

# **RECOPILACIÓN SPRING BOOT**

VERSIÓN 1

# Índice

¿Qué es Spring Boot? .....	3
Maven .....	4
¿Qué es el archivo pom.xml? .....	5
Java Persistence API (JPA).....	5
JPA, Hibernate VS JDBC.....	6
Spring Security .....	8
Oauth 2 .....	9
¿Qué es H2? .....	12
Configurar una base de datos H2 con Spring Boot.....	12
Configurar conexión de Spring Boot con H2 .....	13
Acceder a la interfaz gráfica de H2 .....	14
Problemas y solución con H2 .....	15
Bibliografía: .....	16

## ¿Qué es Spring Boot?

Spring Boot es un framework desarrollado para el trabajo con Java como lenguaje de programación. Se trata de un entorno de desarrollo de código abierto y gratuito. Spring Boot cuenta con una serie de características que han incrementado su popularidad y, en consecuencia, su uso por parte de los desarrolladores back-end. Estas son las características más destacadas de este framework:

- Permite crear todo tipo de aplicaciones en el lado del back-end de forma independiente.
- Facilita el trabajo con otras herramientas como Tomcat, Jetty o Undertow. Lo hace directamente, sin necesidad de implementar archivos específicos para ello.
- Simplifica las dependencias para mejorar la configuración final del proyecto que se desarrolla con Spring Boot.
- Se trata de un framework que se configura de manera simple y es compatible con bibliotecas de terceros.
- Facilita la creación de listas, controles de estado y mejora la configuración externa para el desarrollo de aplicaciones.
- No es necesario generar código para los aspectos que controla Spring Boot y no hay requisitos para la configuración XML.

Gracias a estas características, Spring Boot facilita la creación de todo tipo de aplicaciones basadas en él de manera independiente con el mínimo esfuerzo por parte de los desarrolladores. Y es que se trata de una tecnología que facilita que los desarrolladores se centren solo en la parte de programación, sin necesidad de preocuparse por aspectos como la arquitectura. [1]

## Maven

Apache Maven es una potente herramienta de gestión de proyectos que se utiliza para gestión de dependencias, como herramienta de compilación e incluso como herramienta de documentación. Es de código abierto y gratuita.

Aunque se puede utilizar con diversos lenguajes (C#, Ruby, Scala...) se usa principalmente en proyectos Java, donde es muy común ya que está escrita en este lenguaje. De hecho, frameworks Java como Spring y Spring Boot la utilizan por defecto, por lo que conviene conocerla si trabajas en la plataforma Java y, desde luego, con Spring.

Al contrario que otras herramientas anteriores y más limitadas como Apache Ant (también muy popular), Maven utiliza convenciones sobre dónde colocar ciertos archivos para el proceso de build de un proyecto, por lo que solo debemos establecer las excepciones y por lo tanto simplifica mucho el trabajo. Además, es una herramienta declarativa. Es decir, todo lo que definamos (dependencias en módulos y componentes externos, procesos, orden de compilación, plugins del propio Maven...) se almacena en un archivo XML que Maven lee a la hora de funcionar. Otras alternativas, como Gradle no utilizan archivos XML, sino de otro tipo, pero usan los mismos conceptos de Maven.

Con Maven se puede:

Gestionar las dependencias del proyecto, para descargar e instalar módulos, paquetes y herramientas que sean necesarios para el mismo.

Compilar el código fuente de la aplicación de manera automática.

Empaquetar el código en archivos .jar o .zip.

Instalar los paquetes en un repositorio (local, remoto o en el central de la empresa)

Generar documentación a partir del código fuente.

Gestionar las distintas fases del ciclo de vida de las build: validación, generación de código fuente, procesamiento, generación de recursos, compilación, ejecución de test ...

Además, la mayor parte de los entornos de desarrollo y editores de Java disponen de plugins específicos o soporte directo de Maven para facilitarnos el trabajo con esta, puesto que se ha convertido en una herramienta casi universal. [2]

## ¿Qué es el archivo pom.xml?

La unidad básica de trabajo en Maven es el llamado Modelo de Objetos de Proyecto conocido simplemente como POM (de sus siglas en inglés: Project Object Model).

Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, etc...

El hecho de utilizar un archivo XML revela su edad. Maven se creó en 2002, cuando XML era lo más. Si se hubiese creado unos pocos años después seguramente tendríamos un pom.json y si hubiese sido más reciente un pom.yml. Modas que vienen y van. No obstante, el formato XML, aunque engorroso, es muy útil a la hora de definir con mucho detalle cómo debe ser cada propiedad y, también, para poder comprobarlas.

Incluso, aunque nuestro proyecto, que usa Maven, tenga un archivo pom.xml sin opciones propias, prácticamente vacío, estará usando el modelo de objetos para definir los valores por defecto del mismo. Por ejemplo, por defecto, el directorio donde está el código fuente es src/main/java, donde se compila el proyecto es target y donde ubicamos los test unitarios es en src/main/test, etc... Al pom.xml global, con los valores predeterminados se le llama Súper POM. [2]

## Java Persistence API (JPA)

JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB).

Cuando empezamos a trabajar con bases de datos en Java lo primero que nos enseñan es a utilizar el API de JDBC el cual nos permite realizar consultas directas a la base de datos a través de consultas SQL nativas. JDBC por mucho tiempo fue la única forma de interactuar con las bases de datos, pero representaba un gran problema y es que Java es un lenguaje orientado a objetos y se tenía que convertir los atributos de las clases en una consulta SQL como SELECT, INSERT, UPDATE, DELETE, etc. lo que ocasionaba un gran esfuerzo de trabajo y un provocaba muchos errores en tiempo de ejecución, debido principalmente a que las consultas SQL se tenían que generar frecuentemente al vuelo.

Una de las cosas más importantes para comprender que es JPA es entender que JPA es una especificación y no un Framework como tal, ¿pero quiere decir esto exactamente?, pues bien, una especificación no es más que un documento en el cual se plasman las reglas que debe de cumplir cualquier proveedor que dese desarrollar una implementación de JPA, de tal forma que cualquier persona puede tomar la especificación y desarrollar su propia implementación de JPA,

¿Esto quiere decir que pueden existir muchas implementaciones de JPA? la respuesta es sí, de echo en la actualidad existen varios proveedor como lo son los siguientes:

- [Hibernate](#)
- [ObjectDB](#)
- [TopLink](#)
- [EclipseLink](#)
- [OpenJPA](#)

Dentro de las implementaciones más utilizadas están Hibernate, EclipseLink & TopLink, las dos primeras son las más utilizadas en el mundo open source y TopLink es muy utilizada en desarrollos y productos relacionados con Oracle. Antes de preguntarnos cuál es la diferencia entre todas estas implementaciones tenemos que comprender que en teoría todas debería de ofrecer la misma funcionalidad y el mismo comportamiento, lo que nos permitiría migrar entre una implementación a otra sin afectar en nada nuestra aplicación. Desde luego esto es solo teoría, ya que en la actualidad no todas las implementaciones implementan al 100% la especificación de JPA, además en escenario muy concretos puede que se comporten ligeramente diferente, por lo que puede requerir realizar algunos ajustes antes de migrar correctamente de proveedor. [3]

## JPA, Hibernate VS JDBC

Los que ya han tenido la oportunidad de trabajar con algún ORM como JPA o Hibernate sabrán las bondades que tiene ya que nos permite desarrollar de una forma mucho más rápida y con muchos menos errores en tiempo de ejecución ya que nos permite modelar nuestras entidades como Clases Java las cuales serán convertidas a las instrucciones Insert, Update o Select según sea la operación a realizar. Claro que todos estos beneficios tienen un costo y es que el performance se degrada debido a todas las conversiones que se tiene que hacer para convertir las Entity en Querys y los ResultSet pasarlos a clases además que cada registro representa un Objeto en memoria que tendrá que ser administrado por nuestro servidor.

Por otro lado, tenemos a JDBC que nos brinda total libertad de hacer lo que queramos sin ningún tipo de limitación explotando al máximo las características de la base de datos. JDBC nos permite realizar consultas nativas para cada base de datos lo que ayuda mucho a la velocidad de respuesta y los resultados son devueltos en un ResultSet los cuales podemos extraer solamente los datos que requerimos y no toda la Entity como en el caso de JPA o Hibernate.

### Ventajas

Nos permite desarrollar mucho más rápido.

Permite trabajar con la base de datos por medio de entidades en vez de Querys.

Nos ofrece un paradigma 100% orientado a objetos.

Elimina errores en tiempo de ejecución.

Mejora el mantenimiento del software.

#### Desventajas

No ofrece toda la funcionalidad que ofrecería tirar consultas nativas.

El performance es mucho más bajo que realizar las consultas por JDBC.

Puede representar una curva de aprendizaje más grande.

#### JDBC

##### Ventajas

Ofrece un performance superior ya que es la forma más directa de mandar instrucciones a la base de datos.

Permite explotar al máximo las funcionalidades de la base de datos.

##### Desventajas

El mantenimiento es mucho más costoso.

Introduce muchos errores en tiempo de ejecución.

El desarrollo es mucho más lento.

Probablemente pensaras que JPA o Hibernate son mejores ya que ofrece mayor ventaja que desventajas sin embargo las desventajas que tiene son muy serias y pueden ser cruciales a la hora de decidir que tecnología utilizar ya que si tenemos una aplicación muy buena y fácil de mantener pero que tarda demasiado para consultar datos puede llegar a ser algo muy malo.

Como conclusión yo diría que si requieres una aplicación donde el rendimiento sea el factor más importante utilices JDBC, pero por otra parte si el rendimiento es algo que no está importante puedes utilizar JPA o Hibernate. [4]

## Spring Security

Spring Security es un framework de seguridad. Es un proyecto que se viene a integrar con Spring Framework, ofrece todo lo que es autenticación de usuario a través de credenciales Username y password, también todo lo que es autorización por medio de una lista de control de acceso la cual es una forma de determinar los permisos de accesos apropiados a un determinado recurso.

Por lo tanto nuestro recurso va a ser el backend, nuestro API REST, controladores, también pueden ser la clase de servicios, lógica de negocio; al usuario se le da acceso a estos recursos a través de roles.

Un rol es un grupo o tipo de usuario que se le otorga ciertos privilegios para llevar a cabo una o varias acciones dentro de la aplicación. Consta de un nombre por ejemplo como admin o usuario, puede concederse ya sea a los usuarios o incluso a otros roles.

Se utilizan para crear grupos lógicos de usuarios para la asignación adecuada de los privilegios en la aplicación. [5]



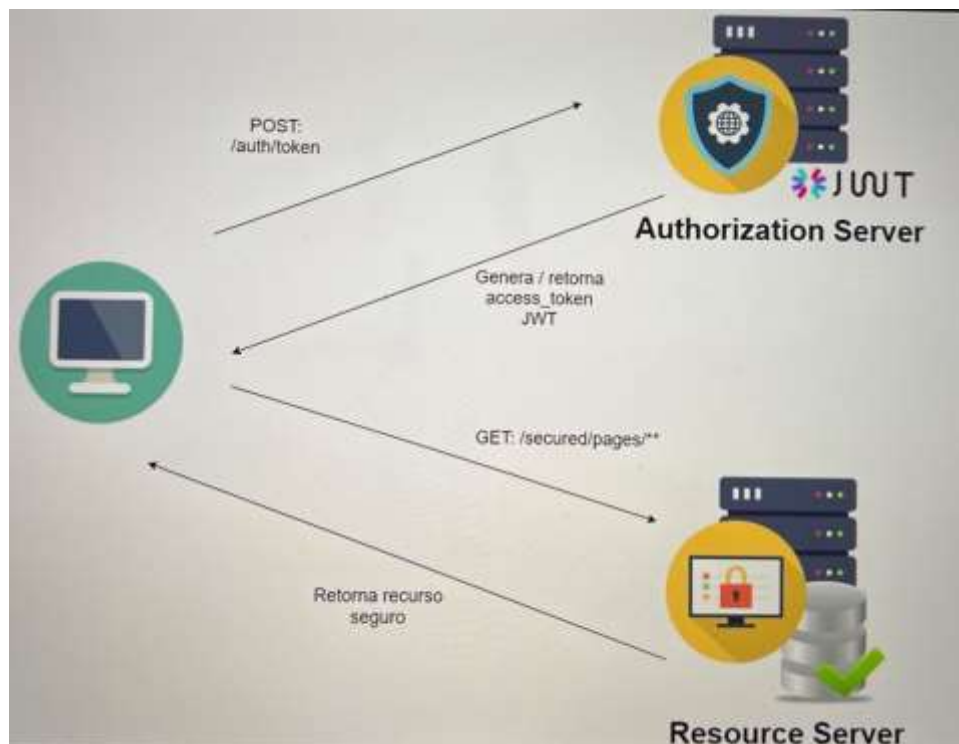


## Oauth 2

Oauth 2 de Spring es un framework y protocolo estándar de autorización que permite a las aplicaciones de terceros ya sea Angular, React, Android autenticar con cuentas de usuario que están en otra aplicación en un servicio HTTP externo para compartir un sistema de autenticación de una aplicación a otra sin compartir toda la información del usuario ni menos las credenciales.

Este mecanismo es utilizado por compañías como Google, Facebook, Twitter y GitHub para permitir a los usuarios compartir información sobre sus cuentas de forma limitada con aplicaciones de terceros. Como se había mencionado independiente si es aplicaciones móviles o aplicaciones web ya sea con Angular, React, Android o con cualquier tecnología. [5]

La implementación de Spring Security Oauth 2 se compone de dos partes:



Primero se tiene el servidor de autorización el Authorization Server, por un lado, el Resource Server o Servidor de recursos.

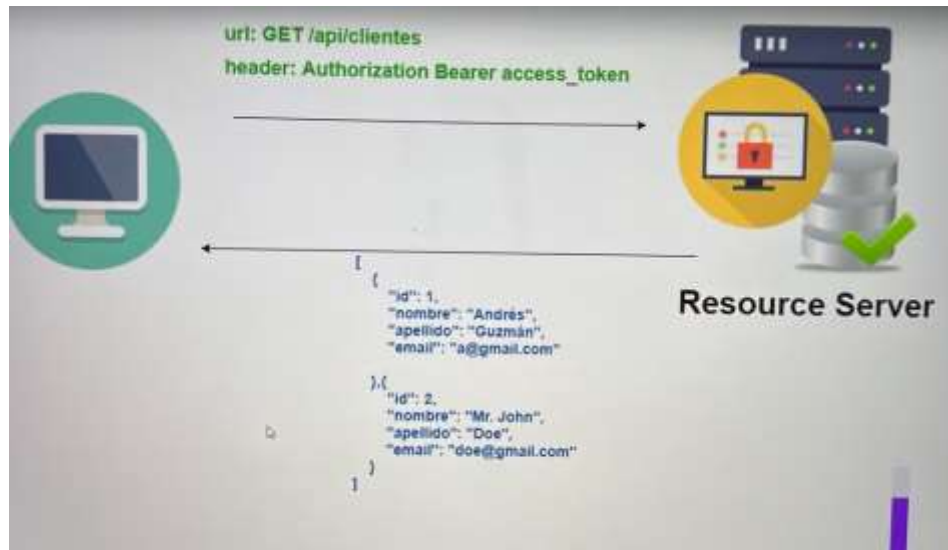
El primero el Authotization se encarga de realizar la autenticación del usuario y según si es válida genera un token de acceso y le provee el JWT al usuario, para que después el usuario a través de este token pueda acceder a los distintos recursos de la aplicación.

Por otro lado, se tiene el servidor de recursos que se encarga de administrar los permisos y accesos hacia las páginas, API Rest y los endpoints que están con seguridad.

Por debajo Resource Server habilita un filtro de Spring OAuth 2 authentication processing filter el cual se utiliza para validar y dar acceso a una petición HTTP o request que viene desde el cliente, Este request es el que envía el token de acceso en las cabeceras y si es válido, el servidor de recursos otorga el permiso para que el cliente pueda acceder a los recursos de estas páginas o endpoints del backend, básicamente nuestras API REST.



Primero se tiene una ruta Wireless del tipo Post auth/token, esta ruta es proveída Spring y la cual permite realizar el login la autenticación y si todo sale bien va a generar el token, se tienen que enviar algunos parámetros en la ruta de auth/token, básicamente en las cabeceras header el parámetro authorization que contiene el Client\_id y Client\_secret que son parámetros del api OAuth 2 el cual se debe de concatenar con los dos puntos y encriptar en base64 una autorización del tipo Basic y además el Content\_type del tipo form\_urlencoded y en el cuerpo del request se tienen que enviar las credenciales (Username, Password y grant\_Type de tipo password), entonces se envían estos datos o parámetros al servidor, el cual valida si todo sale bien, si es así retornara el token de acceso con algunos parámetros más. [5]



El servidor de recurso si se quiere acceder a una página protegida de nuestro backend de nuestra aplicación, por ejemplo la ruta del tipo GET api/clientes y en las cabeceras se debe enviar el el atributo authorization del tipo Bearer con el token de acceso el cual es el que se recibe cuando se inicia sesión en el Authorization Server, si el token es válido otorga el acceso y los permisos para acceder al recurso y retornara el contenido en formato json (JavaScript Object Notation) ahora el siguiente paso es agregar las dependencias en nuestro proyecto. [5]

Lo primero que se debe de hacer es instalar y configurar la dependencia en el pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework.security.oauth/spring-security-oauth2 -->
```

```
<dependency>
```

```
<groupId>org.springframework.security.oauth</groupId>
```

```
<artifactId>spring-security-oauth2</artifactId>
```

```
<version>2.3.4.RELEASE</version>
```

```
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-jwt -->
```

```
<dependency>
```

```
<groupId>org.springframework.security</groupId>
```

```
<artifactId>spring-security-jwt</artifactId>
```

```
<version>1.0.9.RELEASE</version>
```

```
</dependency>
```

## ¿Qué es H2?

H2 es una base de datos relacional que se encuentra escrita en Java y funciona como una base de datos en memoria, cuyo uno de sus puntos fuertes es que puede trabajar como una base de datos embebida en aplicaciones Java o ejecutarse en modo cliente servidor. Es decir, que añadida en nuestras aplicaciones como una dependencia más y una vez configurada la conexión, nos va a permitir realizar pruebas y trabajar como si fuera una base de datos relacional. Pero hay que tener en cuenta que es en memoria, por lo que nunca debería ser usada para entornos productivos. [6]

## Configurar una base de datos H2 con Spring Boot

Para poder hacer uso de una base de datos H2 en Spring Boot, va a ser necesario añadir las dependencias de H2 a nuestro proyecto.

Vamos a añadir la única dependencia maven que hace falta para incorporar H2 como base de datos embebida:

```
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
</dependency>
```

Por lo general esta base de datos se usará con test por lo que es mejor añadir ese scope:

```
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<scope>test</scope>
</dependency>
```

Y no te olvides de añadir en tu pom.xml la dependencia de spring data para realizar la autoconfiguración y conexión con la base de datos, lo cual nos facilitará las conexiones con las bases de datos gracias a Hibernate.

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

Una vez hemos añadido la dependencia maven necesaria habrá que realizar la configuración en el fichero application.properties o application.yaml para poder conectarnos con nuestra base de datos.

### Configurar conexión de Spring Boot con H2

Para establecer la conexión con nuestra base de datos H2, es necesario indicarle a Spring Boot como conectarse, para ello lo hacemos a través del fichero application.yaml o application.properties.

Por defecto, Spring Boot nos va a configurar nuestra aplicación para conectarnos a nuestra base de datos en memoria H2 con username sa y la password vacía.

Aunque podemos parametrizar nuestra aplicación con los siguientes parámetros:

```
spring.datasource.url: jdbc:h2:mem:testdb  
spring.datasource.driverClassName: org.h2.Driver  
spring.datasource.username: sa  
spring.datasource.password: password  
spring.jpa.database-platform: org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled: true
```

Hay que tener en cuenta que estamos usando una base de datos en memoria, por lo que cuando la aplicación finalice, la base de datos se borrará. Por lo que para poder persistir los datos de manera física vamos a añadir el siguiente parámetro:

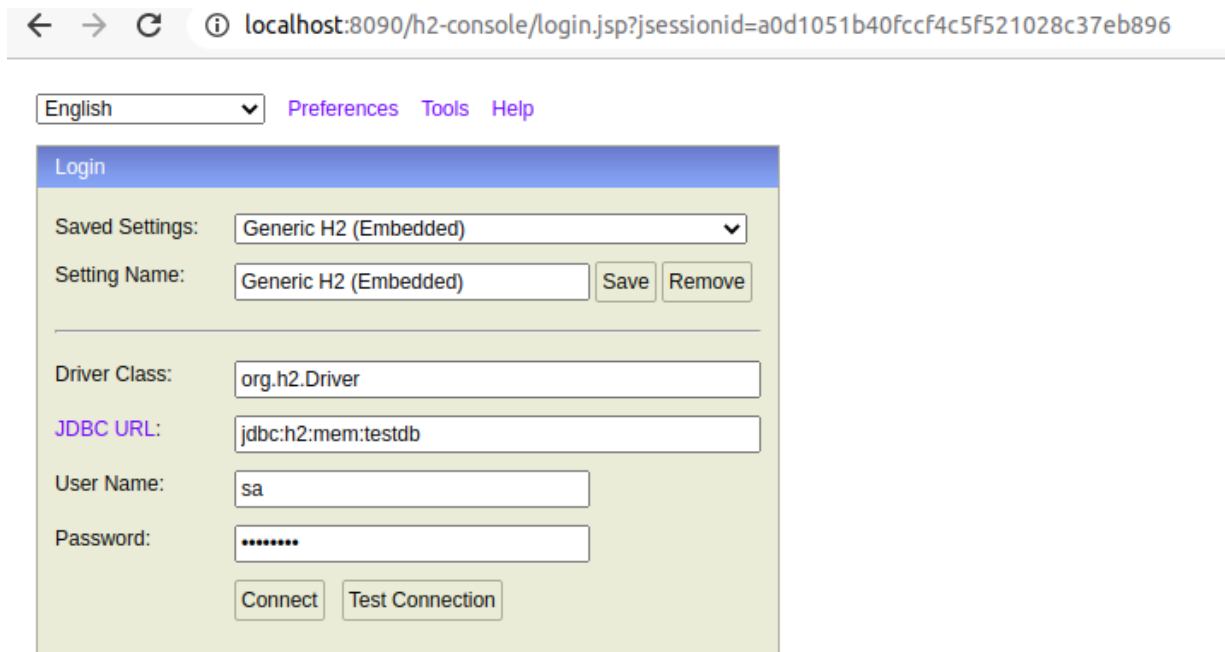
```
spring.datasource.url=jdbc:h2:file:/data/refactorizando/h2
```

## Acceder a la interfaz gráfica de H2

La Base de Datos H2 tiene una interfaz gráfica a la que se accede a través del navegador, que funcionará como un gestor de base de datos, lo que nos permitirá ver tablas, roles y realizar queries. La consola de H2 no se encuentra activada por defecto en Spring, por lo que vamos a añadir el siguiente comando por configuración:

```
spring.h2.console.enabled=true
```

Una vez arranca la aplicación podemos acceder a la consola a través de este endpoint <http://localhost:8080/h2-console>.



The screenshot shows a web browser window with the address bar displaying `localhost:8090/h2-console/login.jsp?jsessionid=a0d1051b40fccf4c5f521028c37eb896`. The page has a navigation bar with a language dropdown set to "English" and links for "Preferences", "Tools", and "Help". The main content area is titled "Login" and contains the following fields and buttons:

- Saved Settings:** A dropdown menu currently showing "Generic H2 (Embedded)".
- Setting Name:** A text input field containing "Generic H2 (Embedded)", with "Save" and "Remove" buttons to its right.
- Driver Class:** A text input field containing "org.h2.Driver".
- JDBC URL:** A text input field containing "jdbc:h2:mem:testdb".
- User Name:** A text input field containing "sa".
- Password:** A text input field with masked characters (dots).
- Buttons:** "Connect" and "Test Connection" buttons at the bottom.

Login de base de datos H2

The screenshot shows the IntelliJ IDEA interface with the SQL editor open. The top toolbar includes buttons for running queries, auto-completing, and clearing SQL statements. The left sidebar shows a project tree with a database connection named 'H2 1.4.200 (2019-10-24)'. The main editor area displays the 'Important Commands' section, which lists various SQL commands and their corresponding keyboard shortcuts. Below this, there is a 'Sample SQL Script' section with a list of commands and their descriptions. The bottom section is titled 'Adding Database Drivers' and provides instructions on how to add database drivers to the environment variables.

**Important Commands**

Icon	Description
	Displays the Help Page
	Shows the Command History
	Executes the current SQL statement
	Executes the SQL statement defined by the text selection
	Auto-complete
	Disconnects from the database

**Sample SQL Script**

Command	Description
SHOW TABLES	SHOW TABLES IF EXISTS TEST;
CREATE TABLE	CREATE TABLE TEST (ID INT PRIMARY KEY);
ALTER TABLE	ALTER TABLE TEST ADD COLUMN NAME VARCHAR(255);
INSERT INTO	INSERT INTO TEST VALUES (1, 'John');
SELECT	SELECT * FROM TEST ORDER BY ID;
UPDATE	UPDATE TEST SET NAME='Jane' WHERE ID=1;
DELETE	DELETE FROM TEST WHERE ID=1;
HELP	HELP

**Adding Database Drivers**

Additional database drivers can be registered by adding the file location of the driver to the environment variable `jdbcDrivers` in `CLASSPATH`. Example (Windows): to add the database driver located at `C:\Programs\mysql\bin\mysql-connector-j-8.0.29.jar`, set the environment variable `jdbcDrivers` to `C:\Programs\mysql\bin\mysql-connector-j-8.0.29.jar`.

Una vez nos hemos conectado podemos comenzar a realizar queries.

```
spring.h2.console.path=/h2-console
```

Uno de los problemas más típicos que nos puede ocurrir cuando hacemos uso de H2 es no haber definido el dialecto y tendríamos un error como el siguiente:

Para solucionar este error hay que asegurarse de haber metido la siguiente línea:

```
properties.hibernate.dialect: org.hibernate.dialect.H2Dialect
```

## Bibliografía:

- [1] <https://www.tokioschool.com/noticias/spring-boot/#:~:text=Spring%20Boot%20es%20un%20framework,de%20c%C3%B3digo%20abierto%20y%20gratuito.>
- [2] <https://www.campusmvp.es/recursos/post/java-que-es-maven-que-es-el-archivo-pom-xml.aspx>
- [3] <https://www.oscarblancarteblog.com/tutoriales/java-persistence-api-jpa/>
- [4] <https://www.oscarblancarteblog.com/2014/07/15/jpa-hibernate-vs-jdbc/>
- [5] <https://neoris.udemy.com/course/angular-spring/learn/lecture/12491054#overview>
- [6] <https://refactorizando.com/base-de-datos-memoria-h2-spring-boot/#:~:text=H2%20es%20una%20base%20de,ejecutarse%20en%20modo%20cliente%20servidor.>