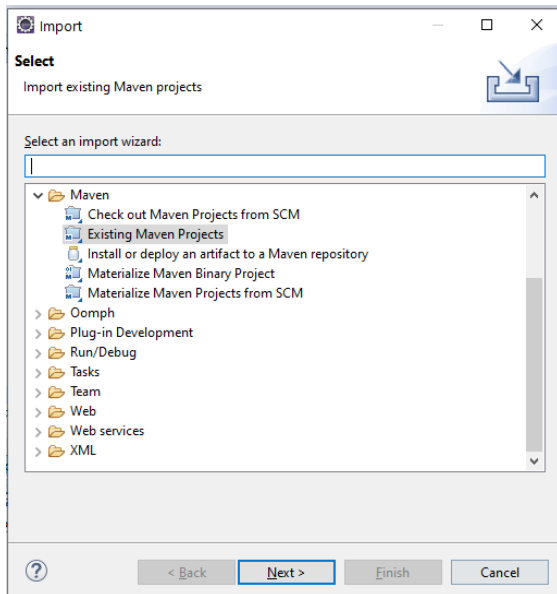
Spring Web ☒ WEB

Build web, including RESTful, applications using Spring MV default embedded container.

La configuración del proyecto es bastante intuitiva, pero señalaremos algunos puntos importantes que no se nos deben pasar por alto:

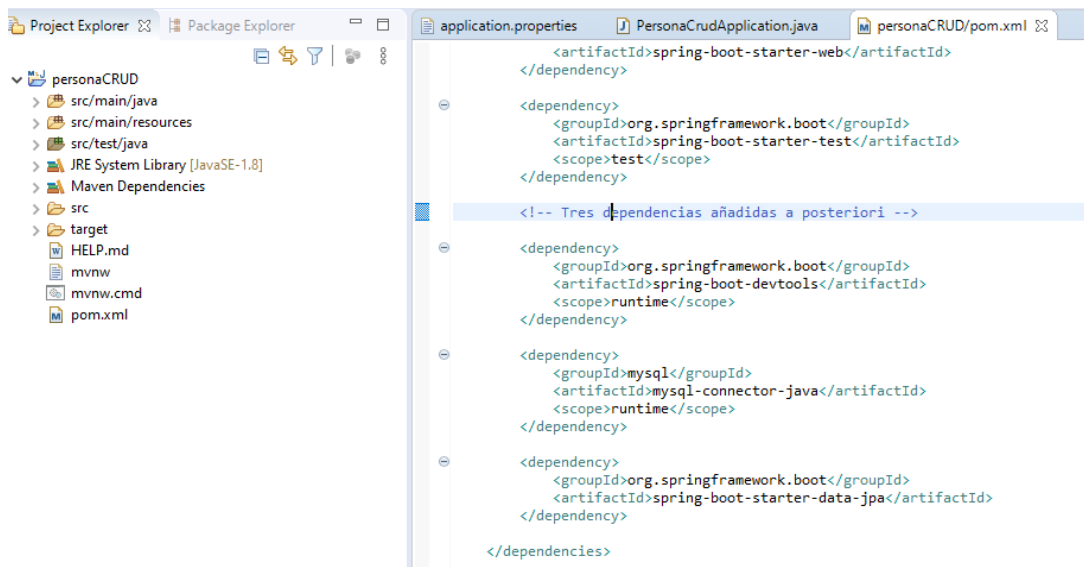
- Las versiones de Java deben coincidir, es por ello que hemos marcado la 8 ya que nuestro IDE trabaja con la 8.
- Debemos empaquetar con un JAR, de otra forma tendremos problemas a la hora del despliegue ya que se trata de una API back-end.
- Añadiremos, por el momento, la dependencia “Spring web”

Una vez hecho esto, generaremos el proyecto y lo descomprimiremos en la carpeta donde vayamos a trabajar. Este proyecto vamos a importarlo a Eclipse como proyecto maven de la siguiente forma:



Desde Eclipse, en el menú “File > Import” seleccionaremos la opción que nos aparece en la ventana de la izquierda. Continuamos con “Next” y seleccionaremos la carpeta donde está nuestro proyecto descomprimido.

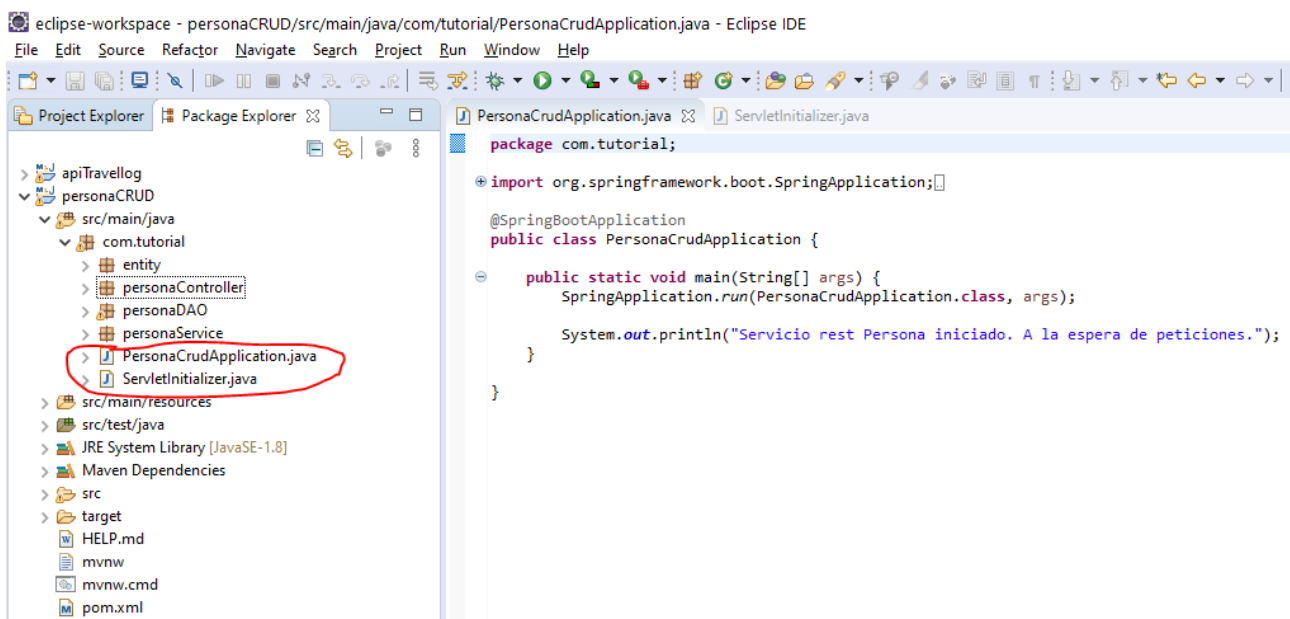
El resultado final debería quedar tal y como aparece en la imagen de más abajo. Se ha añadido tres dependencias más a fin de poder trabajar con JPA (Java Persistence API), MySQL y que nuestro servicio rest se refresque cada vez que hagamos un cambio y no tengamos que parar y arrancar el servidor, siendo esto último muy útil a fin de agilizar las pruebas.



## 2. Desarrollo de objetos y servicios

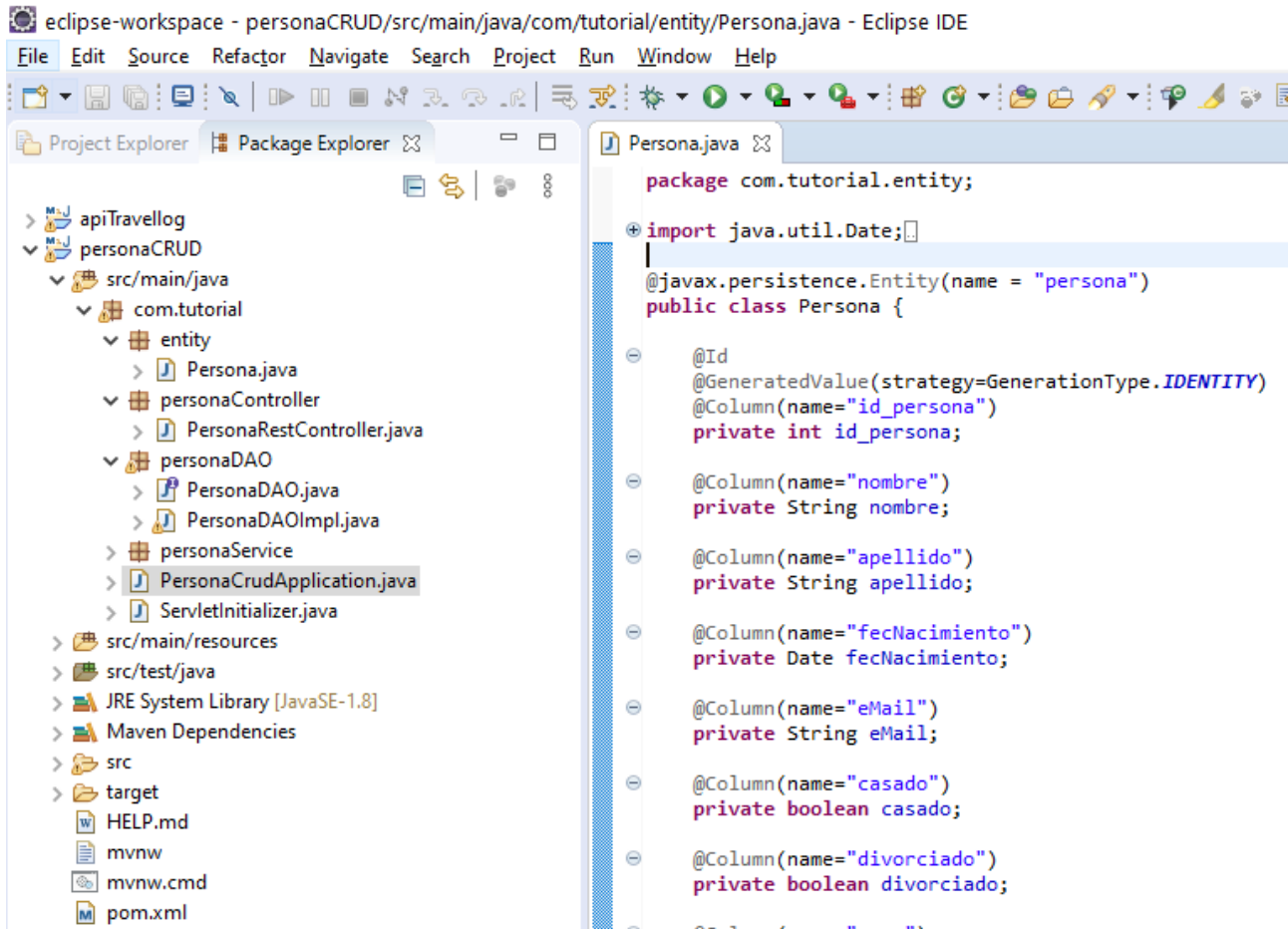
### 2.0 Aplicación

Nuestra aplicación está basada en los dos objetos que aparecen marcados (aplicación y servlet). Es **muy importante** que no estén contenidos en ningún paquete, de lo contrario no serán capaces de comunicarse correctamente con el resto de objetos y, aún arrancando la aplicación, no será capaz de recibir las peticiones que se hagan a los puntos de entrada.



## 2.1 Modelo

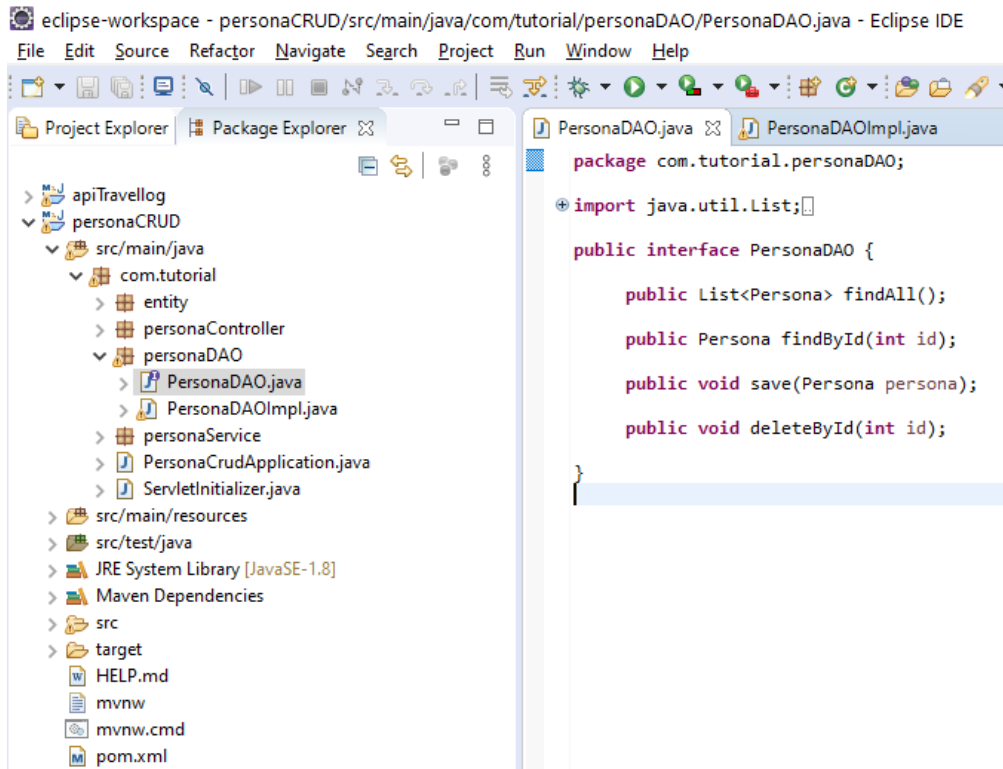
Nuestro servicio REST va a realizar el mantenimiento de una sola tabla/objeto llamado Persona. Las características de este objeto serán las que aparecen a continuación:



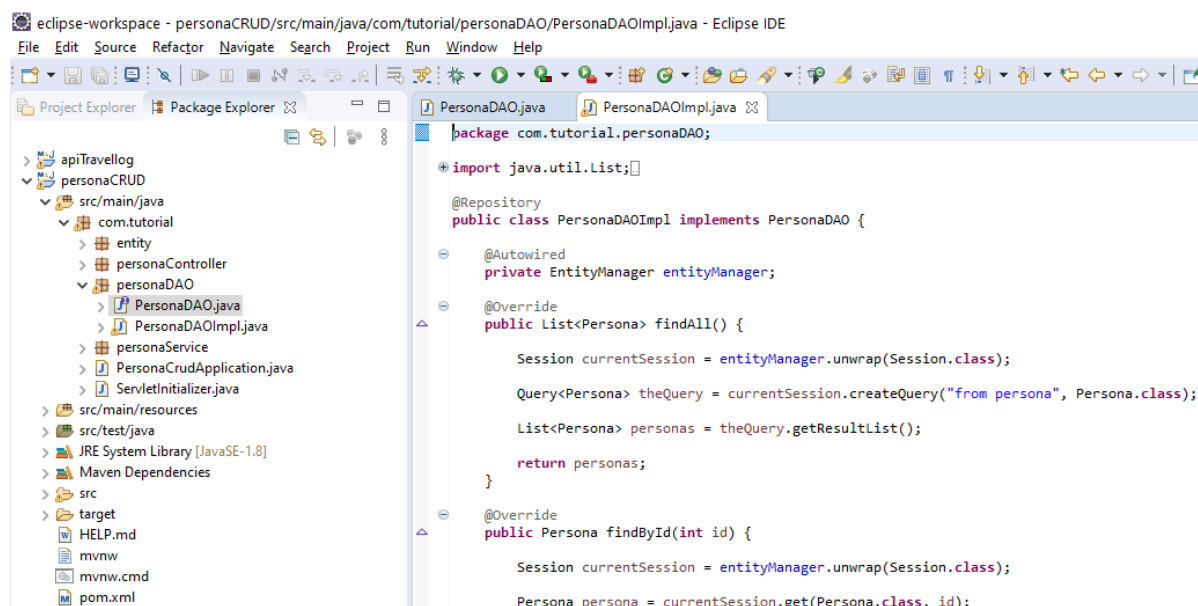
Las anotaciones que aparecen serán para implementar la funcionalidad que JPA nos ofrece a la hora de gestionar nuestra BB.DD. utilizando entidades o clases java.

## 2.2 Objeto DAO

Previo a la construcción de nuestro objeto de acceso a datos, realizaremos la interfaz que lo implementará. Tendrá una funcionalidad básica que podremos ir ampliando a lo largo de nuestro desarrollo, tal y como veremos más adelante.



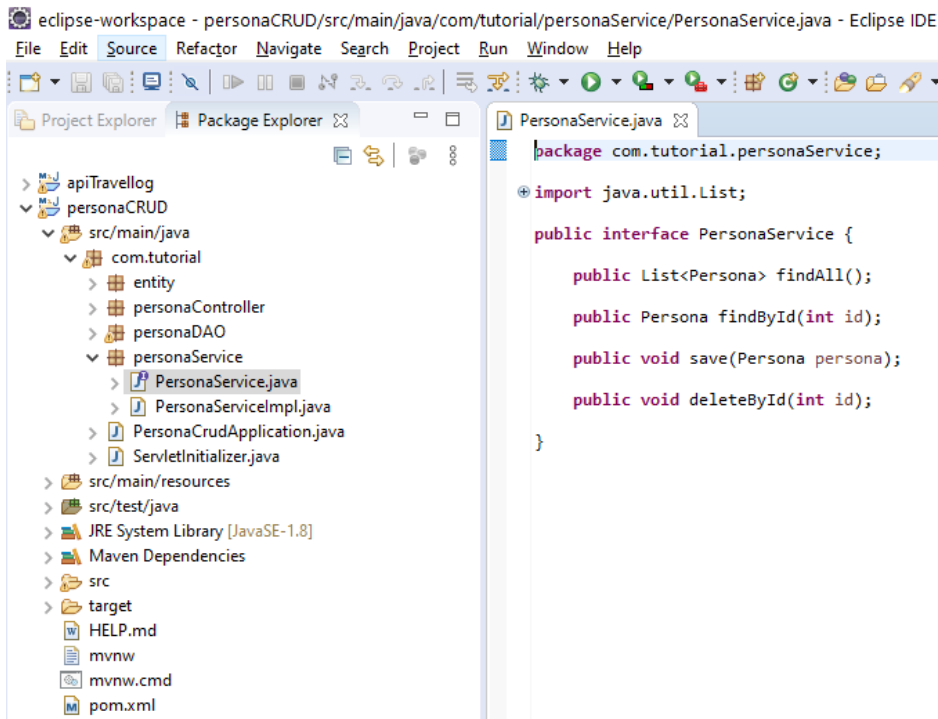
Su implementación será realizada a través de las clases que nos ofrece JPA e Hibernate para la gestión de las acciones que se realizan sobre el modelo (objetos/BB.DD.). En el caso del PUT y DELETE es **indispensable** el utilizar la anotación `@Transactional`, que también podremos usar si queremos en el resto de métodos.



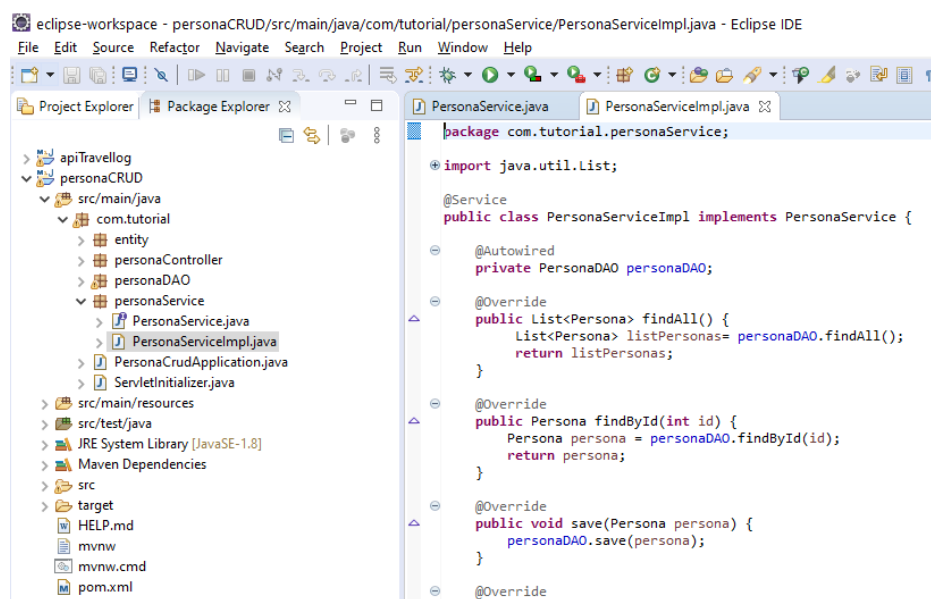


## 2.3 Servicio

Para conectar nuestro objeto DAO con el controlador utilizaremos el servicio como objeto de conexión entre ambos. Este gestionará las peticiones que lleguen a la API y hará las llamadas necesarias al objeto DAO. Al igual que en el DAO, crearemos en primer lugar la interfaz.

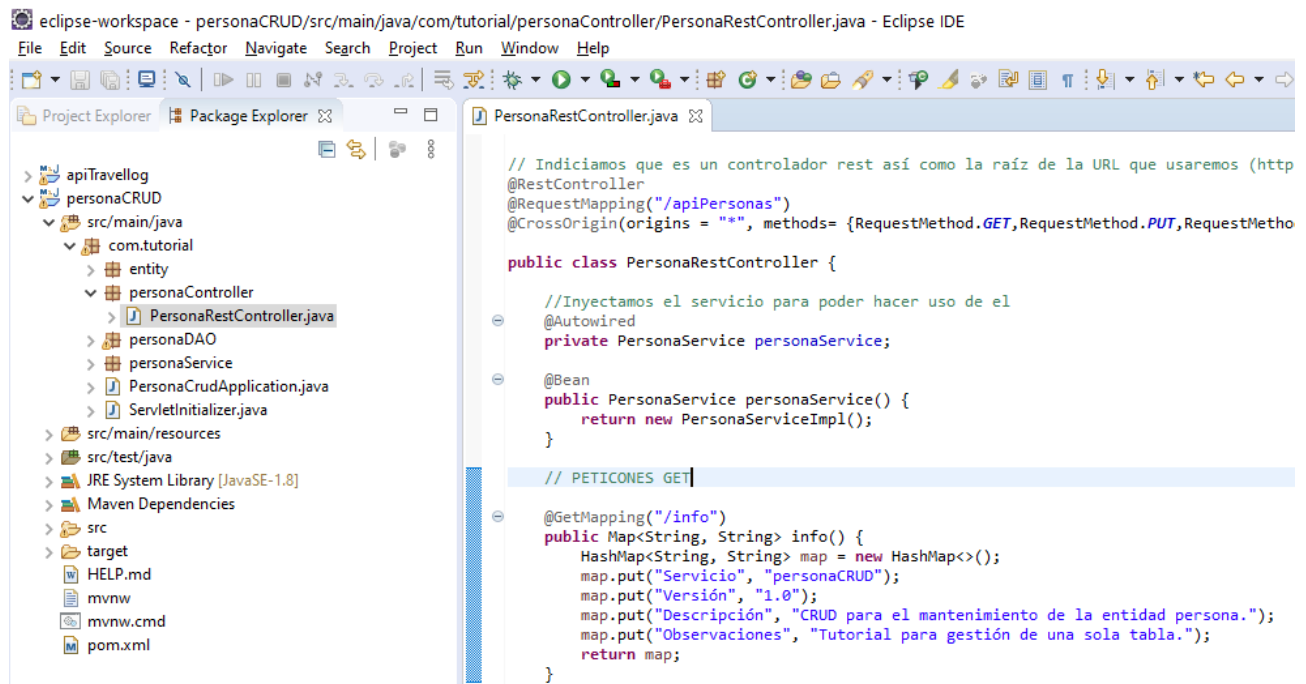


Esta interfaz será la que implementemos en nuestro servicio. Añadiremos las anotaciones correspondientes para indicar que es el servicio (@Service) e inyectaremos el objeto DAO (@Autowired) y así conseguir el desacople.



## 2.4 Controlador

El controlador rest (restController) será el encargado de gestionar y dirigir las peticiones que lleguen a nuestra API. Vamos a implementar los puntos de entrada habituales correspondientes a los métodos más usuales (GET, POST, PUT y DELETE). Podremos ampliar la funcionalidad añadiendo nuevos puntos de entrada que respondan a nuestras necesidades.



```
eclipse-workspace - personaCRUD/src/main/java/com/tutorial/personaController/PersonaRestController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer Package Explorer PersonaRestController.java

> apiTravellog
> personaCRUD
  > src/main/java
    > com.tutorial
      > entity
        > personaController
          > PersonaRestController.java
        > personaDAO
        > personaService
        > PersonaCrudApplication.java
        > ServletInitializer.java
    > src/main/resources
    > src/test/java
    > JRE System Library [JavaSE-1.8]
    > Maven Dependencies
    > src
    > target
    > HELP.md
    > mvnw
    > mvnw.cmd
    > pom.xml

// Indicamos que es un controlador rest así como la raíz de la URL que usaremos (http)
@RestController
@RequestMapping("/apiPersonas")
@CrossOrigin(origins = "*", methods= {RequestMethod.GET, RequestMethod.PUT, RequestMethod.DELETE})

public class PersonaRestController {

    //Inyectamos el servicio para poder hacer uso de el
    @Autowired
    private PersonaService personaService;

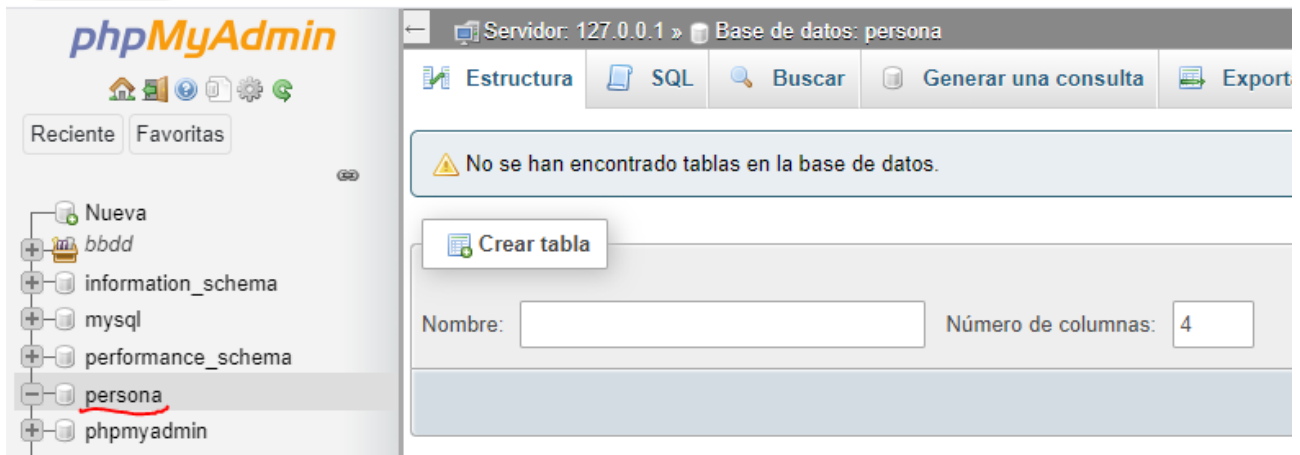
    @Bean
    public PersonaService personaService() {
        return new PersonaServiceImpl();
    }

    // PETICIONES GET

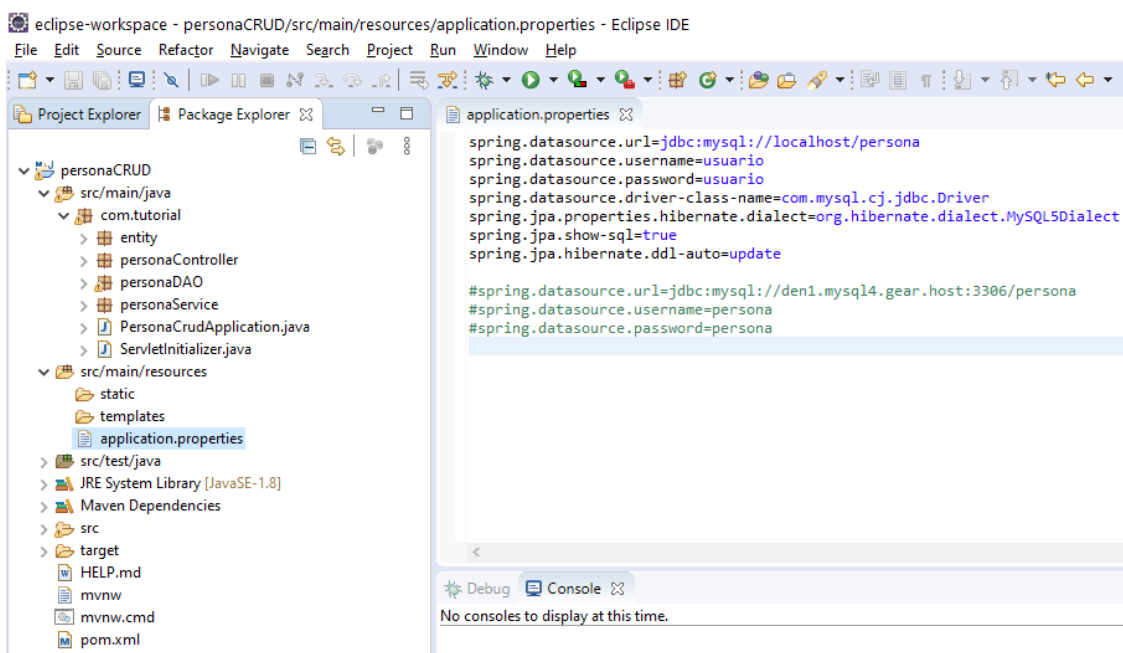
    @GetMapping("/info")
    public Map<String, String> info() {
        HashMap<String, String> map = new HashMap<>();
        map.put("Servicio", "personaCRUD");
        map.put("Versión", "1.0");
        map.put("Descripción", "CRUD para el mantenimiento de la entidad persona.");
        map.put("Observaciones", "Tutorial para gestión de una sola tabla.");
        return map;
    }
}
```

### 3. Persistencia.

En MySql, o aquel SGBD que deseemos utilizar, crearemos nuestra base de datos vacía (sin ninguna tabla) y la llamaremos "persona".



Para que nuestro servicio pueda acceder a la BB.DD. es necesario informar el fichero "application.properties" a fin de que Spring sea capaz de controlar el flujo de información hacia la BB.DD., en este caso MySql.

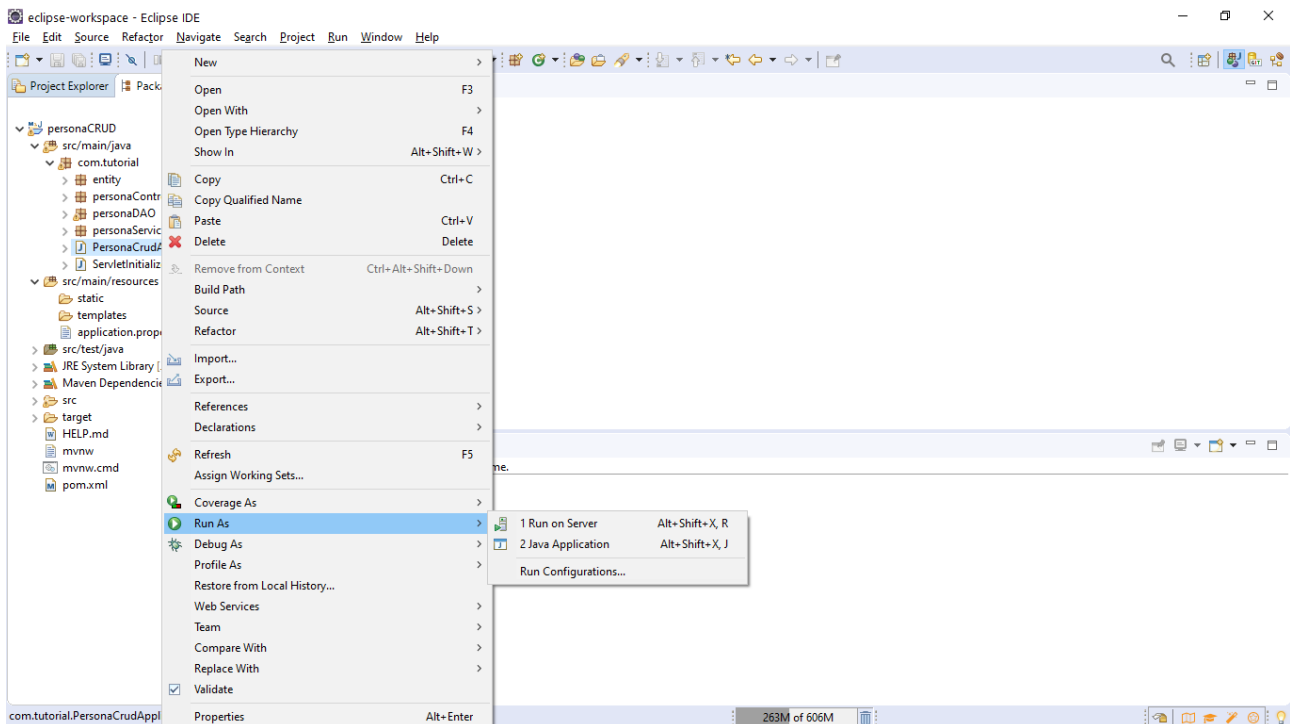


Nuestra base de datos es local para poder realizar las primeras pruebas sin embargo, como veremos más adelante, la desplegaremos en [GearHost](#) para que esté a disposición de nuestra api.

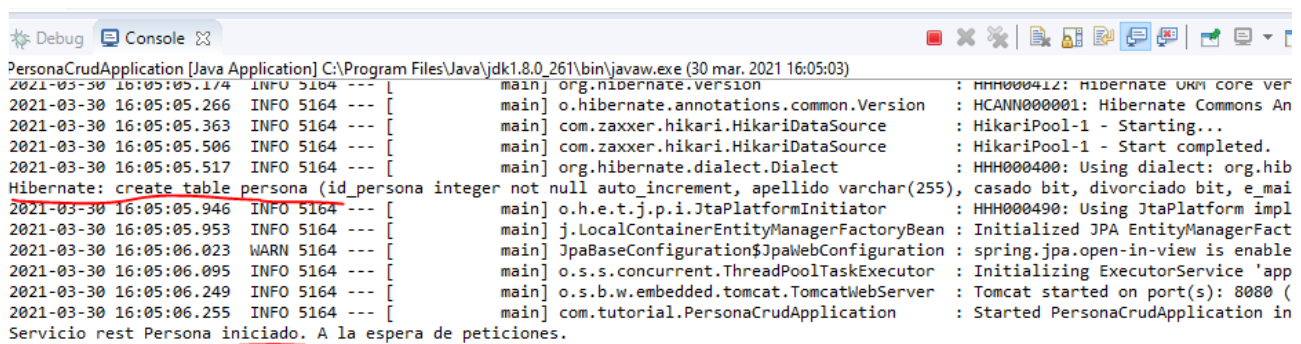
```
#spring.datasource.url=jdbc:mysql://den1.mysql4.gear.host:3306/persona
#spring.datasource.username=persona
#spring.datasource.password=persona
```

## 4. Puesta en marcha.

Nuestro servicio está listo para ser probado, para ello ejecutaremos el “PersonaCrudApplication” como aplicación java, tal y como vemos en la imagen de más abajo.

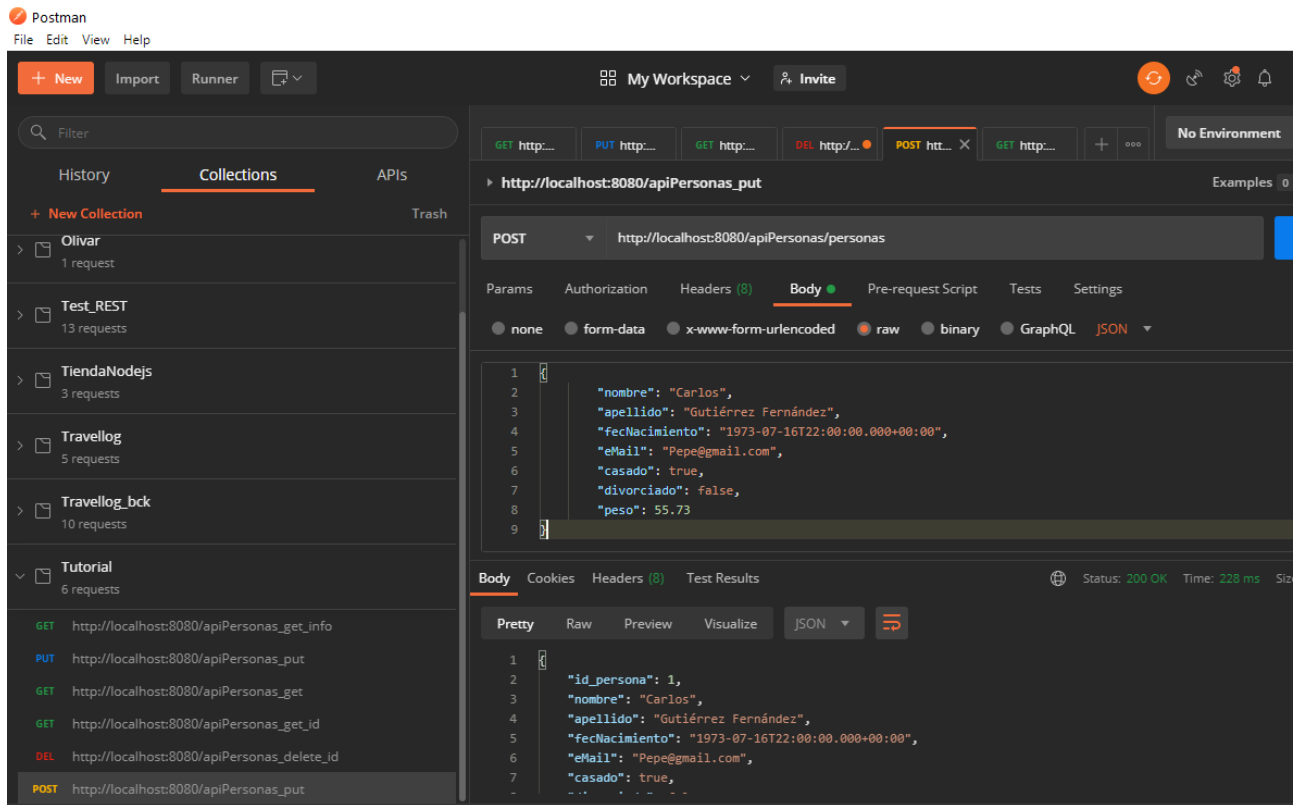


El resultado deberá ser un mensaje por consola que nos confirma que la tabla se ha creado y que el servicio está a la espera de peticiones.



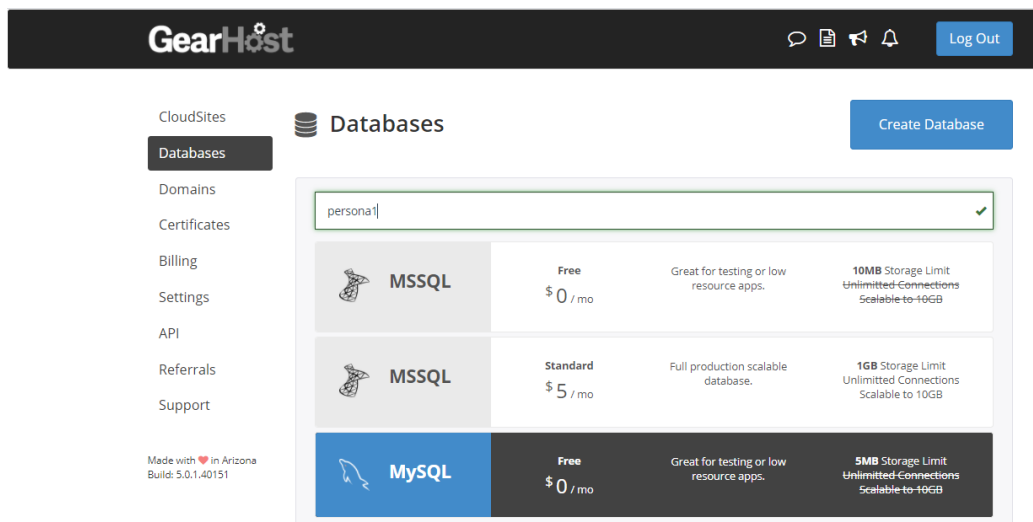
## 5. Pruebas en local

Para nuestras pruebas locales utilizaremos un conjunto de peticiones básicas (POST, PUT, etc) que lanzaremos mediante el Postman. Estas peticiones serán añadidas al proyecto en Github para poder importarlas desde el Postman y realizar las pruebas desde cualquier entorno.



## 6. Despliegue en GearHost de la BB.DD.

Para la creación de la BB.DD. en GearHost deberemos primero estar registrados y crear nuestra BB.DD previamente, también sin ninguna tabla. Debido a que el nombre “persona” ya estaba en uso, utilizaremos “persona1” como nombre.



La BB.DD. se generará en vacío con la contraseña que nosotros elijamos. Este sería el resultado, donde se nos muestra también la cadena de conexión.

The screenshot shows the GearHost dashboard. On the left is a sidebar with navigation links: CloudSites, Databases (selected), Domains, Certificates, Billing, Settings, API, Referrals, and Support. The main area has a header with the GearHost logo, a database icon, the name 'persona1', and a 'Log Out' button. Below this is a 'Free Database' section with a database icon and a message: 'You are using free database which is limited. It's recommended to upgrade to Standard.' with an 'Upgrade Now' button. A summary row shows 'database server' as 'den1.mysql2.gear.host', 'plan' as 'Free (upgrade)', and 'data utilization' as '0.00B'. Below this is a 'database users' table:

Username	Password	Permission
persona1	persona-1	Read & Write

At the bottom left, it says 'Made with ❤️ in Arizona Build: 5.0.1.40151'.

Como ya dijimos en el punto 6, persistencia, vamos a redirigir nuestro servicio hacia esta BB.DD alojada en GearHost. Para ello sólo tendremos que cambiar los parámetros de conexión en el fichero “application.properties” de nuestro proyecto. En nuestro caso, hemos comentado los de la conexión local. Volvemos a iniciar el servicio y veremos como se conecta.

The screenshot shows the Eclipse IDE. The 'Project Explorer' on the left shows the project structure for 'personaCRUD'. The 'Package Explorer' shows the 'src/main/resources' folder. The 'application.properties' file is open in the editor, showing the following configuration:

```
1 spring.datasource.url=jdbc:mysql://den1.mysql2.gear.host:3306/persona1
2 spring.datasource.username=persona1
3 spring.datasource.password=persona-1
4
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
7 spring.jpa.show-sql=true
8 spring.jpa.hibernate.ddl-auto=update
9
10 #spring.datasource.url=jdbc:mysql://localhost/persona
11 #spring.datasource.username=usuario
12 #spring.datasource.password=usuario
13
```

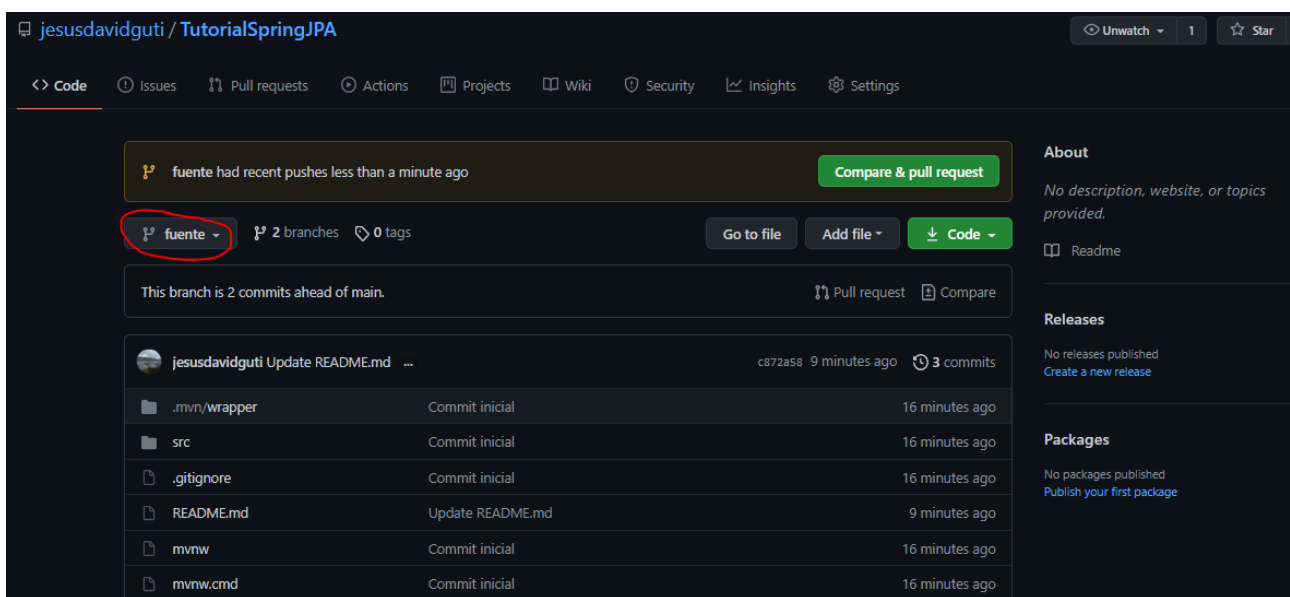
The 'Console' at the bottom shows the output of the application, indicating that the service is running and waiting for requests:

```
2021-03-30 16:44:09.894 INFO 13276 --- [main] org.hibernate.Version
2021-03-30 16:44:09.987 INFO 13276 --- [main] o.hibernate.annotations.common.Version
2021-03-30 16:44:10.090 INFO 13276 --- [main] com.zaxxer.hikari.HikariDataSource
2021-03-30 16:44:11.914 INFO 13276 --- [main] com.zaxxer.hikari.HikariDataSource
2021-03-30 16:44:11.925 INFO 13276 --- [main] org.hibernate.dialect.Dialect
2021-03-30 16:44:12.735 INFO 13276 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator
2021-03-30 16:44:12.741 INFO 13276 --- [main] j.LocalContainerEntityManagerFactoryBean
2021-03-30 16:44:12.814 WARN 13276 --- [main] JpaBaseConfiguration$JpaWebConfiguration
2021-03-30 16:44:12.886 INFO 13276 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor
2021-03-30 16:44:13.036 INFO 13276 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer
2021-03-30 16:44:13.042 INFO 13276 --- [main] com.tutorial.PersonaCrudApplication
Servicio rest Persona iniciado. A la espera de peticiones.
```

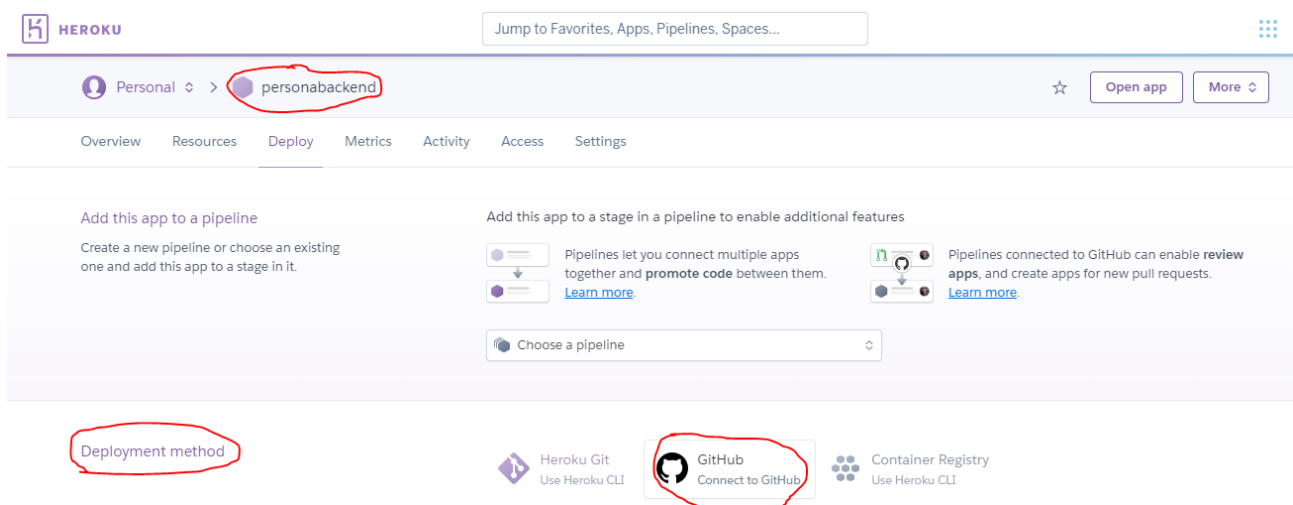
Una vez tengamos la conexión, podremos utilizar las mismas pruebas que hemos realizado en el punto 5 para verificar el correcto funcionamiento del servicio pero esta vez con la conexión redirigida hacia GearHost.

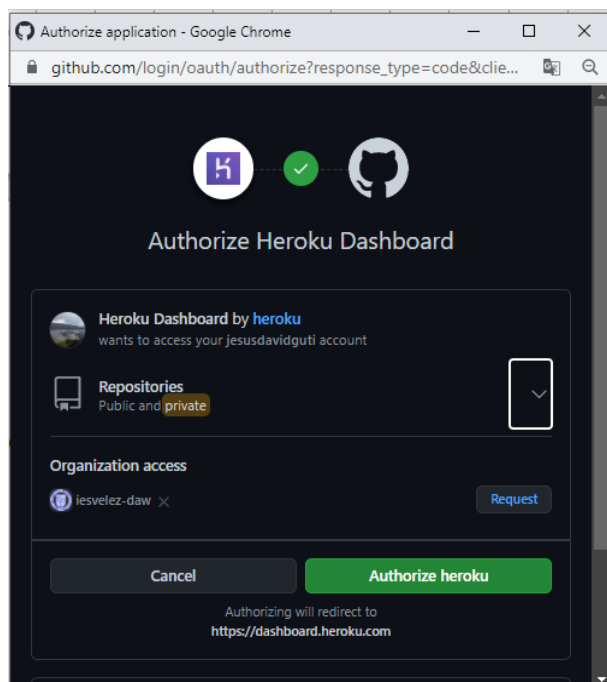
## 7. Despliegue de la aplicación en Heroku.

El servicio será desplegado en Heroku apoyándonos en la funcionalidad que ofrece para realizar los despliegues desde una rama de Github, lo cual facilita mucho este tipo de tareas. Por ello, primero crearemos una rama llamada “fuente” en nuestro repositorio Github donde ubicaremos nuestro código fuente.



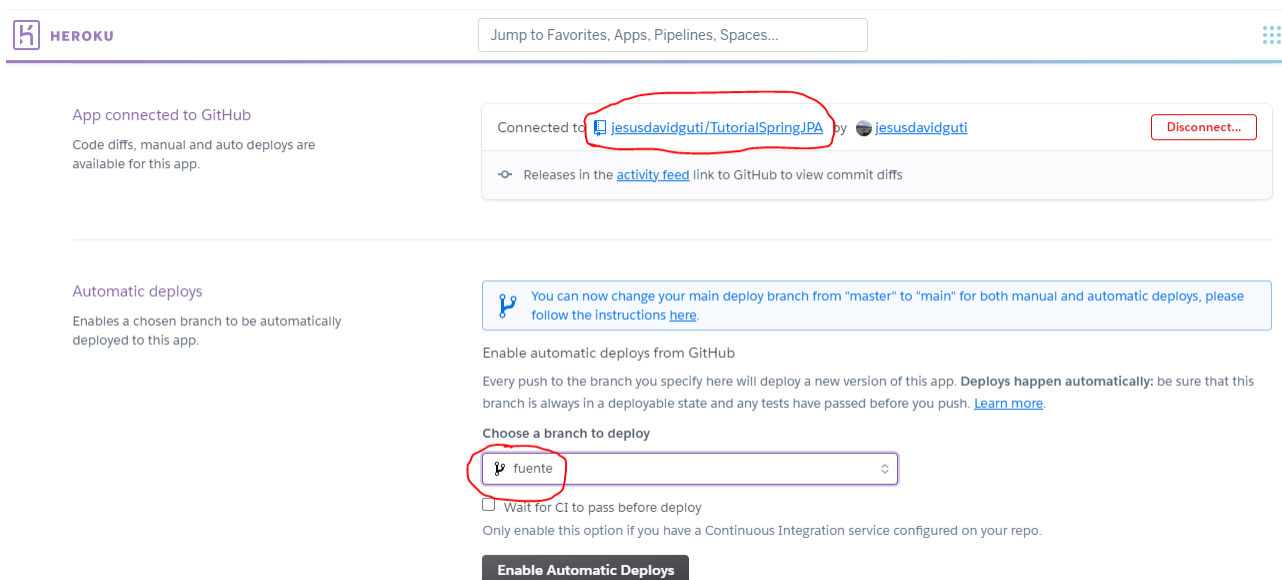
En Heroku, una vez estemos registrados, crearemos la aplicación con el nombre que estimemos oportuno, en este caso “personabackend”, y utilizaremos la opción de Github como forma de despliegue.





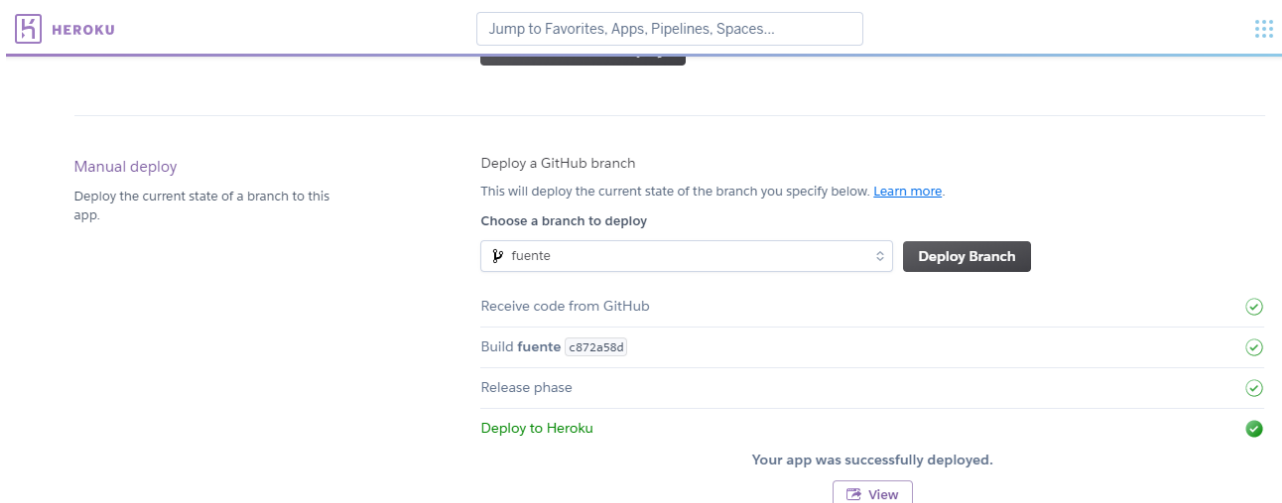
Deberemos de autorizar a Heroku para que conecte con nuestro repositorio.

Una vez conectados a Github, deberemos elegir desde qué repositorio desplegaremos y la rama en cuestión, caso de que fuera distinta de main como es nuestro caso. Optaremos por un despliegue manual, el cual podremos cambiar en el futuro.

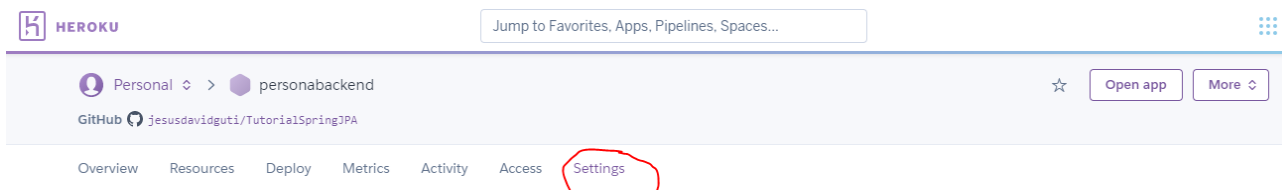




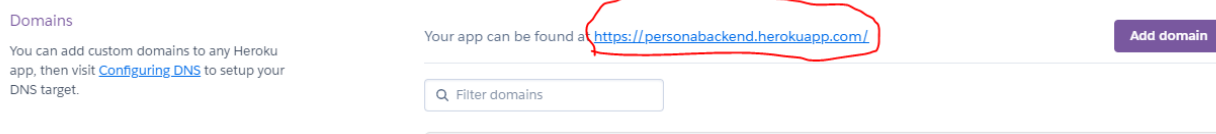
Si todo ha ido bien, veremos una pantalla como la siguiente. En caso de error, podremos ver los logs del despliegue que nos darán información sobre el error que se ha producido.



Desde la opción “settings” podremos ver la url donde se ha desplegado nuestra aplicación.

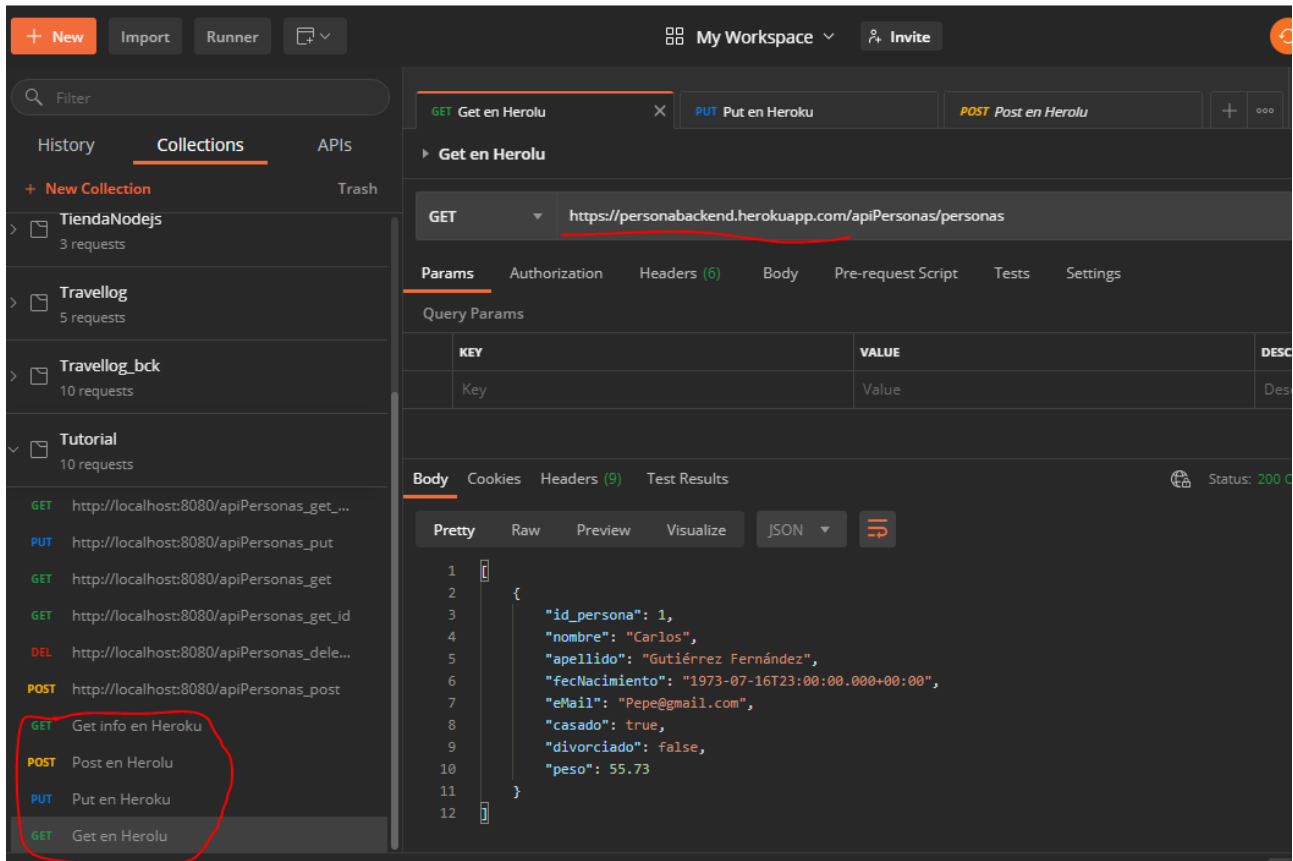


Aparece en la parte de más abajo de todas las opciones.



## 8. Pruebas en el servidor.

Una vez desplegada la aplicación y la BB.DD. podremos realizar pruebas, tal y como hicimos en el punto 5 pero apuntando a nuestra aplicación desplegada. Bastará con modificar la url de nuestras pruebas en local para poder ver el resultado.



## 9. Documentación.

El software, así como la documentación adjunta, están ubicados en el repositorio [TutorialSpringJPA](#). El software está en la rama “fuente”, mientras que la documentación está en la rama “main”

Las pruebas que hemos visto a lo largo de todo el documento serán adjuntadas mediante un fichero en Github para que sean importadas en Postman.