	CÓDIGO DEL PROYECTO: 1072
	TÍTULO: SPRING BOOT FRAMEWORK
	AUTOR: Ismael Heras Salvador



**IES LOS SAUCES
BENAVENTE (ZAMORA)**



**TITULO GRADO SUPERIOR DE
DESARROLLO DE APLICACIONES WEB**

DEPARTAMENTO: INFORMÁTICA

TITULO PROYECTO: SPRING BOOT FRAMEWORK

AUTOR: Ismael Heras Salvador

CURSO: 2019/2020


PERIODO: JUNIO

TIPO DE PROYECTO: Proyecto de investigación y desarrollo

CÓDIGO DE PROYECTO: 1072

TUTORÍA COLECTIVA: Amor Rodríguez Navarro


TUTORÍA INDIVIDUAL: María Criado Domínguez

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

SPRING BOOT FRAMEWORK

Presentación del famoso framework de java, spring boot, todas las herramientas necesarias para desarrollar en él y demostración práctica con un CRUD.



	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

El presente proyecto de investigación y desarrollo cuyo título es SPRING BOOT FRAMEWORK del módulo PROYECTO ha sido realizado por Ismael Heras Salvador del IES LOS SAUCES del Ciclo Formativo Grado Superior Desarrollo de aplicaciones web con el fin de la obtención del Título de Técnico Superior en Desarrollo de Aplicaciones Web.

La tutoría individual de dicho proyecto ha sido llevada a cabo por D^a. María Criado Domínguez profesor de segundo curso de CFGS Desarrollo de Aplicaciones web.


En Benavente, __ de mayo de 2020

Fdo. NOMBRE Y
APELLIDOS


Fdo. Amor Rodríguez
Navarro

Fdo.: María Criado
Domínguez

Fdo.: Ismael Heras
Salvador

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

1	Resumen del proyecto	5
2	Conceptualización del problema.	5
3	Objetivos	6
4	Que es spring boot y sus características.....	6
5	Tecnologías de Spring Boot.....	8
5.1	Dependencias.....	8
5.2	Base de datos MYSQL.....	13
5.3	Método de conexión a la base de datos JPA	14
5.4	Anotaciones y beans	15
5.5	Contenedores y anotaciones en nuestro CRUD.....	21
5.6	Patrones de diseño en el scope de SPRING BOOT	25
5.7	Spring Security.....	27
5.8	Modelo Vista Controlador (MVC)	31
5.9	Servidor Tomcat embebido en Spring Boot	32
6	Implementación.....	32
6.1	Modelo de red, equipos y sistemas operativos	32
6.2	IDE y herramientas de desarrollo utilizados.....	34
6.3	Lenguajes de programación y lenguajes de marcas utilizados.	34
6.4	SGBD o sistema de almacenamiento utilizado.....	36
6.5	Repositorio de software	38
6.6	Otras tecnologías utilizadas de Spring Boot	38
6.7	Análisis y diseño del supuesto práctico:	39
6.8	Modelo de clases.....	40
6.9	Modelo de casos de uso	41
6.10	Árbol de navegación	42
6.11	Modelo físico de datos	43
7	Conclusiones y líneas futuras.....	44
8	Anexos	45
8.1	Instalación de spring boot.....	45
8.2	Despliegue de la aplicación en heroku	52

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

1 Resumen del proyecto

El proyecto que queremos desarrollar es un simple **CRUD** con **Spring Boot** para ver lo sencillo que es realizarlo y ver todas las tecnologías utilizadas para desarrollarlo y como utilizarlas en un futuro proyecto de verdad.

Antes de desarrollar el proyecto haremos una investigación teórica de que es Spring Boot, que tecnologías implementa (que son los beans, dependencias, anotaciones) en que nos ayuda y como trabajar en este **framework de java**.


También estudiaremos otras tecnologías necesarias para el desarrollo de aplicaciones tales como bases de datos, conectores para la **base de datos, IDE** que vamos a utilizar y por ultimo documentaremos la instalación de Spring Boot y el despliegue de la misma en **Heroku**.

2 Conceptualización del problema.

La temática general del problema que vamos a describir es la presentación del framework de java **SPRING BOOT** y las tecnologías utilizadas para poder llevar a cabo su desarrollo.

La principal razón que me han llevado a estudiar este **framework** es la laboral, ya que **JAVA** y sus Frameworks son lo más utilizado en el mundo empresarial.

Considero que este asunto es importante de estudio por que el framework lo veo con mucho futuro, ya que hoy por hoy los desarrolladores de software solo se dedican a eso a desarrollar la lógica y no a la tediosa **configuración del entorno**, para eso existen otros puestos de trabajo específicos, y nuestro framework de estudio nos proporciona una fácil configuración de nuestro entorno de trabajo, hasta nos provee de un **servidor embebido**.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

3 Objetivos

El objetivo de este proyecto es presentar el framework de java **Spring Boot**, para ello vamos a explicar que es y mostrar todas sus tecnologías y la instalación.

1. Como funciona.
2. Inyección de dependencias.
3. Clases.
4. Acceso a datos
5. Realización de un pequeño CRUD a modo de ejemplo.


4 Que es spring boot y sus características

Spring Boot es una herramienta que nos facilita el desarrollo de aplicaciones en el marco de trabajo del framework de **java Spring Core**. Con Spring Boot solo te centraras en el desarrollo de la aplicación, olvidándote de la compleja configuración de Spring Core.

Por lo tanto es una herramienta para desarrollar todo tipo de aplicaciones de java (Tanto WEB como de escritorio) en un marco ya pre configurado y listo para empezar a desarrollar.

Antes hemos mencionado que **Spring Boot** es una herramienta para implementar la configuración de **Spring Core**, este es un framework de java más popular para el mundo empresarial, para crear código de alto rendimiento, liviano y reutilizable. Su finalidad es estandarizar y agilizar los marcos de desarrollo, todo esto está enfocado a que los equipos de desarrollo se centren en la lógica de negocio.

Pero la principal desventaja que tiene Spring frente Spring Boot es que es difícil la configuración inicial, por eso vino Spring Boot, para facilitarnos las cosas y que los desarrolladores solo se centren en la lógica de negocio.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--


Por lo tanto las ventajas que tiene Spring Boot frente a Spring son la configuración inicial y la facilidad que se le da al desarrollador para iniciar un nuevo proyecto y ponerse a desarrollar de inmediato

Con todas estas tecnologías ya tenemos las bases para realizar nuestro proyecto, aunque hay muchas más tecnologías que más adelante mostraremos.

Características

Como antes ya hemos explicado lo que era Spring Boot en líneas generales ahora vamos a explicar las principales características de este framework de java.

- **Configuración:** Spring Boot viene con un complejo módulo de autoconfiguración para poder ejecutar la aplicación sin tener que definir absolutamente nada.
- **Dependencias:** Con Spring Boot solo determinaremos que tipo de proyecto vamos a desarrollar y el solo implementa todas las dependencias y librerías para que pueda funcionar.
- **Despliegue:** Spring Boot se puede ejecutar como una aplicación estándar, o como una aplicación WEB, ya que trae incorporado un servidor tomcat.
- **Métricas:** Spring Boot por defecto cuenta con servicios para saber el estado actual de nuestra aplicación, saber si está encendida o apagada memoria utilizada y disponible y número y detalle de los beans (en el siguiente capítulo explicaremos que son) creados por la aplicación.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

5 Tecnologías de Spring Boot

Como en todos los proyectos de cualquier lenguaje de programación necesitamos unas tecnologías adicionales para poder desarrollarlos completamente, bases de datos a utilizar gestor de dependencias método de conexión a la base de datos etc...

Por eso a continuación describiremos cada una de las herramientas para poder desarrollar en *Spring Boot*.

5.1 Dependencias


Existen dos configuraciones para esta herramienta que son maven o gradle (yo usare maven), los gestores de dependencias se crearon para simplificar los procesos de build, antes en los proyectos existía una persona para la creación de estas build.

Ahora los crea el programa automáticamente por lo que la agilidad al crear proyectos es mucho mayor que antes.

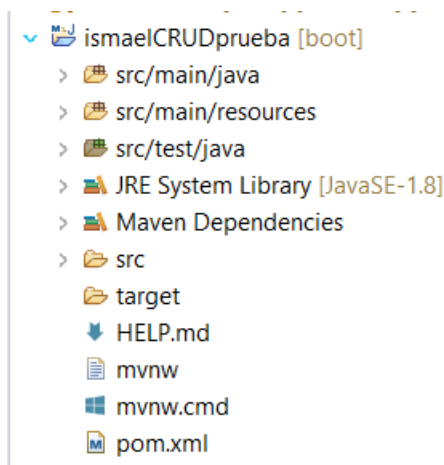
Maven es una herramienta para generar proyectos de java que se basa en ficheros XML. Su objetivo es la compilación del código y la generación del proyecto empaquetado, se encarga de obtener todas las dependencias del proyecto y de subirlo al repositorio final para que otros proyectos que dependan de él puedan usarlo.

Yo en mi proyecto he utilizado maven por los arquetipos. Un arquetipo es como una plantilla de proyecto, un patrón que define el tipo de proyecto, utilizo estos patrones para definir el esqueleto de mi aplicación. Maven nos da la posibilidad de elegir un patrón determinado (en mi caso elijo spring boot application starter) en función de nuestras necesidades y a partir de ahí empezar a trabajar casi al instante sin preocuparte de las librerías a usar, que versiones son compatibles y como crear la estructura del proyecto.

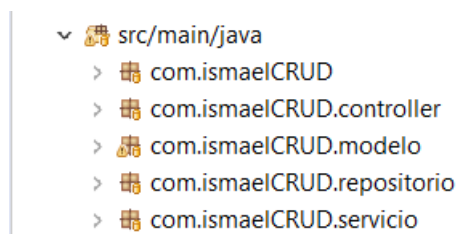
También se puede utilizar maven para desplegar el proyecto en un servidor o para generar test de calidad de código.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

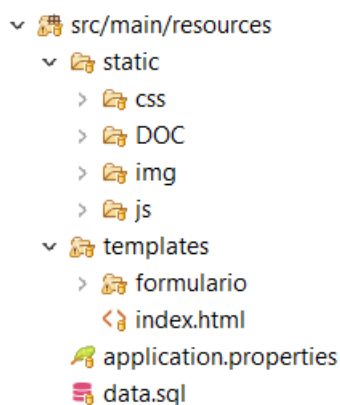
A continuación mostraremos la estructura de directorios que nos genera maven al iniciar un proyecto en Spring Boot y para qué es cada apartado.




Src/main/java: en este directorio es donde van todos los paquetes de nuestras clases, donde está el main el controlador el modelo y el servicio.



Src/main/resources: en este directorio va a ir la parte visual, los archivos HTML, CSS, JS, los documentos, las imágenes y donde va el archivo application-properties y el archivo de carga inicial de la base de datos data.sql.



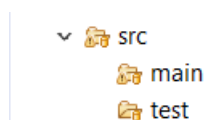
	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Src/test/java: aquí es donde va la herramienta para realizar un test automático de prueba de la clase o clases que queramos.

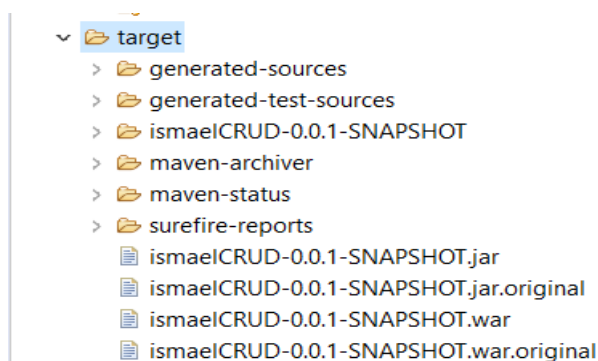
JRE System Library: en este directorio es donde maven nos aloja todas las librerías que necesita nuestro proyecto para funcionar.

Maven Dependencies: aquí es donde maven aloja las dependencias para todo nuestro proyecto.

Src: aquí es donde se junta los dos primeros directorios el main y el resources, es donde va alojado todo nuestro código a la hora de verlo en un repositorio o en los archivos del proyecto.




Target: en este directorio se guardan los archivos jar o war que se generan al compilar la aplicación y que luego sirven para desplegarla en un servidor.



HELP.md: este archivo se utiliza para poner las instrucciones de uso de la aplicación cuando se sube a un repositorio de software.

Mvnw y mvnw.cmd: estos archivos se utilizan para poder ejecutar el proyecto sin necesidad de tener instalado maven mvnw es para Linux y mvnw.cmd es para entornos Windows.

Pom.xml: en primer lugar pom responde a las siglas Project Object Model, es un fichero XML, que es la unidad principal de nuestro proyecto maven.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Contiene información acerca del proyecto, fuentes, test, dependencias, plugins, versiones etc...

Con todo lo dicho antes es uno de los archivos más importantes de nuestro proyecto ya que spring boot cada vez que ejecuta el programa mira este archivo.

Si queremos añadir dependencias a nuestro proyecto este es el archivo donde hay que realizarlas.


En la siguiente imagen mostraremos el pom.xml de nuestra aplicación y luego analizaremos cada parte

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.5.RELEASE</version>
    <relativePath/>
  </parent>
  <groupId>com.ismaelCRUD</groupId>
  <artifactId>ismaelCRUD</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ismaelCRUD</name>
  <description>Demo project for Spring Boot thymeleaf login</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

```

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

```

    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
      </plugins>
    </build>

  </project>

```

Ahora analizaremos cada parte de este importante archivo.

Parent: aplicación de donde hereda nuestra aplicación.

GroupID: es la raíz de nuestro proyecto (donde alojamos las clases del proyecto).

ArtifactId: es el nombre de nuestro proyecto.


Versión: es la versión (SNAPSHOT = en desarrollo).

Description: una pequeña descripción del proyecto.

Properties: las propiedades de nuestro proyecto.

Dependencies: librerías necesarias que nuestra aplicación necesita para funcionar.

Build: funcionalidades que añadimos a maven para que nos ayude a la gestión de nuestra aplicación.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

5.2 Base de datos MYSQL

Como todo proyecto para tener, una persistencia de datos necesitamos un motor de base de datos para guardar toda nuestra información.

En nuestro caso vamos a trabajar con mysql ya que es de las más utilizadas.

Con maven agregamos las dependencias al archivo pom.xml para poder utilizar la base de datos

```

1  <!-- Configuración del ORM de spring boot para persistencia -->
2  <dependency>
3      <groupId>org.springframework.boot</groupId>
4      <artifactId>spring-boot-starter-data-jpa</artifactId>
5  </dependency>
6
7  <!-- Conector/librería de MYSQL para java -->
8  <dependency>
9      <groupId>mysql</groupId>
10     <artifactId>mysql-connector-java</artifactId>
11     <scope>runtime</scope>
12 </dependency>
13


```

Y luego añadimos en el archivo **application.properties** lo relativo a la conexión a la base de datos.

```

1  #Data source
2  #Indica el driver/lib para conectar java a mysql
3  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
4
5
6  #Url donde esta el servicio de tu mysql y el nombre de la base de datos
7  spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
8
9  #Usuario y contraseña para tu base de datos descrita en la línea anterior
10 spring.datasource.username=root
11 spring.datasource.password=root
12
13 #[opcional]Imprime en tu consola las instrucciones hechas en tu base de datos.
14 spring.jpa.show-sql = true

```

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

5.3 Método de conexión a la base de datos JPA

Existen varios métodos para conexión a la base de datos en JAVA, pero nosotros vamos a utilizar El sistema JPA, que nos permite crear una correlación entre una base de datos relacional y un sistema orientado a objetos.

Esta correlación se llama ORM (Objet Relational Mapping) esta a su vez genera anotaciones sobre entidades.

JPA es el encargado de convertir los objetos de java en instrucciones para el manejador de base de datos (MDB).

Para poder utilizar JPA necesitamos instalar en nuestro archivo pom.xml las dependencias correspondientes.


```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Dependencia JPA en pom.xml

Al instalar las dependencias en el archivo pom.xml maven automáticamente nos importa las librerías necesarias para JPA.

```
> resources.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib
> rt.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib
> jsse.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib
> jce.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib
> charsets.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib
> jfr.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib
> access-bridge-64.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> cldrdata.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> dnsns.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> jaccess.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> jfxrt.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> localedata.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> nashorn.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> sunec.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> sunjce_provider.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> sunmscapi.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> sunpkcs11.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
> zipfs.jar - C:\Program Files\Java\jdk1.8.0_241\jre\lib\ext
```

Librerías para el funcionamiento de JPA

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

La relación que existe entre el modelo es decir nuestra clase Usuario y JPA es la siguiente.

Al crear una clase de java y anotarla con @Entity, JPA automáticamente la convierte en una tabla en nuestra base de datos.

Los atributos de la clase marcada como @Entity los marcamos con la anotación @Column y así estos se convertirán en columnas de nuestra tabla.

Como nota final podemos decir que JPA puede ser implementada por un gestor de persistencia de nuestra elección (Hibernate, TopLink, EclipseLink).

5.4 Anotaciones y beans


Hemos mencionado antes que eran anotaciones e identidades, vamos a profundizar más en lo que son estos conceptos:

Una **identidad** o **Bean** es una clase POJO que debe proporcionar un constructor por defecto (sin argumentos), todos los getters y setters de los atributos de la clase, no puede ser final y debe implementar Serializable para accesos remotos.

```

SpringCRUD2/pom.xml  *Persona.java
2
3 import java.io.Serializable;
4
5 import javax.persistence.Entity;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.GenerationType;
8 import javax.persistence.Id;
9 import javax.persistence.Table;
10
11 @Entity
12 @Table(name= "persona")
13 public class Persona implements Serializable{
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private int id;
18     private String nombre;
19     private String telefono;
20
21     public Persona() {
22         // TODO Auto-generated constructor stub
23     }
24

```

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Como podemos observar en la captura anterior hemos creado una clase llamada persona, hemos puesto sobre ella dos anotaciones, con la primera **@Entity** indicamos que es una entidad y con la segunda **@table (name = "persona")** le indicamos que cree una tabla en la base de datos con el nombre persona.

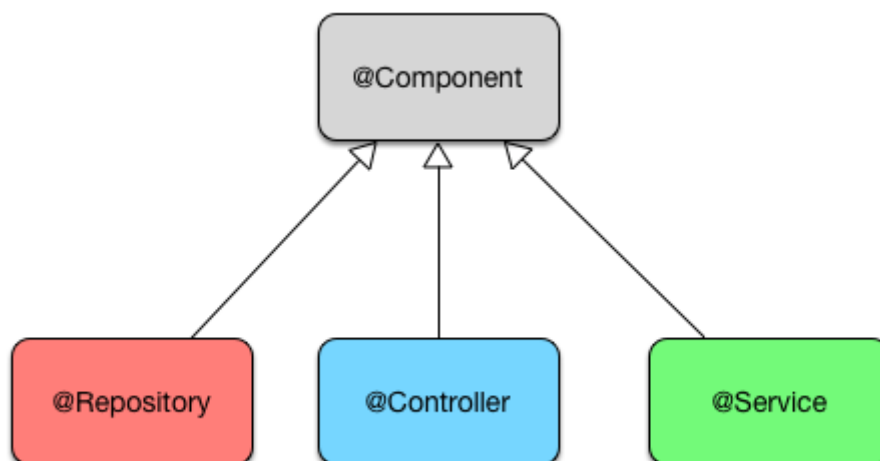
Ya en las propiedades de la clase hemos añadido otras anotaciones, **@Id y @GeneratedValue (Strategy = GenerationType.IDENTITY)**

Que indican que el atributo de la clase persona ID es la clave primaria en nuestra tabla persona.


Con todo esto podemos decir que las anotaciones son unas marcas para que categoricen cada uno de los componentes asociándoles una responsabilidad concreta. Y con los beans conseguimos que estas clases sean reutilizables y estén encapsuladas y tienen una mejor estructura.

Existen cuatro tipos de anotaciones generales que las explicaremos a continuación.

- **@Component** es el estereotipo general y permite anotar un bean (más adelante explicaremos que son los bean) para que spring lo considere uno de sus objetos.



Esquema de la anotación component

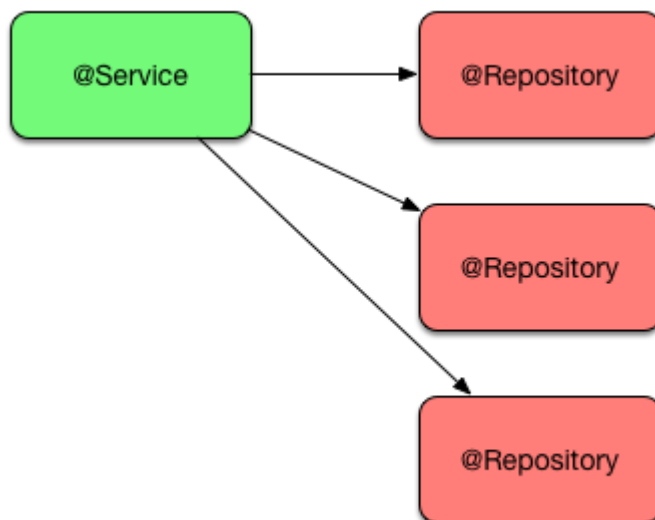
	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

- **@Repository** es el estereotipo que se encarga de dar de alta un bean para que implemente el patrón repositorio que es el encargado de almacenar datos en una base de datos.




Esquema de la anotación Repository

- **@Service** es el estereotipo que se encarga de gestionar las operaciones de negocio más importantes a nivel de aplicación, su tarea principal es la de agregador.

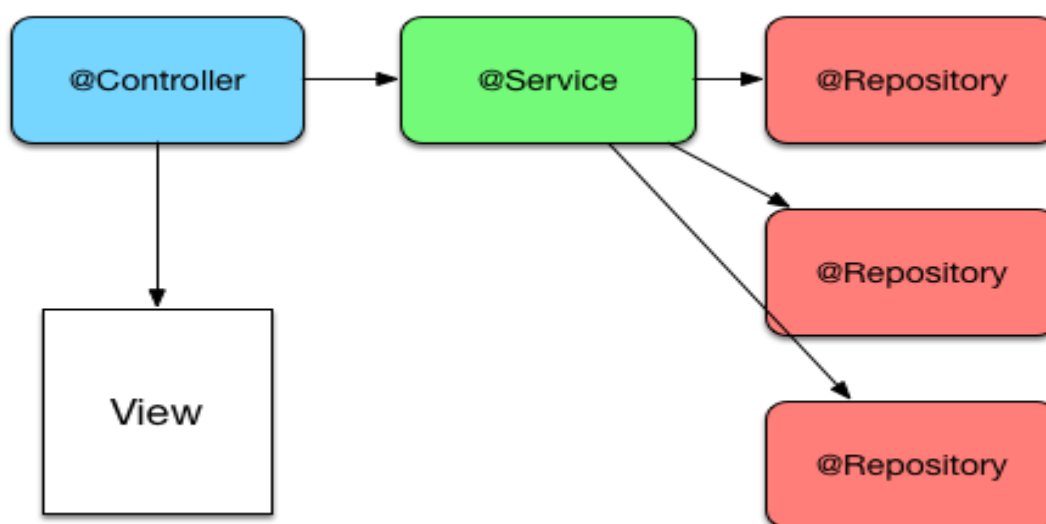


Esquema de la anotación Service

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

- **@Controller** realiza las tareas de controlador y gestión de la comunicación entre el usuario y la aplicación, normalmente se apoya en algún motor de plantillas que facilitan la creación de páginas.

•




Esquema de la anotación Controller

Ahora que ya sabemos que son las anotaciones vamos a repasar las anotaciones que hemos utilizado para realizar nuestro CRUD y para que hibernate y JPA puedan tener el acceso a la base de datos.

@SpringBootApplication: esta anotación se crea por defecto en la clase main para configurar todos los componentes necesarios para un proyecto en Spring Boot al iniciar nuestro proyecto.

@Configuration: esta anotación la utilizamos en nuestra clase SeguridadWeb, que es la clase donde configuramos todo para la seguridad de nuestra aplicación y la anotación indica que es una clase de configuración.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

@EnableWebSecurity: esta anotación también va en la clase SeguridadWeb y seguida de la anotación Configuration lo que nos hace es habilitar SpringSecurity en nuestro proyecto.

@Controller: la anotación Controller la utilizo en mi clase Ucontroller y lo que hace es indicarnos que la clase es un controlador y que va a manejar las peticiones @GetMapping y @PostMapping.

@GetMapping: Esta la utilizamos en los métodos de la clase Ucontroller para asignarles solicitudes HTTP GET, para la navegación por las distintas páginas de la aplicación.

@PostMapping: esta también es utilizada en la clase Ucontroller y la utilizamos en sus métodos para asignarles solicitudes HTTP POST, para pedir datos del usuario.


@Autowired: Lo que hace esta anotación es buscar un objeto que implementa nuestra interface ServicioUsuario, de esta manera no es necesario crear una instancia nueva del objeto cada vez que se necesite la funcionalidad de la clase que lo implemente.

@Entity: esta anotación es muy importante ya que nos permite indicarle a JPA que la clase marcada con esta anotación es una entidad (es decir nos creara una tabla en nuestra base de datos) y la clase marcada en mi aplicación con esta anotación es Usuario.

@Id: con esta anotación le decimos a un atributo de una clase marcada como @Entity que ese atributo es la clave primaria de nuestra entidad, en mi aplicación esta anotación la he aplicado sobre el atributo de la clase Usuario id.

@GeneratedValue: esta anotación la utilizamos también en el atributo id de la clase Usuario y como le habíamos indicado que es la clave primaria con esta anotación le decimos que sea autoincrementable.

@GenericGenerator: es lo mismo que la GeneratedValue solo que esta solo es para la sesión de hibernate.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

@Column: esta anotación nos permite indicarle JPA que un atributo de una clase marcada como Entity es una columna de nuestra base de datos, yo he marcado con esta anotación a los atributos de la clase Usuario nombre, apellidos, email, nombreusuario y password y por lo tanto JPA crea las columnas en nuestra base de datos con los nombres de los atributos de la clase.

@Size: esta anotación se realiza en los atributos de la clase Usuario para la validación del tamaño de los mismos, podemos restringir el tamaño mínimo y el máximo.


@NotBlank: esta anotación es un validador de cadenas de texto y valida que sea una cadena valida y longitud mínima de 1 carácter, con esta anotación marco los atributos de tipo cadena de la clase Usuario.

@Transient: esta anotación sirve para indicar a JPA que un atributo de una clase marcada como Entity no debe persistir en la base de datos, en mi aplicación he marcado el atributo confirmacionPassword de la clase Usuario porque no quiero que sea un campo de la base de datos.

@Repository: esta es otra de las anotaciones más importantes ya que nos indica que una clase o interface marcada con ella es un repositorio, es decir un mecanismo para encapsular el almacenamiento y la recuperación de datos de la base de datos. En la aplicación que he realizado he hecho esta marca sobre la interface RepositorioUsuario para indicarle que es La encargada de simular las consultas sql sobre la base de datos.

@Service: esta anotación sirve para indicar que una clase es un servicio, es decir donde vamos a aplicar la lógica de negocio, en la aplicación he añadido esta anotación a la clase ImpServicioUsuario y a LoginUsuario ya que es la encargada de borrar los usuarios, de editarlos de crearlos y de logearnos (es nuestra capa de negocio).

@Transactional: esta anotación sirve para marcar a la clase especificada que tiene como misión hacer una única transacción a la base de datos, en mi caso he marcado a la clase LoginUsuario, ya que es la encargada únicamente de hacer el login de usuario usando la base de datos.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

5.5 Contenedores y anotaciones en nuestro CRUD

Ahora que ya sabemos que son los beans y las anotaciones vamos a ver como lo podemos implementar en nuestra aplicación.

Lo primero es explicar que es el la **inyección de dependencias**.

Es un patrón de diseño orientado a objetos, cuya finalidad es la de suministrar objetos a una clase mediante el Spring **Container**, sin que la propia clase tenga que crearlos.

En resumen, con Spring Boot definimos objetos (Beans) y mediante el spring Container (Contenedores) los inyectamos en la clase que los necesite.

Como hemos **programado un CRUD** vamos a explicar cómo funciona todo con las anotaciones y los contenedores.


En primer lugar explicaremos que es un CRUD, es un acrónimo de las primeras cuatro letras de las operaciones fundamentales de aplicaciones con persistencia de datos. **C**reate (creación de usuarios), **R**ead (leer registros), **U**ppdate (actualizar registros) y **D**eleate (borrar registros), en resumen son las funciones requeridas por el usuario para crear y gestionar datos.

Nuestra aplicación es un CRUD donde tenemos una página WEB que podemos registrarnos y una vez registrados podemos logearnos con nuestras credenciales de nombre de usuario y password, y luego podemos actualizar y borrar los usuarios y con esto tendríamos los cuatro puntos del CRUD cubiertos.

Ahora podremos ver como utilizamos y donde las **anotaciones en nuestro CRUD**.

Lo primero es crear una clase modelo que en nuestro caso se llama Usuario, y anotarla con @Entity e implementamos Serializable para que puedan guardarse los datos, y con esta anotación JPA nos convierte la clase Usuario en una tabla de la base de datos.

```
@Entity
public class Usuario implements Serializable {
```

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

En nuestra clase Usuario creamos los atributos necesarios para la gestión de los usuarios en nuestra base de datos y le ponemos las anotaciones correspondientes para que los atributos se conviertan en columnas para nuestra tabla.

```

@Id
@GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
@GenericGenerator(name = "native", strategy = "native")
private long id;

@Column
@NotBlank
@Size(min = 2, max = 20, message = "Numero de caracteres invalido")
private String nombre;

@Column
@NotBlank
@Size(min = 2, max = 50, message = "Numero de caracteres invalido")
private String apellidos;

@Column
@NotBlank
private String email;

@Column
@NotBlank
@Size(min = 2, max = 15, message = "Numero de caracteres invalido")
private String nombreusuario;


@Column
@NotBlank
private String password;

@Transient
private String confirmacionPassword;

```

Como podemos ver hemos anotados los atributos de la clase Usuario con las anotaciones que vimos en el punto anterior @Column, @NotBlank y @Size, con esto conseguimos que los atributos sean las columnas de la tabla.

A continuación creamos la clase controlador, que recibe las peticiones del usuario y esta a su vez se las manda al modelo y el controlador las recibe y las vuelve a mandar al usuario (vista).

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

```

@Controller
public class UController {

    @Autowired
    ServicioUsuario servicioUsuario;

    @GetMapping({ "/", "index" })
    public String index() {
        return "index";
    }

    @GetMapping("/registro")
    public String registro(Model modelo) {
        modelo.addAttribute("formulario", new Usuario());
        modelo.addAttribute("formTab", "active");
        return "formulario/registro";
    }


    @GetMapping("/vista")
    public String getFormulario(Model modelo) {
        modelo.addAttribute("formulario", new Usuario());
        modelo.addAttribute("userList", servicioUsuario.getAllUsers());
        modelo.addAttribute("listTab", "active");
        return "formulario/vista";
    }

    @PostMapping("/registro")
    public String crearUsuario(@Valid @ModelAttribute("formulario") Usuario usuario, BindingResult resultado,
        ModelMap modelo) {
        modelo.addAttribute("formulario", usuario);
        modelo.addAttribute("registro", true);
        if (resultado.hasErrors()) {
            modelo.addAttribute("formulario", usuario);
            modelo.addAttribute("formTab", "active");
            return "formulario/registro";
        }
    }
}

```

Como podemos ver en la imagen la clase es marcada con la anotación @Controller que vimos en el punto anterior.

Los métodos también están marcados con las anotaciones @GetMapping y @PostMapping para la navegación de las páginas y recibir las peticiones del formulario de los usuarios.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

También creamos una interface y añadimos la anotación `@Repository` y extendemos la clase `CrudRepository` para indicar que es la encargada de las peticiones a la base de datos.

```
@Repository
public interface RepositorioUsuario extends CrudRepository<Usuario, Long> {
```


Continuamos creando la clase login para poder realizar la autenticación de los usuarios mediante la base de datos, y la añadimos para ello las anotaciones `@Service` y `@Transactional`, y con esto le indicamos que esta clase es un servicio.

```
21 @Service
22 @Transactional
23 public class LoginUsuario implements UserDetailsService {
24
```

Por ultimo creamos la clase `ImpServicioUsuario` y la anotamos también con `@Service`, esta clase es un servicio y sirve para realizar todas las operaciones del CRUD.

```
@Service
public class ImpServicioUsuario implements ServicioUsuario {
```

Con esto ya sabemos que son las anotaciones y como las hemos implementado en nuestra aplicación.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

5.6 Patrones de diseño en el scope de SPRING BOOT

Como ya hemos dicho en el punto anterior, la inyección de dependencias es un patrón de diseño orientado a objetos.

Existen varios patrones principales de diseño que son el singleton y el prototype.


- **Singleton** este patrón de diseño tiene como objetivo asegurar que solo allá una instancia u objeto por clase y un punto global a ella.

Las ventajas que tiene es que tiene un control estricto a como se accede a las instancias y un mejor desempeño de la herencia, este es el patrón por defecto en el core de Spring.

- **Prototype** este patrón es lo contrario al singleton, su objetivo es la creación de varios objetos a partir de un modelo o prototipo, esto lo hace con la clonación de objetos creados previamente.

Siempre es más óptimo clonar un objeto que no crearlo desde cero, el objeto tendrá sus propios valores desde los setters.

- **Request** en este patrón el contenedor de Spring creara una nueva instancia del objeto definido por el bean cada vez que reciba un HTTP Request.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

```

Bean de ámbito Request
2  @Bean
3  @Scope(value = WebApplicationContext.SCOPE_REQUEST)
4  public BeanSample beanSample() {
5      return new BeanSample ();
6  }
7
8  // Beans basados en @Component
9  @RestController()
10 @RequestScope
11 public class RequestScopedController extends AbstractController{
12
13     @GetMapping(AbstractController.REQUEST_SCOPE_ENDPOINT)
14     public String getUuid() {
15         return super.getUuid();
16     }
17
18 }


```

- **Session** el contenedor de Spring creara una nueva instancia del objeto definido por el bean por cada una de las sesiones HTTP y entregara esa misma instancia cada vez que reciba una nueva petición dentro de la misma sesión.

```

1  // Beans basados en anotaciones @Bean
2  @Bean
3  @Scope(value = WebApplicationContext.SCOPE_SESSION)
4  public BeanSample beanSample() {
5      return new BeanSample ();
6  }
7
8  // Beans basados en componentes
9  @RestController
10 @SessionScope
11 public class SessionScopedController extends AbstractController {
12
13     @GetMapping(AbstractController.SESSION_SCOPE_ENDPOINT)
14     public String getUuid() {
15         return super.getUuid();
16     }
17 }

```

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

- **Websockets** el contenedor de Spring creará una nueva instancia del objeto definido por el bean y la devolverá por cada petición que se produzca dentro del ciclo de vida de un websocket.

Un websocket es una tecnología avanzada que hace posible abrir una sesión comunicación interactiva entre el navegador del cliente y el servidor, con esta API puedes enviar mensajes a un servidor y recibir mensajes controlados por eventos sin tener que consultar el servidor para una respuesta.

En resumen el patrón de dependencias nos permite crear aplicaciones más fácilmente ya que nos permite unir las distintas dependencias con las clases que las necesitan.


Spring Container es el aliado perfecto para usar estos patrones de diseño ya que nos permite configurar las dependencias a inyectar mediante anotaciones o código XML de manera que podemos librar a nuestras clases de tener código innecesario.

En nuestra aplicación utilizaremos el patrón singleton (el que utiliza por defecto Spring).

5.7 Spring Security

Spring security es un **marco de autenticación y control de acceso**, potente y altamente configurable en el entorno de spring.

Spring security se centra en proporcionar autenticación y autorización a las aplicaciones java. El verdadero poder de esta herramienta se encuentra en la facilidad con la que se puede extender para cumplir los requisitos personalizados.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

A continuación enumeraremos algunas de sus **características**.

1. Soporte completo y extensible para **autenticación y autorización**.
2. Protección contra ataques como la fijación de sesiones, **falsificación de solicitudes entre sitios** etc...
3. Integración API Servlet.
4. Integración con **Spring MVC**.

En nuestro **CRUD** como no podía ser de otra manera hemos utilizado Spring Security para la autenticación y autorización.

Lo primero es añadir al archivo **pom.xml** la dependencia de Spring Security

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Una vez añadida spring boot nos instalara todo lo necesario para poder utilizar esta herramienta.


Para configurar toda la seguridad de nuestra aplicación tenemos que crear una clase llamada SeguridadWeb y extender de la clase de spring Security **WebsecurityConfigurerAdapter**, desde esa clase controlaremos toda la seguridad de nuestra aplicación.

```

@Configuration
@EnableWebSecurity
public class SeguridadWeb extends WebSecurityConfigurerAdapter {

```

Como vemos Añadimos las anotaciones que vimos anteriormente para habilitar la seguridad en spring security.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

A continuación crearemos un array donde indicaremos las carpetas que tienen permiso para acceder.

```

4 @Configuration
5 @EnableWebSecurity
6 public class SeguridadWeb extends WebSecurityConfigurerAdapter {
7
8     // arreglo donde nombramos las carpetas que en el siguiente metodo le damos
9     // permiso para entrar si no nos cargaria el js ni los css ni las imagenes
10    String[] resources = new String[] { "/css/**", "/img/**", "/js/**", "/DOC/**" };
11

```


Después debemos implementar un método de la clase que hemos extendido que se llama configure, es aquí donde programaremos el grueso de la seguridad de nuestra WEB.

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        // cualquiera que tenga la ruta resources tiene acceso
        .antMatchers(resources).permitAll()
        // cualquiera tiene acceso al index principal
        .antMatchers("/", "/index", "/registro").permitAll()
        // cualquier request necesitara autentificacion
        .anyRequest().authenticated().and().formLogin()
        // esta es la pagina que utilizamos en el login que esta permitida la entrada a
        // todo el mundo.
        .loginPage("/index").permitAll()
        // nos redirige ala lista al logearnos con exito
        .defaultSuccessUrl("/vista")
        // si falla el login nos muestra error
        .failureUrl("/index?error=verdad")
        // estos parametros se corresponden a los campos name de nuestro formulario de
        // login
        .usernameParameter("nombreusuario").passwordParameter("password")
        // y con esta ultima instruccion le decimos que cuando pulsemos el logout nos
        // cierre la sesion y nos redigira al login
        .and().csrf().disable().logout().permitAll().logoutSuccessUrl("/index?logout");
}

```

Como podemos ver toda la seguridad se centra al login de los usuarios que van a acceder a nuestra aplicación, y el cierre de sesión de los usuarios.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

El último punto de seguridad de nuestro CRUD es el **cifrado de la password** que introduce el usuario al registrase

Spring Security nos ofrece la clase BCryptPasswordEncoder. Para cifrar nuestra contraseña

```
BCryptPasswordEncoder passwordDescifrada;


// este es el metodo de la clase para decodificar la contraseña.
@Bean
public BCryptPasswordEncoder passwordEncoder() {
    // el numero 4 indica la longitud de la contraseña.
    passwordDescifrada = new BCryptPasswordEncoder(4);
    return passwordDescifrada;
}

// hacemos uso de la interface userDetailservice que se utiliza para cargar los
// datos del usuario y implementa el metodoAuthenticationManagerBuilder
@Autowired
UserDetailsService userDetailsService;

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    // Especifica el encargado del login y desenscriptacion del password
    auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
}
```

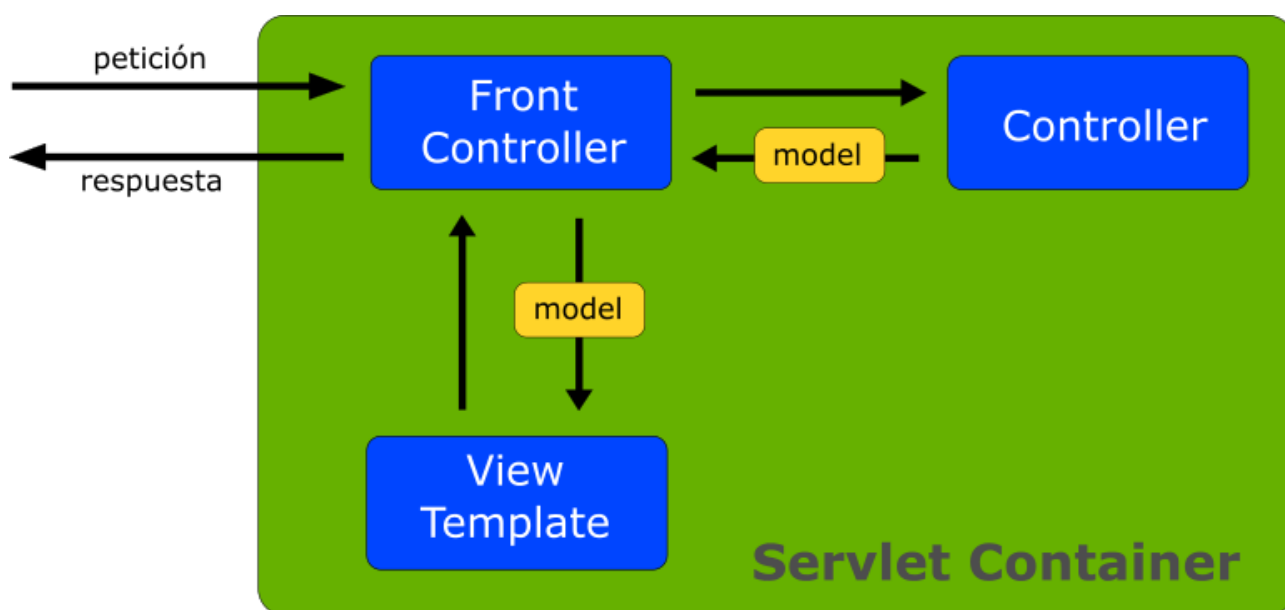
Hacemos uso de la interface UserDetailsService para cargar los datos del usuario he implementamos el método AuthenticationMnagerBuilder, que será el encargado de desenscriptar la password y hacer el login.

Y con esto ya tendríamos toda la configuración e seguridad de nuestra aplicación


	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

5.8 Modelo Vista Controlador (MVC)

El modelo vista controlador (MVC) es un patrón de diseño de software que nos facilita el proceso de creación de aplicaciones WEB, donde el modelo representa los datos, la vista son los elementos de la interface de usuario UI y el controlador es el encargado de manipular los datos en base a la interacción del usuario.



En nuestra aplicación también hemos utilizado este patrón de diseño, tenemos la clase **Usuario** que es nuestro modelo (es donde JPA ha creado la tabla y las columnas para nuestra base de datos), la clase **Ucontroller** que es nuestro controlador (es la encargada de recibir las peticiones de nuestra vista y pedir las a el modelo para volver a entregar a la vista en una respuesta), la vista son los elementos de la interface de usuario (los mecanismos que tiene el usuario para comunicarse con el controlador) en nuestro caso serían el formulario para el registro, el login y la lista con los datos de los usuarios.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

5.9 Servidor Tomcat embebido en Spring Boot

Cuando creamos una aplicación con Spring Boot este utiliza un servidor tomcat embebido para el despliegue, al empaquetarlo como jar, spring va auto configurarnos algunos aspectos de las aplicación para solo tener que centrarnos en el desarrollo, a diferencia del empaquetado como war, donde debemos de configurarlo nosotros mismos.

Otra ventaja del servidor embebido tomcat que nos facilita Spring Boot es que de esta forma, evitamos el problema de tener que mantener la sincronización entre las versiones de la aplicación y la configuración del servidor, es decir, todo es manejado desde la aplicación.


De esta forma se simplifica mucho el despliegue en plataformas como Heroku, AWS o Microsoft Azure, facilitando la integración continua.

6 Implementación

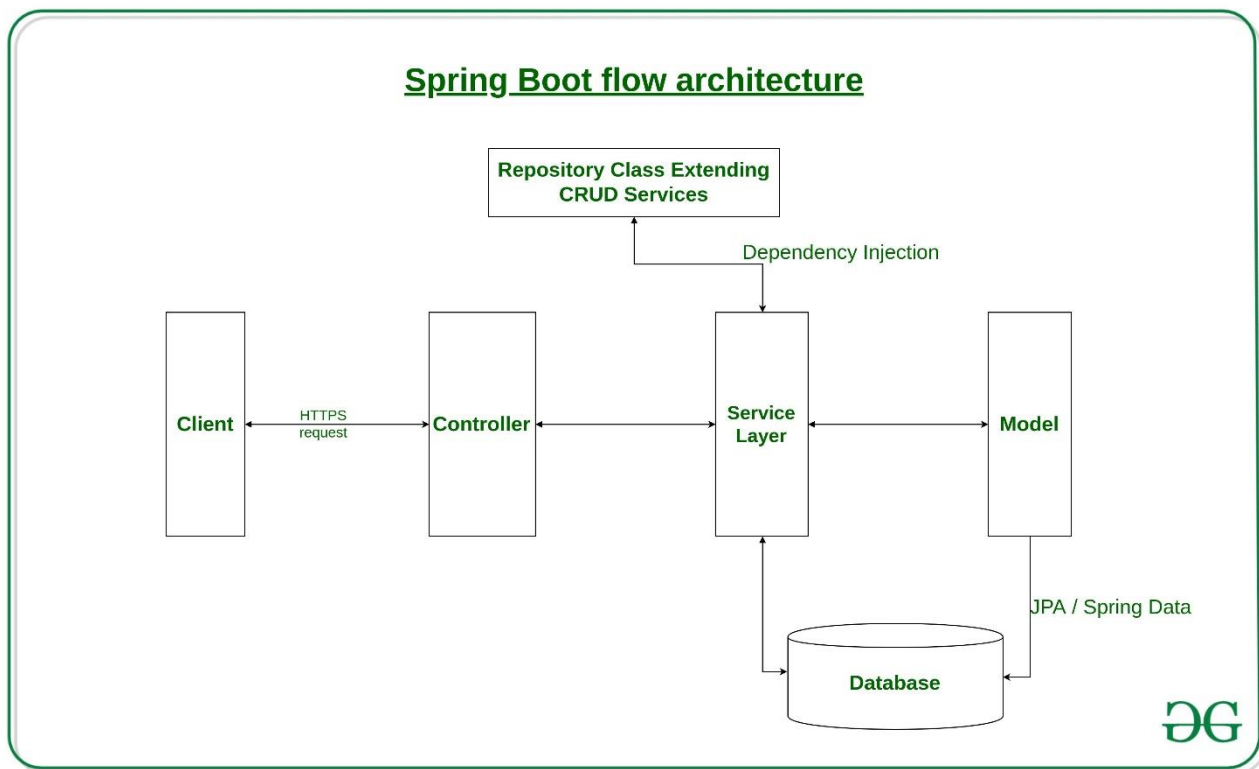
6.1 Modelo de red, equipos y sistemas operativos

El modelo de red que he utilizado es **RED DE AREA PERSONAL O PAM** ya que todo lo he hecho en mi casa particular mente.

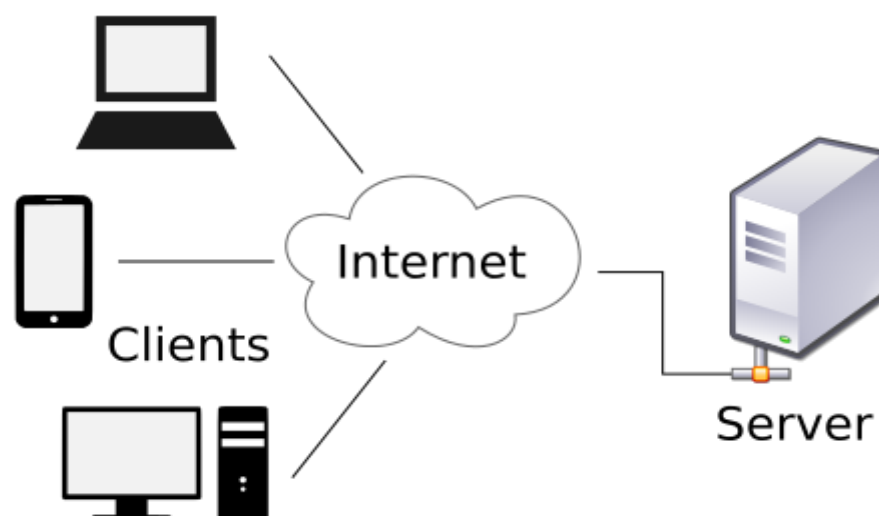
He utilizado un ordenador portátil con un sistema operativo Windows 10.


	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Aquí tendríamos un esquema del modelo de arquitectura de nuestro CRUD en **Spring Boot**



Y aquí el modelo de red utilizado



	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

6.2 IDE y herramientas de desarrollo utilizados

Instalaremos el Spring Tools Suite, que es un IDE basado en eclipse por lo cual es portátil y está muy optimizado para Spring Boot, pero se podría utilizar cualquier IDE del mundo de java (NetBeans, IntelliJ, Eclipse).

No necesitaremos instalar gestor de dependencias ya que Spring Boot ya tiene instalado el que quieras utilizar, **maven o gradle**.

6.3 Lenguajes de programación y lenguajes de marcas utilizados.


Los lenguajes utilizados son **JAVA y JAVASCRIPT** como lenguajes principales de programación.

JAVA es un lenguaje de programación **orientado a objetos** y multiplataforma, es un lenguaje compilado ya que corre sobre una máquina virtual y ha de ser traducido para que esta lo pueda entender, se creó en los años noventa por una empresa llamada Sun Microsystem y posteriormente adquirido por Oracle. En la actualidad puede utilizarse gratuitamente.

JAVA es el principal lenguaje de programación que he utilizado para la realización de la aplicación, con el he construido todas las clases he interfaces necesarias para poder tener acceso a la Base de datos y poder realizar todas las operaciones de nuestro CRUD.

JAVASCRIPT es un lenguaje de programación interpretado se define como orientado a objetos y basado en prototipos imperativo y débilmente tipado, se utiliza principalmente en el lado del cliente como parte de un navegador web para mejorar la interfaces de usuario y permitir que las páginas WEB sean dinámicas.

JAVASCRIPT en nuestra aplicación lo hemos utilizado para la implementación de la librería de JavaScript datatable, para mostrar los modales de la página principal y para realizar el slider de la página principal.

	CÓDIGO DEL PROYECTO: 1072 TÍTULO: SPRING BOOT FRAMEWORK AUTOR: Ismael Heras Salvador
---	---

Y como lenguajes de marcas utilizaremos **HTML CSS Y XML**.

HTML (Hyper Text Markup Language) es un lenguaje de marcado para la realización de páginas WEB, es un estándar que define la estructura básica de toda página WEB, imágenes texto, videos etc...


HTML lo utilizo en mi aplicación para crear la estructura básica de nuestra página WEB.

CSS (Cascading Style Sheets) es un lenguaje de diseño gráfico para poder dotar de cualidades visuales y estéticas a un documento HTML.

CSS en nuestra aplicación lo utilizo para mejorar el aspecto visual de nuestra página web dotándola de animaciones colores y un mejor posicionamiento, ya que si solo tuviéramos HTML la página no luciría bonita.

XML (Extensible Markup Language) es un lenguaje de marcado utilizado para almacenar datos en forma legible, puede dar soporte a bases de datos o siendo útil cuando varias aplicaciones se tienen que comunicar entre sí.

XML lo utilizo en mi aplicación para manejar las dependencias que van en el archivo pom.xml de nuestra aplicación.

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

6.4 SGBD o sistema de almacenamiento utilizado

La importancia que hoy en día tienen las **bases de datos** es muy grande, la cantidad de información que se maneja y la necesidad de **almacenarla y consultarla** es muy importante por eso surgieron las bases de datos. Hoy en día las aplicaciones que todos conocemos no podrían existir **sin persistencia de datos**.

MYSQL es un **SGBD** (sistema gestor de bases de datos) para la administración de bases de datos relacionales, que está considerada la base de datos de código abierto más popular del mundo. En la actualidad pertenece a la empresa Oracle. Está escrito en c++. Utiliza el lenguaje de consultas SQL para realizar las consultas a nuestra base de datos.

Para la gestión de la base de datos vamos a utilizar el **phpmyadmin** que es un programa escrito en **PHP** que corre en nuestro navegador para gestionar **MYSQL**, realiza consultas y actualizaciones sobre nuestra BD, es muy popular en la gestión de las bases de datos ya que es muy simple de manejar.


Nos permite importar y exportar bases de datos y archivos csv, xml, pdf.

Para hacer la conexión a la base de datos de nuestra aplicación tenemos que introducir los parámetros en el archivo application-properties, tales como el puerto (mysql utiliza el puerto 3306) el nombre de nuestra base de datos y un nombre de usuario y password.

```

8
9 spring.datasource.url=jdbc:mysql://localhost:3306/ismaelCRUD?serverTimezone=UTC
10 spring.datasource.username=root
11 spring.datasource.password=
12 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
13 spring.jpa.hibernate.ddl-auto=update

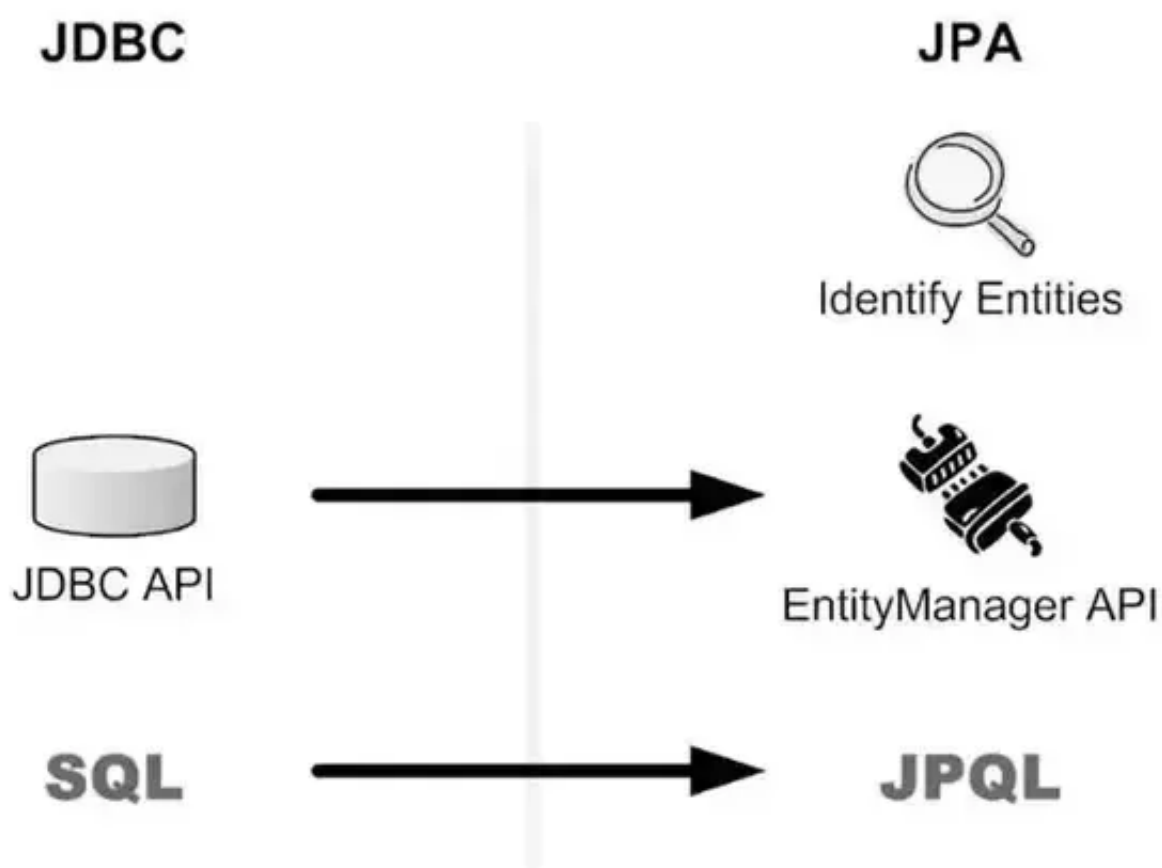
```


	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Para la conexión a la base de datos en java existen dos formas de hacerla que son mediante JDBC o JPA.

La principal diferencia entre **JPA** y **JDBC**, es que JDBC ejecuta directamente consultas nativas de SQL sobre la base de datos, mientras que en JPA permite interactuar con la base de datos por medio de objetos, de esta forma JPA es el Encargado de convertir los objetos de JAVA en instrucciones para el manejador de la base de datos (MDB).

Como ya hemos explicado en líneas anteriores en nuestra aplicación utilizaremos JPA.



	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

6.5 Repositorio de software

El repositorio de código que vamos a utilizar es el **GitHub**, es un repositorio de código que está en la web y está basado en **GIT**.

Los repositorios de código se utilizan para almacenar nuestro código en un sitio seguro para poder hacer nuestros commit y volver a un estado anterior que nos interese y mezclar ramas de desarrollo con la rama master.

[Repositorio GitHub de mi proyecto](#)

6.6 Otras tecnologías utilizadas de Spring Boot

THYMELEAF

Thymeleaf es un motor de plantillas que nos permite trabajar con archivos HTML en lugar de los clásicos JSP, se basa en incluir nuevos atributos (no etiquetas) a las etiquetas de HTML, esto se conoce como **natural templating**.

Esto permite que nuestra plantilla pueda renderizarse en local y que esa misma plantilla también sea procesada dentro del motor de plantillas, con lo cual las tareas de programación y diseño se pueden llevar conjuntamente.


```

<tbody>
  <tr th:each="usuario: ${userList}">
    <!-- <th scope="row">1</th> -->
    <td th:text="${usuario.id}"></td>
    <td th:text="${usuario.nombre}"></td>
    <td th:text="${usuario.apellidos}"></td>
    <td th:text="${usuario.nombreusuario}"></td>
    <td th:text="${usuario.email}"></td>
    <td>
      <a href="#" th:href="@{/editarUsuario/' + ${usuario.id}'}"><i class="fas fa-edit"></i></a>
      | <a href="#" th:onclick="'javascript:confirmacionBorrado('${usuario.id} + '\');'"><i class="fas fa-user-times"></i></a>
    </td>
  </tr>
</tbody>
</table>

```

Ejemplo de atributos en thymeleaf

Como podemos ver en la imagen tenemos una tabla en nuestra aplicación que muestra los datos de los usuarios, con thymeleaf creamos un for each con el atributo th: each y con la sintaxis usuario: \$(userlist) hacemos referencia al modelo de nuestra aplicación y cada campo muestra los datos de nuestros usuarios, en realidad es como si estuviera llamando a los métodos get de nuestra clase modelo.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

Como hemos visto en la imagen utilizamos varios atributos y dentro de estas expresiones propias de thymeleaf que no comprendemos, por eso vamos a repasar las expresiones y los atributos más utilizados que nos ofrece thymeleaf.

ATRIBUTOS BASICOS

th: text: permite reemplazar el texto de la etiqueta por el valor de la expresión que le demos.

```
<p th:text="$ (hola)">hola</p>
```

th: each: nos permite iterar sobre los elemento de una colección.

```
<li th: each="coche: $(coches)"
th:text="$ (coche.matricula)"></li>
```

TIPOS DE EXPRESIONES

Expresiones variables: son las más utilizadas y se representan de la siguiente manera: **\$()**.


Expresiones de selección: nos permiten reducir la longitud de la expresión si prefijamos un objeto mediante una expresión variable, se utiliza de la siguiente manera. ***()**

Expresiones de enlace: nos permite crear URL que pueden tener parámetros o variables, se utilizan con la siguiente nomenclatura. **@()**.

6.7 Análisis y diseño del supuesto práctico:

El análisis de nuestra aplicación es muy sencillo, como hemos realizado un CRUD para mostrar las tecnologías del framework Spring Boot, lo mostraremos como un catálogo De requisitos.

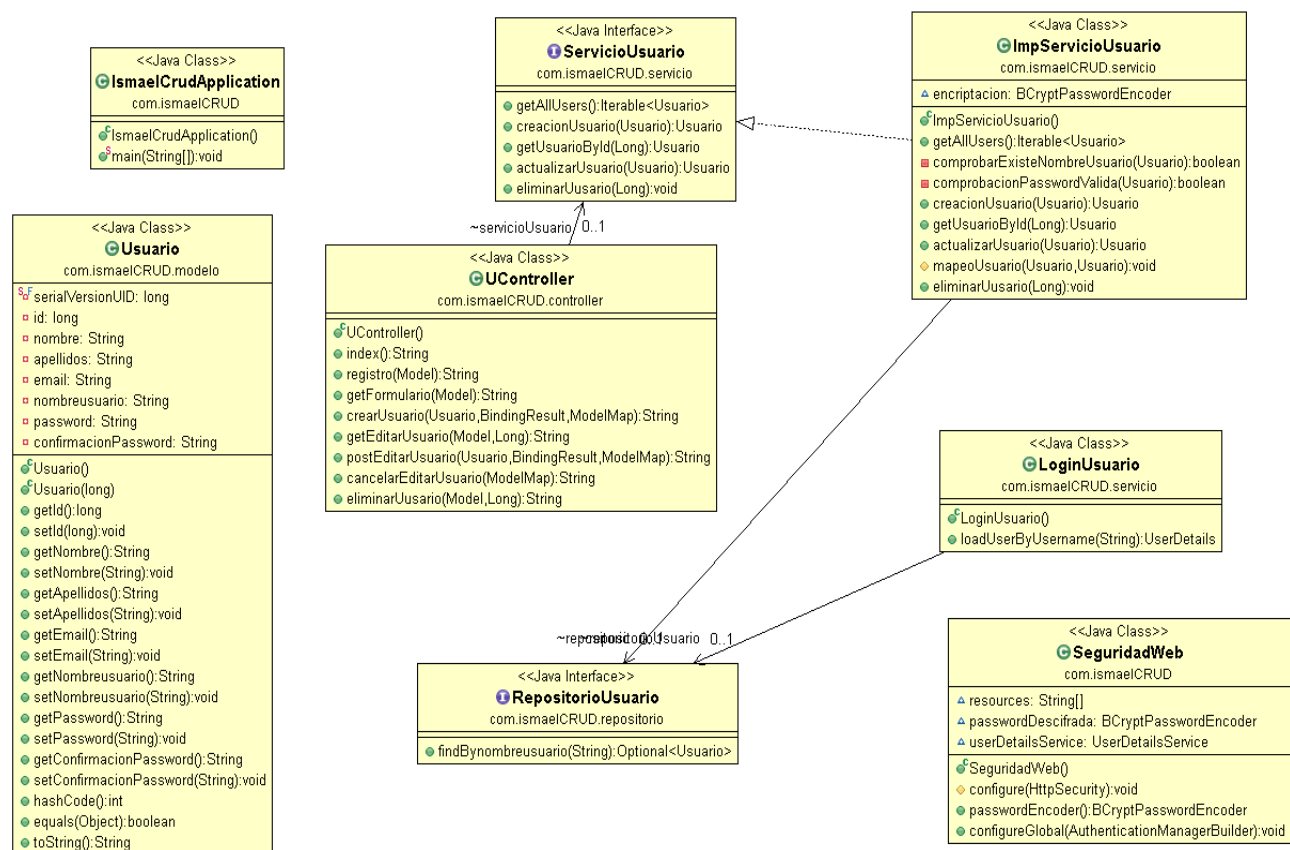
1. Los usuarios pueden registrarse en la aplicación introduciendo los datos que se les piden en el formulario de registro.


	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

2. los usuarios registrados pueden hacer el login a la aplicación utilizando las credenciales que se les pide en el formulario (nombreusuraio y password).
3. los usuarios logeados podrán modificar sus datos (exceptuando el password).
4. los usuarios logeados podrán crear más usuarios.
5. los usuarios logeados podrán borrar usuarios.
6. los usuarios logeados podrán consultar todos los datos de todos los los usuarios registrados.

6.8 Modelo de clases

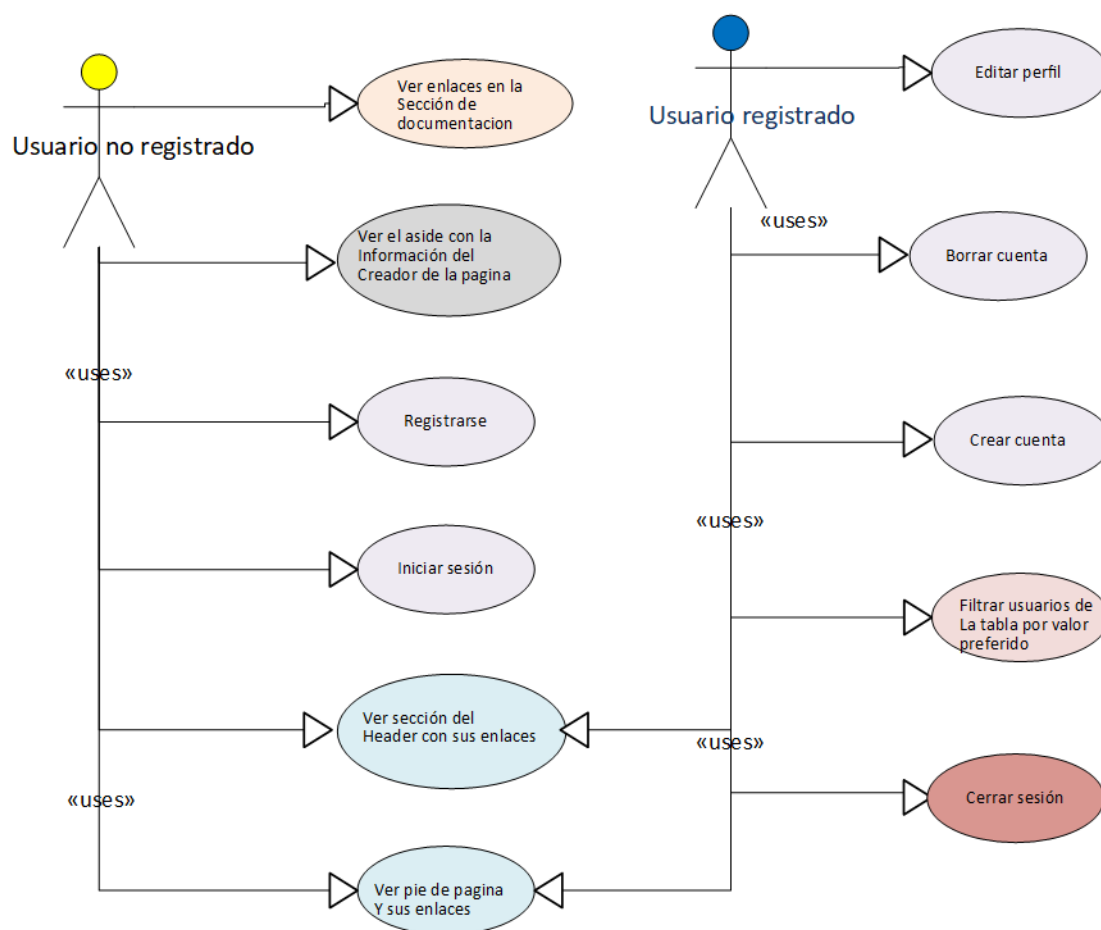
Diagrama de clases del login de nuestro CRUD realizado con spring boot. En él nos muestra todas las **clases e interfaces** con sus **métodos y atributos y la relación entre ellas**.




	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

6.9 Modelo de casos de uso

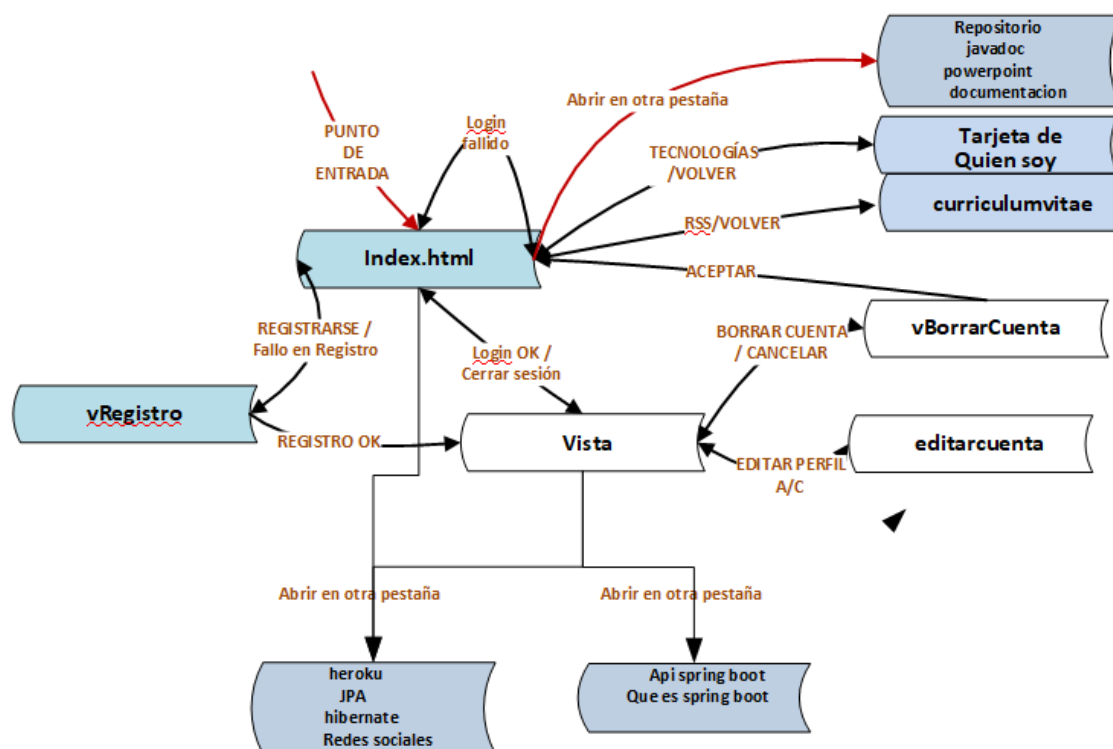
En este modelo veremos lo que puede hacer un usuario registrado y otro no registrado




	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

6.10 Árbol de navegación









El árbol de navegación es muy similar al modelo de casos de uso pero es una visión un poco más grafica de la aplicación




	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

6.11 Modelo físico de datos

El modelo físico de datos en nuestro caso solo cuenta con una tabla usuario, que es donde almacenamos los usuarios y con los datos de ella podemos hacer el login del usuario.

  ismaelcrud usuario
 id : bigint(20)
 apellidos : varchar(50)
 email : varchar(255)
 nombre : varchar(20)
 nombreusuario : varchar(15)
 password : varchar(255)

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

7 Conclusiones y líneas futuras

Una vez ya explicado el proyecto y todas las tecnologías que vamos a utilizar para realizarlo toca el turno para hacer las comparaciones de este proyecto con lo que hemos aprendido en el curso.

La primera y principal diferencia es el lenguaje que vamos a utilizar en nuestra aplicación en el lado del servidor. En el módulo de **DAW** (desarrollo de aplicaciones WEB) el lenguaje de back-end que hemos utilizado es el **PHP**, y nosotros en la aplicación vamos a utilizar **JAVA**.


La principal diferencia de JAVA con PHP es que JAVA es un lenguaje compilado (se compila en la JVM que es la máquina virtual de java) mientras PHP es un lenguaje de scripting, es decir no necesita ser compilado.

Otra de las grandes diferencias es que JAVA tiene un tipado fuerte mientras PHP tiene un tipado débil esto quiere decir que PHP es más flexible y confía en la programación con sentido común, pero puede llevar a complicaciones en la programación orientada a objetos, mientras que java es un lenguaje de programación totalmente orientado a objetos y es más óptimo para este paradigma de programación.

En resumen JAVA se utiliza en **equipos de desarrollo muy grandes** orientado a la creación de aplicaciones empresariales y PHP en entornos más pequeños y para **soluciones más pequeñas** tipo **CMS o blogs**.

Con respecto a los lenguajes de marcas **HTML** y **CSS** varían poco lo único que en nuestros archivos HTML incrustaremos código de la herramienta que nos da Spring Boot, thymeleaf como ya explicamos en un punto anterior.

En el tema bases de datos La base de datos es la misma **MYSQL**, la única diferencia es la conexión a la base de datos, que en primero utilizamos JDBC y en este proyecto utilizaremos JPA.

	CÓDIGO DEL PROYECTO: 1072
	TÍTULO: SPRING BOOT FRAMEWORK
	AUTOR: Ismael Heras Salvador

Como líneas futuras quería añadir que a la hora de hacer este CRUD no he añadido roles para gestionar los usuarios de la aplicación y es una cosa que me ha quedado en el tintero, porque entiendo que un CRUD sin roles no tiene mucho sentido, así que nos marcamos como siguiente reto realizar este CRUD pero añadiéndole la funcionalidad de R

8 Anexos


8.1 Instalación de spring boot

Ahora mostraremos la documentación de la instalación de todo el componente Para poder realizar nuestro proyecto con **Spring Boot**.

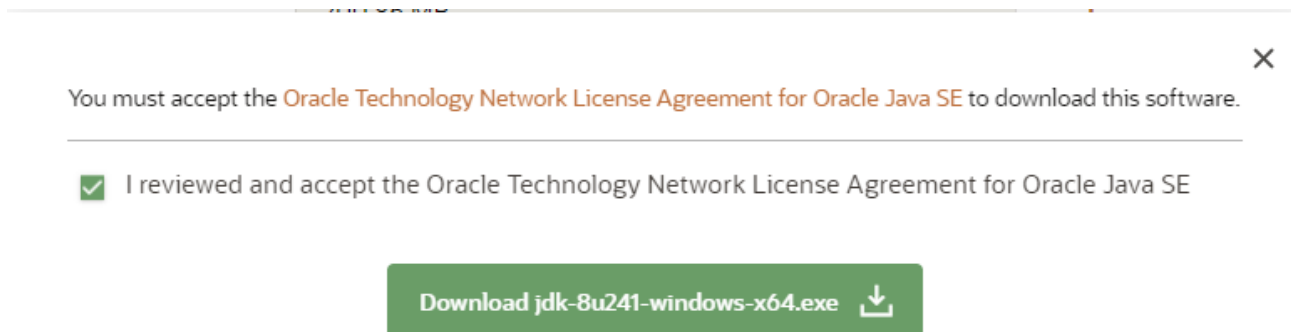
Lo primero es instalar el **JDK** que es una herramienta para convertir nuestro código fuente (.java) en byteCode (.class) el cual será interpretado por la máquina virtual de java **JVM**.

Nos dirigimos a la página de Oracle oficial para descargar el JDK

<https://www.oracle.com/java/technologies/javase-jdk8-downloads.html>

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Instalamos la versión 8 de **JDK**




Como nota cabe destacar que ahora para descargar **JDK** hay que registrarse en Oracle para poder descargarla.

Una vez instalado el **JDK** lo que tendremos que instalar es el **IDE** para el desarrollo de la aplicación que en este caso sería el **eclipse**.

Nos dirigimos a la página oficial de eclipse <https://www.eclipse.org/downloads/>

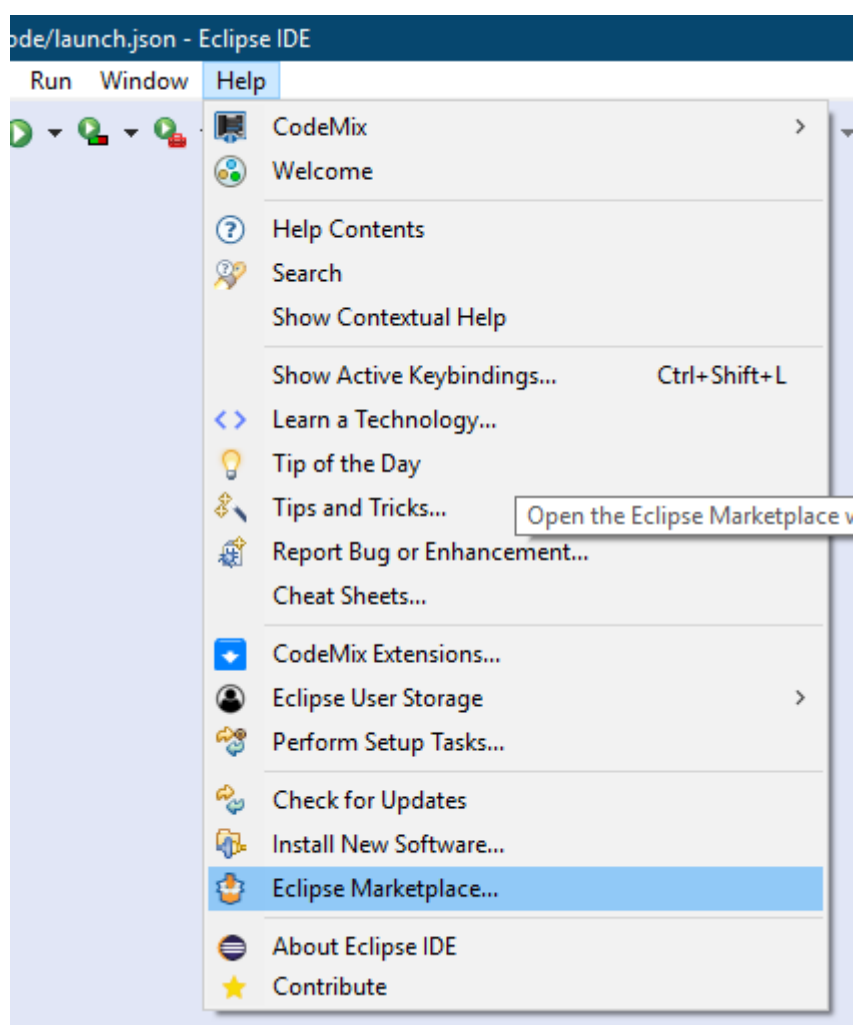
Y nos descargamos la versión de 64 bits para Windows




	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

Una vez instalado el **eclipse** tenemos que instalar el spring suite tools para que nos genere todo el proyecto de Spring Boot.

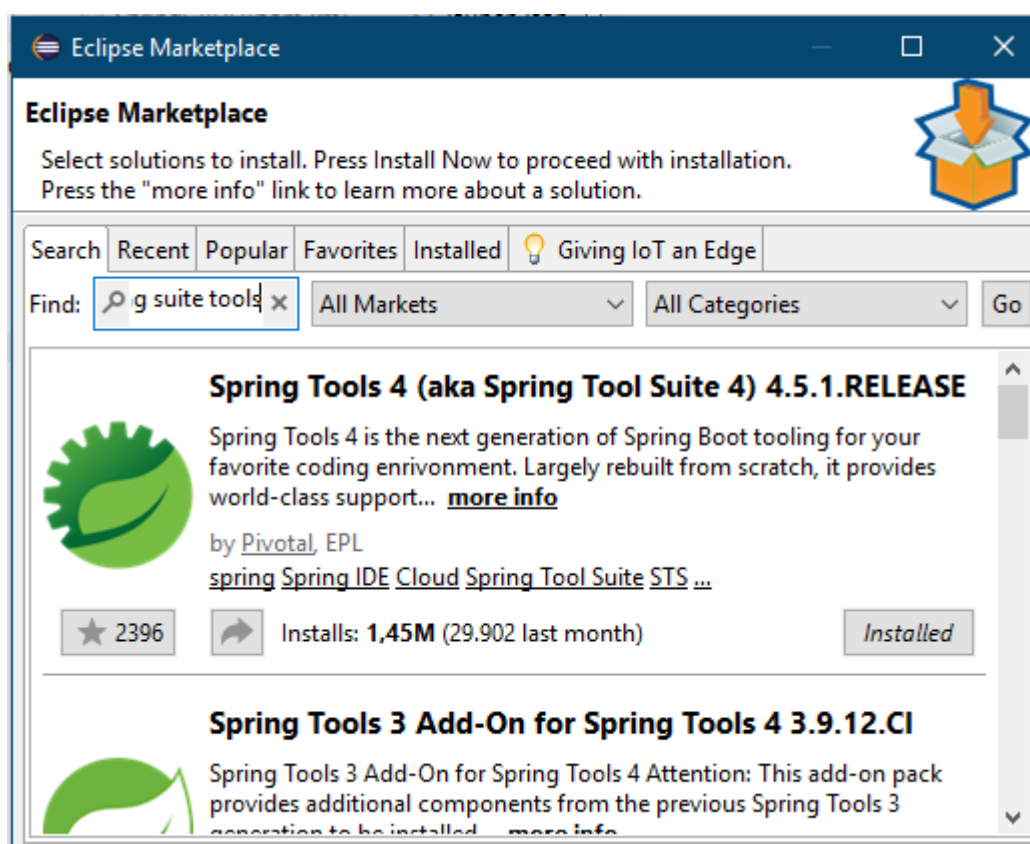
En la ventana de help de eclipse abrimos el eclipse **Marketplace**




	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Una vez en el **Marketplace** buscamos el **Spring Suite Tools**.

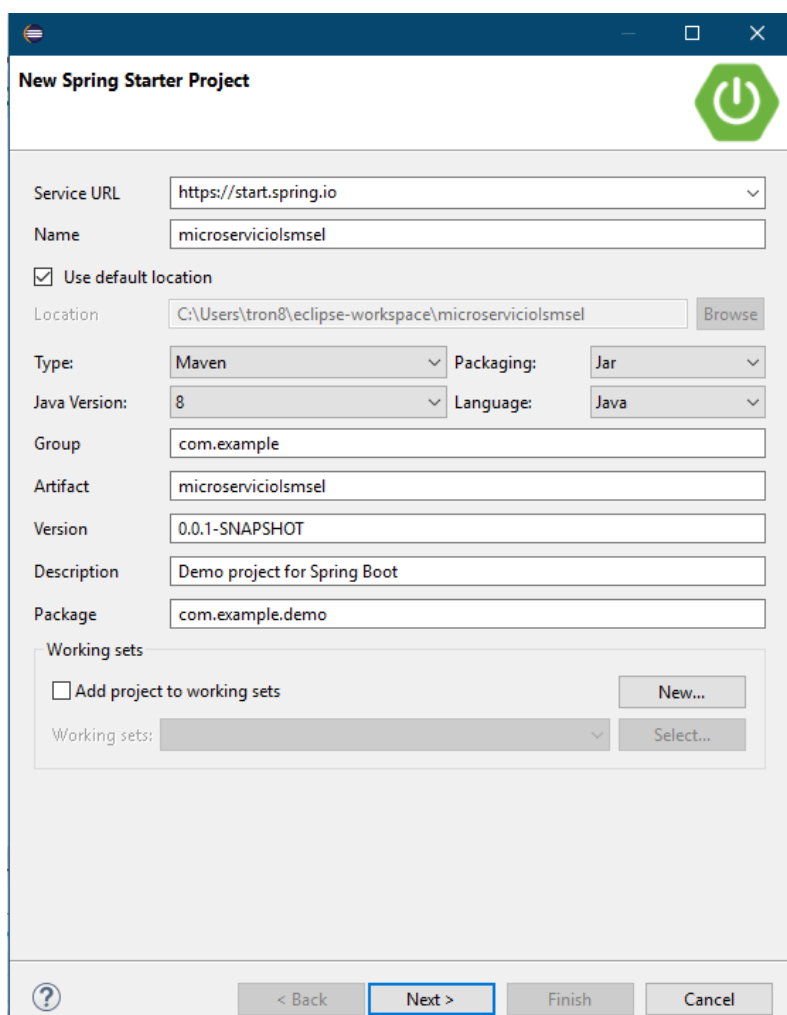
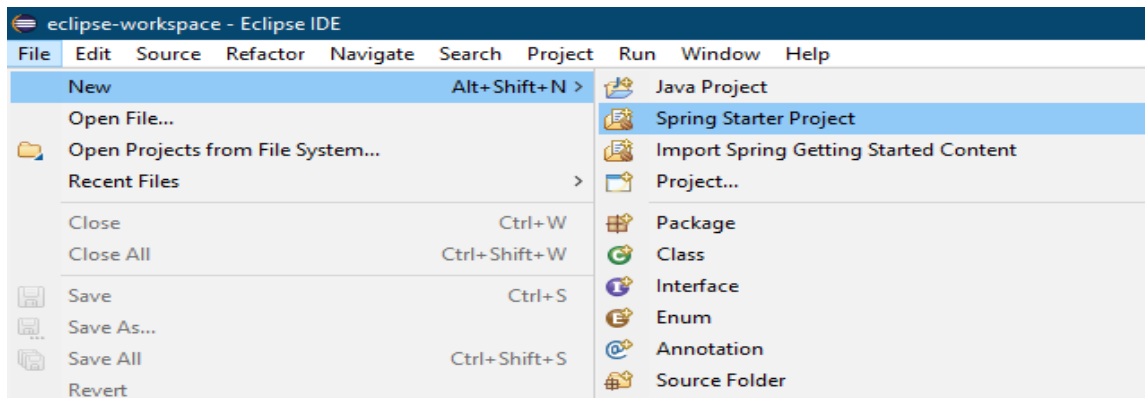
Y lo instalamos.




Ahora ya tenemos instalado todo lo necesario para poder generar nuestro proyecto de Spring Boot y poder realizar la aplicación.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

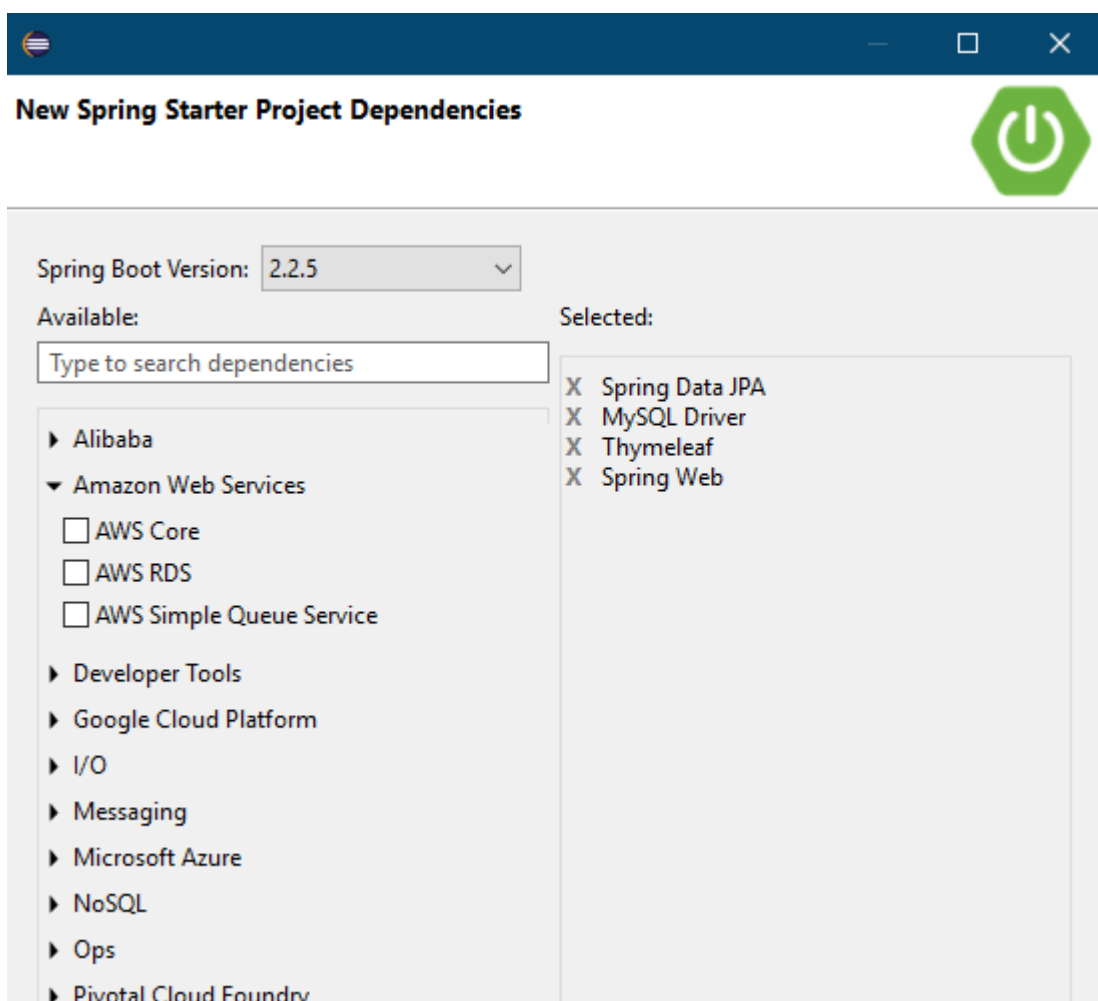
Lo último que queda es **generar el proyecto con elipse**.
En new vamos a spring starter Project para general un nuevo proyecto.




	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	---

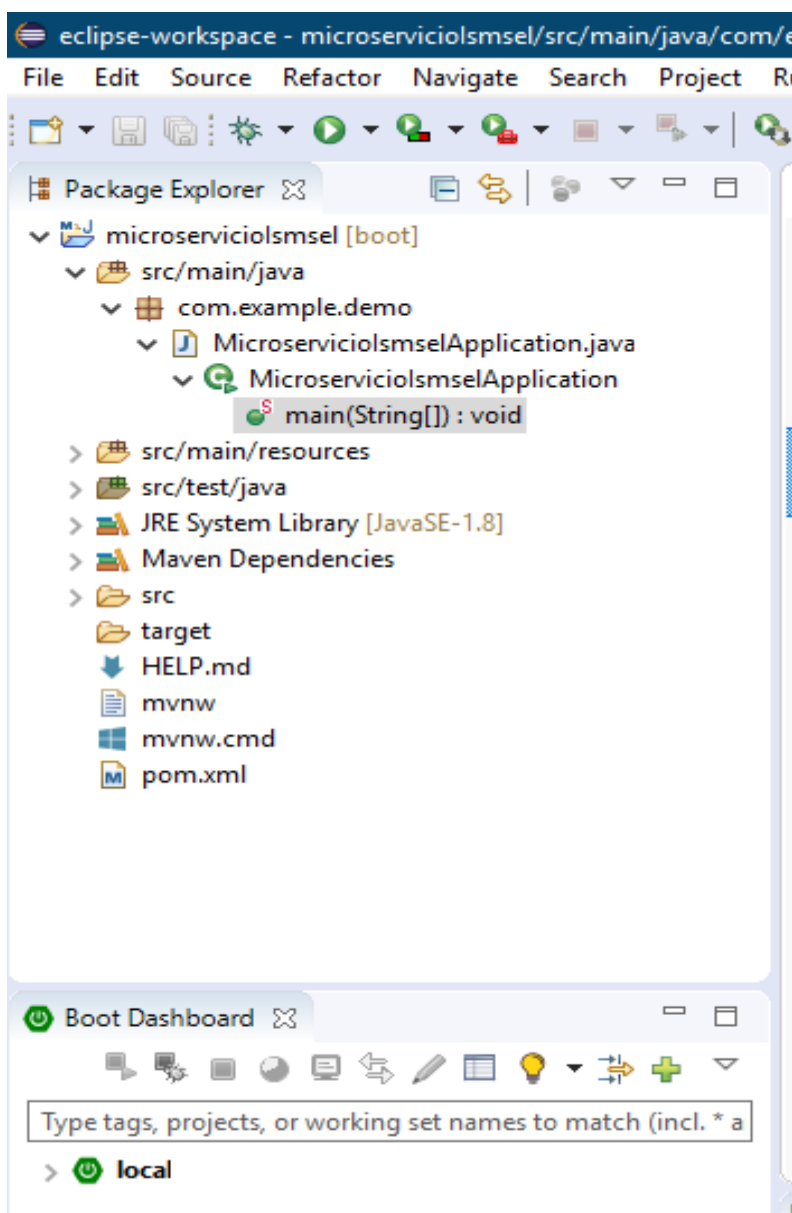
Como vemos en la captura anterior al generar el nuevo proyecto de **Spring Boot** nos sale la pantalla donde elegimos el **packaging** (aunque sea un proyecto web elegiremos jar) el nombre y el nombre de los paquetes.


Y en la siguiente captura es donde elegimos todas las dependencias que necesitamos en nuestro proyecto, como vemos vamos a utilizar **Spring WEB, MYSQL Driver Thymeleaf y Spring Data JPA**. Todos ellos necesarios para la creación de nuestra aplicación.



	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Y con todo esto ya tendríamos preparado nuestro entorno para poder realizar nuestra aplicación (Spring Boot viene con un servidor tomcat embebido), cosa que destriparemos en próximos punto.



	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

8.2 Despliegue de la aplicación en heroku

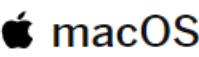
Para que todo el mundo pueda ver nuestra aplicación (ya que no tiene por qué tener el eclipse en casa para poder probarla) hemos desplegado nuestra aplicación en una plataforma llamada Heroku.

Heroku es una plataforma para particulares y empresas donde podemos subir nuestra aplicación, con heroku no nos preocupamos de la infraestructura solo le decimos el lenguaje de back-end que vamos a utilizar y la base de datos y el solo configura nuestro entorno, además podemos conectar nuestro repositorio GitHub con Heroku y cada cambio en el repositorio se refleja en nuestra aplicación en internet.

Explicado ya lo que es esta plataforma vamos a ver como lo hemos desplegado.

Lo primero es instalar git si no lo tenemos instalado, lo necesitaremos para vincular nuestro repositorio con heroku, siga los pasos del manual ya que son muy sencillos he instálelo, [pulsa en este enlace para instalar git](#).

Una vez instalado nos dirigimos a la página oficial de Heroku para instalar su cliente, [instalar cliente de heroku](#).

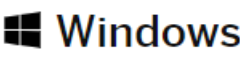


macOS

Download the installer

Also available via Homebrew:

```
$ brew tap heroku/brew && brew install heroku
```




Windows

Download the appropriate installer for your Windows installation:

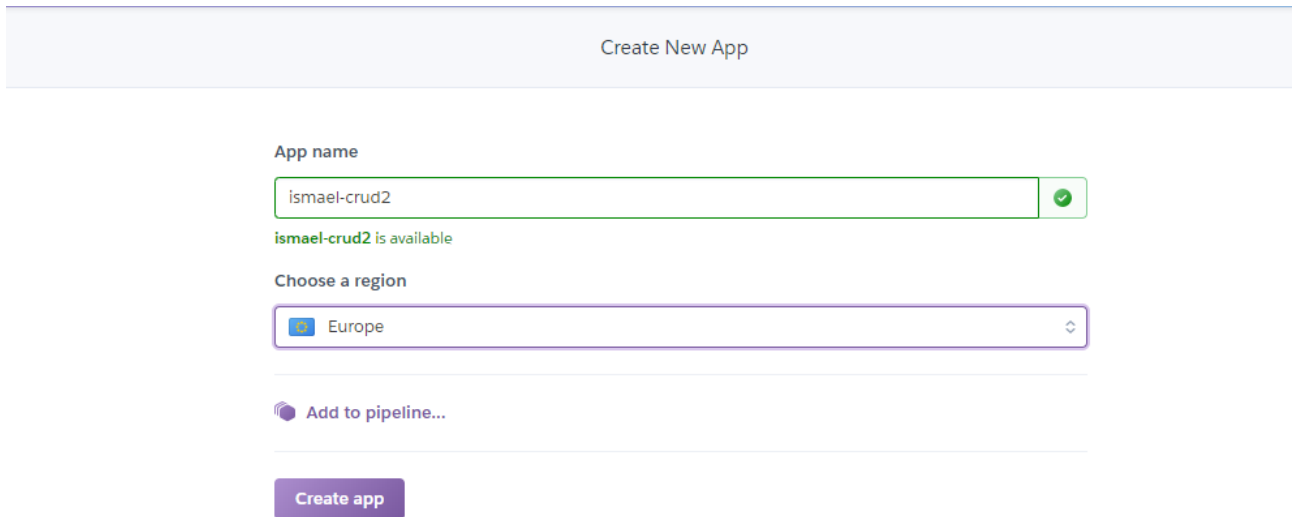
64-bit installer

32-bit installer

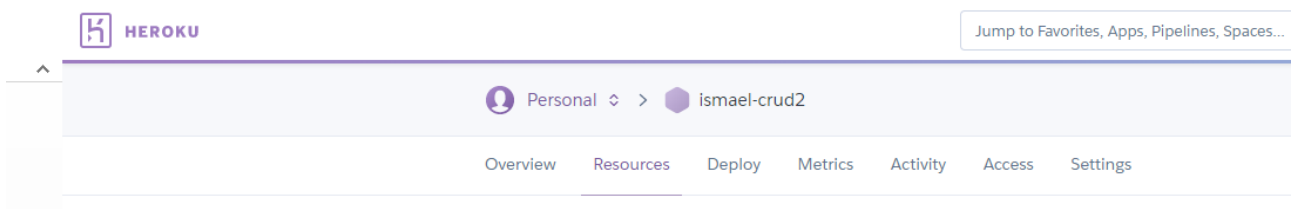
	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--


Con el cliente de Heroku ya instalado nos vamos a la [página oficial de heroku para crear nuestra cuenta.](#)

Luego de crear la cuenta nos logeamos para crear un nuevo proyecto (se entiende que ya tenemos nuestro código en un repositorio remoto, GitHub, Subversión etc...).

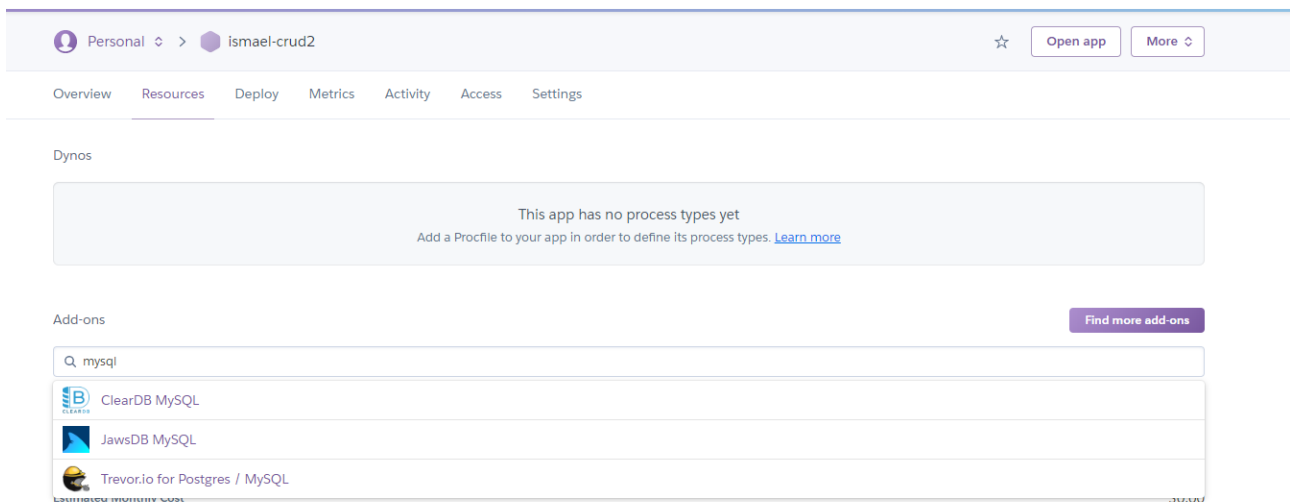


Elegimos el nombre de nuestro proyecto y le damos a crear App, luego nos vamos a la pestaña resources, allí es donde elegiremos la base de datos para nuestra aplicación.

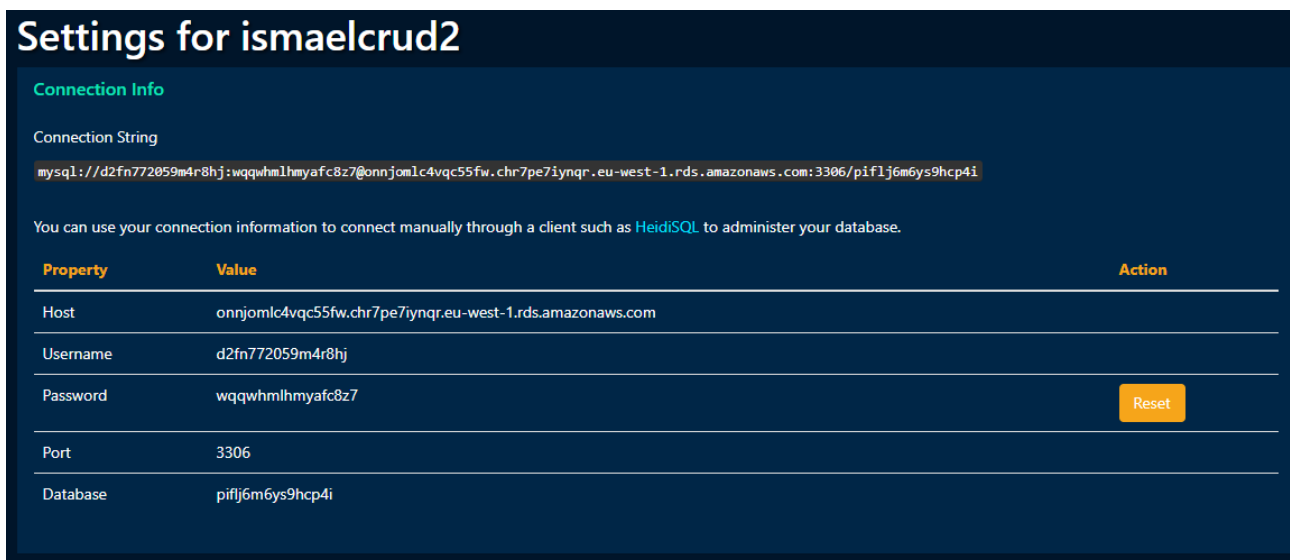


	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--


En el buscador que nos aparece ponemos la base de datos que queremos implementar (en nuestro caso es MYSQL) y de las opciones que tenemos elegimos la versión gratuita (en este apartado vamos a destacar que si queremos tener una base de datos gratuita vamos a tener que dar un número de cuenta, no nos cobran nada pero solo es para verificar la cuenta) yo elegí JawsMySQL pero puedes elegir la que quieras.



Una vez elegida pinchamos en ella y nos mostrara los datos de nuestra conexión que luego deberemos implementar en nuestra aplicación.



Property	Value	Action
Host	onnjomlc4vqc55fw.chr7pe7iynqr.eu-west-1.rds.amazonaws.com	
Username	d2fn772059m4r8hj	
Password	wqqrwhmlhmyafc8z7	<button>Reset</button>
Port	3306	
Database	piflj6m6ys9hcp4i	

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Con los datos de nuestra base de datos de heroku cabíamos el archivo application.properties de nuestra aplicación para que se pueda conectar a ella.


```
properties - Eclipse IDE

index.html application.properties
1 spring.datasource.url=jdbc:mysql://onnjomlc4vqc55fw.chr7pe7iynqr.eu-west-1.rds.amazonaws.com:3306/piflj6m6ys9hpc4i
2 spring.datasource.username= d2fn772059m4r8hj
3 spring.datasource.password=wqqwhmlhmyafc8z7
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.show-sql=true
6 spring.jpa.hibernate.ddl-auto=create
7 spring.datasource.initialization-mode=always
8
9
```

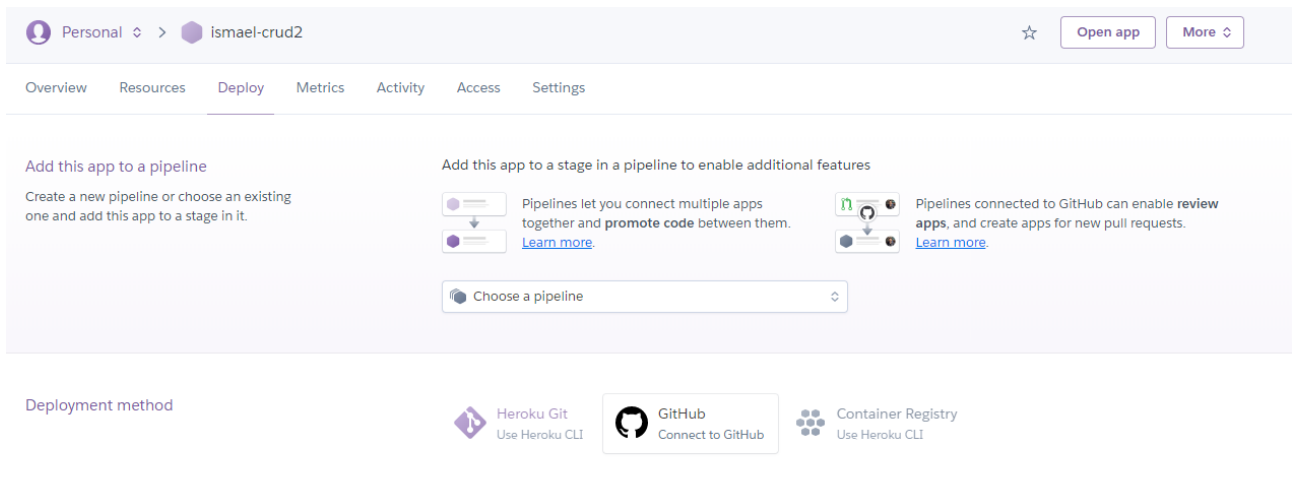
Las tres últimas líneas son para que la aplicación al conectarse por primera vez a la base de datos nos cree las tablas y todas las demás veces que se conecte las actualice.

Luego hay que crear un archivo llamado data.sql con la carga inicial de la base de datos, este archivo lo crearemos en la raíz de resources, junto con el application.properties.

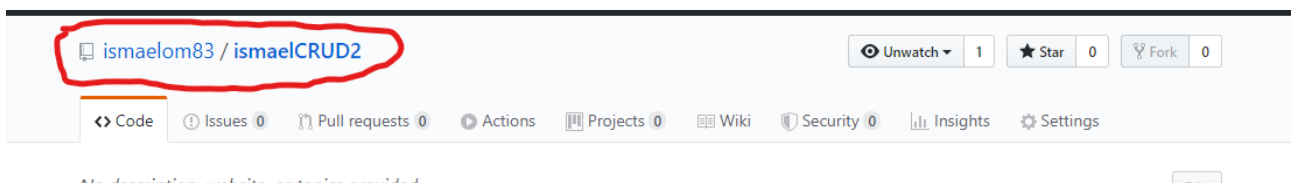
```
index.html application.properties data.sql
ismaelCRUD src main resources data.sql
1
2
3 INSERT INTO `usuario` (`email`, `nombre`, `apellidos`, `password`, `nombreusuario`) VALUES ('ismael@admin.com', 'Ismael', 'Heras',
4 ('jose@admin.com', 'Jose', 'Salvador', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Jose88'),
5 ('alex@admin.com', 'Alex', 'Dominguez', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Alex97'),
6 ('pedro@admin.com', 'Pedro', 'Rodriguez', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Pedro70'),
7 ('david@admin.com', 'David', 'Cuerdo', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'David83'),
8 ('pepe@admin.com', 'Pepe', 'Garcia', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Pepe68'),
9 ('mario@admin.com', 'Mario', 'Salvador', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Mario90'),
10 ('sara@admin.com', 'Sara', 'Sanchez', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Sara93'),
11 ('manuela@admin.com', 'Manuela', 'Perez', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Manuela95'),
12 ('ana@admin.com', 'Ana', 'Hernandez', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Ana87'),
13 ('luis@admin.com', 'Luis', 'Gonzalez', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Luis80'),
14 ('patricia@admin.com', 'Patricia', 'Jambrina', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'Patricia82'),
15 ('noelia@admin.com', 'Noelia', 'Perez', '$2a$12$0LaYBfakNUPstyjwfcxz0zsCZFnmTDGwPIC5EIU0vlyJBtyd1.Q2', 'noelia82');
```

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Una vez ya realizado todo con la base de datos vamos a pasar a la fase donde conectamos nuestro repositorio con heroku.
Nos vamos a la opción que dice deploy y seleccionamos la conexión con GitHub.




El siguiente paso es poner las credenciales de nuestro repositorio donde nos lo solicita heroku.
El usuario y el nombre de nuestro repositorio de GitHub



Y donde lo introducimos en herok

Search for a repository to connect to

Missing a GitHub organization? [Ensure Heroku Dashboard has team access.](#)

	<p>CÓDIGO DEL PROYECTO: 1072</p> <p>TÍTULO: SPRING BOOT FRAMEWORK</p> <p>AUTOR: Ismael Heras Salvador</p>
---	--

Le damos a conectar y luego podríamos elegir la rama que queremos conectar y si lo queremos hacer manualmente o automáticamente (la idea es conectar la master y al ir haciendo merges desde nuestras otras ramas a la master se actualice automáticamente).


Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically;** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more](#).

Choose a branch to deploy

 master

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys


Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

 master

Deploy Branch

Yo lo hago manualmente y la doy a deploy y cuando acabe de subir la aplicación nos muestra un enlace a la URL que crea con el nombre de nuestra aplicación y la extensión herokuapp.com (esto es porque es gratuita si quisieras crear un dominio propio tendrías que pagar) y ya tendríamos nuestra aplicación desplegado con persistencia de datos y lista para que la vea todo el mundo desde cualquier sitio con conexión a internet. [Ver mi aplicación](#)

Receive code from GitHub



Build **master** 804f7f98



Release phase



Deploy to Heroku



Your app was successfully deployed.

 **View**