



Escuela Técnica Superior de Ingeniería de Telecomunicación

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Máster

**ESTUDIO DE DESARROLLO DE APP WEB CON
MAVEN, JAVA 11 Y SPRING BOOT EN VISUAL
STUDIO CODE**

Autor: Juan Antonio Tora Cánovas

Tutor: María Dolores Cano Baños

Junio, 2023

Tabla de contenidos

1. Introducción	4
2. Herramientas necesarias.....	6
2.1 Java (Back-end).....	6
2.2 IDE Visual Studio Code.....	7
2.3 Spring Boot	11
2.4 Maven.....	12
2.5 Servidor Tomcat y cliente Http.	16
• Servidor HTTP	16
• Cliente HTTP	18
• Proceso de debug.....	19
2.6 Angular.....	19
3. Instalación y configuración del entorno de trabajo.....	22
3.1 Instalando Java con Open JDK 11 (Back-end)	22
3.2 Instalación del IDE Visual Studio Code.....	30
3.3 Descarga de proyecto prefabricado Spring Boot + Maven (Back-end)	48
3.4 Instalación de Maven en Windows O.S.	61
3.5 Instalación del cliente HTTP.....	73
3.6 Instalación de Angular.	75
4. Desarrollo del proyecto.....	84
4.1 Tutorial Back-end (Spring Boot).....	84
a) ¿Qué hemos explicado ya de Java, Maven y Spring Boot?.....	85
b) Configuración del servidor local Backend.....	86
c) Consultas CRUD con “Insomnia Core”	92
d) Creación de la arquitectura básica del proyecto Backend.....	99
e) Conexión del proyecto Backend a una base de datos “H2”	116
f) Conexión del proyecto Backend a una base de datos Cloud (“Firebase”).....	123
4.2 Tutorial Front-end (Angular)	142
a) Tutorial Básico de TypeScript (con ejercicios):.....	142
b) ¿Qué hemos explicado ya de Angular?.....	152
c) Conceptos Clave y Arquitectura software en Angular.....	153
d) Probando nuestro nuevo proyecto.....	154
e) Viendo el código que incluye nuestro nuevo proyecto.	156
f) Módulos y Componentes en Angular.....	161

g)	Primer componente en Angular: creación, configuración y uso.....	166
h)	Transferencia de datos entre componentes anidados.....	172
i)	Directivas para condicionales y bucles: if, switch, for	179
j)	Interpolación de String y Databinding en Angular	182
k)	Los servicios en Angular	191
l)	Llamadas HTTP y gestión de errores HTTP.....	196
m)	Enrutamiento interno	210
n)	Instalación y uso de librerías de estilos: “Angular Materials” y “Bootstrap”.....	220
o)	Uso de Guardianes en Angular.	225
p)	Instalación de Firebase en un proyecto Angular.	227
q)	Formularios Reactivos.	231
4.3	Aplicación Final (Full Stack).....	233
a)	Incompatibilidades a las que te enfrentarás: solucionadas.....	233
b)	Metodología CI/CD en proyectos Full Stack.....	236
c)	Arquitectura de la aplicación Full Stack.	237
d)	Seguridad de la aplicación Full Stack.	241
e)	Implementación primitiva del Login en el Front-End.....	244
f)	Implementación de la conexión entre cliente Angular y API REST Java.	252
g)	Web final terminada: resultados de codificación.	258
h)	Web final terminada: resultados funcionales.	278
5.	Conclusiones finales.....	286
6.	Anexos	287
	· Anexo I: metodología CI/CD.....	287
	· Anexo II: seguridad.	287
	· Anexo III: gestión del código fuente.	289
	· Anexo IV: buenas prácticas.	290
	· Anexo V: DevOp.	291
	· Anexo VI: Gestión de proyectos ágiles.	294
	· Anexo VII: Servidores Tomcat y compiladores Java en producción.....	295
7.	Bibliografía y webgrafía.....	296

1. Introducción

¿Qué se pretende conseguir con este proyecto tutorial?

La finalidad de este proyecto es crear un tutorial para desarrolladores web principiantes que usen el lenguaje Java, basado en Spring Boot.

El estudiante de este tutorial, al acabar esta lectura sabrá:

- Crear un entorno de trabajo FullStack de forma autónoma desde 0.
- Desarrollar una web FullStack con Java y Angular.
- Las mejores herramientas del mercado propuestas para usarse en futuros despliegues (en entornos de desarrollo, preproducción y producción).

¿A quién va dirigido este tutorial?

El estudiante óptimo para este tutorial es aquel que, teniendo conocimiento medio del lenguaje de programación Java, aún no tenga conocimientos de desarrollo software aplicado a web.

No se requiere que el estudiante tenga un conocimiento previo sobre desarrollo web.

¿Qué tipo de narrativa se ha seguido para la redacción de este tutorial?

La narrativa del tutorial se ha pretendido que sea de tipo “Smooth”, es decir, clara y fácil de entender.

Cada uno de los apartados es eminentemente práctico y se usarán gráficos e imágenes como apoyo cuando se considere necesario.

Se evitan las descripciones o narrativas teóricas siempre que es posible.

¿Cómo se ha conseguido reducir el tiempo necesario desde la fase inicial hasta la fase de lanzamiento a producción?

Mediante el uso de la herramienta “Spring Boot”, la cual nos permite crear y desplegar proyectos como con Spring Framework, pero eliminando ciertas configuraciones repetitivas que existían en Spring Framework.

También se usó Insomnia Core, herramienta DevOp similar a Postman, para probar las llamadas HTTP RESTFull.

¿Cómo podríamos dividir este tutorial?

Este documento se divide en 6 apartados más:

- a) Capítulo 2: descripción teórica de las herramientas que se usarán durante todo el proyecto.
- b) Capítulo 3: instalación completamente guiada de cada herramienta utilizada en el proyecto, antes descritas de forma teórica. Será importante seguir el orden de instalación en que aparecen aquí.
- c) Capítulo 4: contenido central del proyecto. Culmina con el desarrollo de la aplicación final del proyecto.
- d) Capítulo 5: descripción de conclusiones finales y logros adquiridos con este proyecto.
- e) Capítulo 6: área de Anexos, que complementan información de algunos temas tratados en el proyecto.
- f) Capítulo 7: bibliografía y documentación a través de cualquier medio, usada durante la realización de este proyecto.

2. Herramientas necesarias

A continuación, se presentan las herramientas necesarias para desarrollar la aplicación software objeto de este tutorial.

Esta sección será principalmente teórica y técnica, y no será hasta la siguiente sección donde puede encontrarse los tutoriales de instalación de las herramientas que ahora vamos a describir teóricamente.

Para el despliegue del entorno se tendrá en cuenta de forma implícita 2 cosas:

- Metodologías REST.
- Técnicas DevOps o cualquier otra que agilice el proceso de producción y su paso a entorno PRO, sin explicar nada relacionado con DevOps.

2.1 Java (Back-end)

Se usará como leguaje Back-end el lenguaje de programación Java, en su versión 11 Stable (Fig. 2.1).



Fig. 2.1 Logo Java 11

Debido a que la versión 11 Stable de Java es una versión de pago, se usará su alternativa similar al **Open JDK**, es decir, una de sus alternativas Free o gratuitas.

Las 3 razones principales por las que se usará la versión 11 Stable en este tutorial, son:

- La anterior versión Stable v.11 fue Java v.9.
- Java v.11 Stable incorpora medidas de seguridad extras que versiones previas no contienen.
- Java v.11 Stable elimina opciones de seguridad que en versiones previas ya estaban “deprecated”.

¿Más razones para elegir Java 11 Stable?

- Elimina los módulos CORBA:
 - alternativa a RMI que nunca tuvo un uso extendido, y que se prefirió usar antes SOAP y ahora REST (en lugar de CORBA).
- Incorpora sintaxis para la creación de funciones Lambda:

- se permite declarar las variables de funciones lambda con la palabra reservada “var” con inferencia de tipos.
- Permite el uso del cliente HTTP sin ningún problema:
 - El cliente HTTP incorporado desde Java 9 y con soporte para HTTP/2 ahora en la versión Java 11 alcanza la categoría de ESTABLE. Este cliente HTTP será una forma sencilla de hacer llamadas a servicios web REST.
 - Las clases del cliente HTTP se encuentran bajo el paquete “java.net.http”.
 - No serán necesarias librerías de terceros para este cliente HTTP por estar incorporado en el propio JDK de Java 11.
- Incorpora el estándar Unicode 10 que añade diversos nuevos caracteres soportados:
 - 128 nuevos emojis.
 - 19 símbolos nuevos para estándares de TV 4K.
 - Más de 16.000 nuevos caracteres.
- Permite el uso de protocolo TLS 1.3:
 - Versiones anteriores de TLS dejan de tener soporte por considerarse inseguras.
 - Presentar soporte de TLS v1.3 hace que aplicaciones Java sean más seguras y compatibles.
 - El protocolo TLS v1.3 es más seguro y rápido que sus versiones anteriores.

Y... ¿de dónde descargaremos un Open JDK para Java 11 Stable?

- Opción 1: <https://adoptopenjdk.net/>
- Opción 2: <https://adoptium.net/>

2.2 IDE Visual Studio Code

El primer lugar, es importante aclarar lo siguiente: ¿qué es un IDE?

- Un IDE es una aplicación o software informático que integra diversos servicios en ella, para facilitarle al desarrollador el desarrollo de software.
- IDE es un acrónimo que proviene del inglés, traducido al español como “Entorno de Desarrollo Integrado”.

Hace años se usa como IDE a Eclipse, tanto en entornos de docencia como en entornos profesionales. Sin embargo, existen otros IDE's con:

- mejor soporte
- más ligeros
- más actualizados
- más completos
- incluso más simples de instalar que Eclipse.

¿Cuáles son los IDE's que hoy en día vienen a cubrir las carencias de Eclipse?

- intelliJ
- Visual Studio Code

Y en este tutorial, ¿Qué IDE vamos a elegir para el desarrollo de nuestra aplicación?

- Visual Studio Code.

¿Por qué elegimos Visual Studio Code frente a otros IDE's del mercado? Se detallan algunas razones:Desventajas de Eclipse:

- Eclipse actualmente tiene “**deprecated**” **muchos de sus repositorios** para conexión de servicios Cloud.
- Si usáramos Eclipse, el punto anterior **nos fuerza a usar varios IDE's para varios servicios, y esta sería una solución engorrosa.**

Ejemplo:

Es imposible de integrar en Eclipse los servicios de SalesForce (update/download) entre el lado remoto y el lado local.

- Eclipse crea **muchos más directorios de control** en el WorkSpace, en comparación con otros IDE's.
- **Los servicios que poseen mayor complejidad**, así como los proyectos de mayor envergadura, **tienen facilidad para corromperse**, incluso en ediciones y entornos Enterprise.
- **Eclipse funciona más lento**, si lo comparamos con Visual Studio Code.

Desventajas de intelliJ:

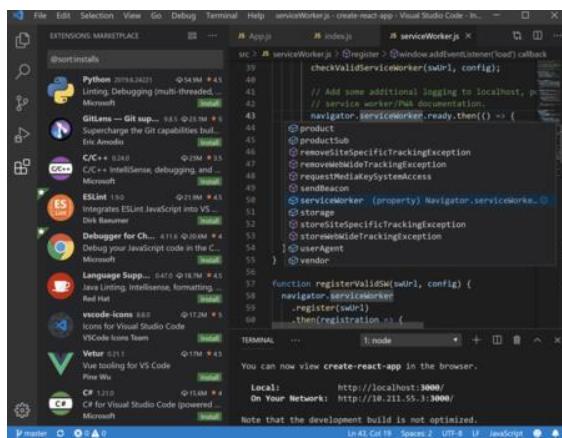
- Pese a su robustez en la integración de servicios como Docker desde el propio IDE, **presenta comando de teclado de “acceso rápido” muy raros y confusos.**

Ventajas destacables de Visual Studio Code frente a los demás:

- Es un IDE que soporta multitud de lenguajes.
- Descarga de todas las librerías y plugins necesarios, desde su propia interfaz (fácil y rápido).
- Detecta y lanza sugerencias automáticas, para las descargas de librerías necesarias (por el propio IDE).
- Repositorios para la conexión con servicios Cloud actualizados y actualizados.
- Interfaz de configuración más intuitiva si lo comparamos con Eclipse, pero similar comparado con intelliJ.
- Respecto a lo compacto y funcional que resulta: mejora respecto a Eclipse, pero similar a intelliJ.

- Mayor nº de comandos de teclado de “acceso rápido” comparado con Eclipse, y similar nº de comandos a los existentes en intelliJ (pero son más intuitivos que los de intelliJ).
- Visual Studio Code incorpora una interfaz de comandos “Command Prompt”, que suele usarse con frecuencia para tareas en entornos virtuales sin necesidad de crear una ventana externa “cmd”.

Puede verse en la siguiente imagen un ejemplo de la interfaz gráfica de Visual Studio Code (Fig. 2.2):



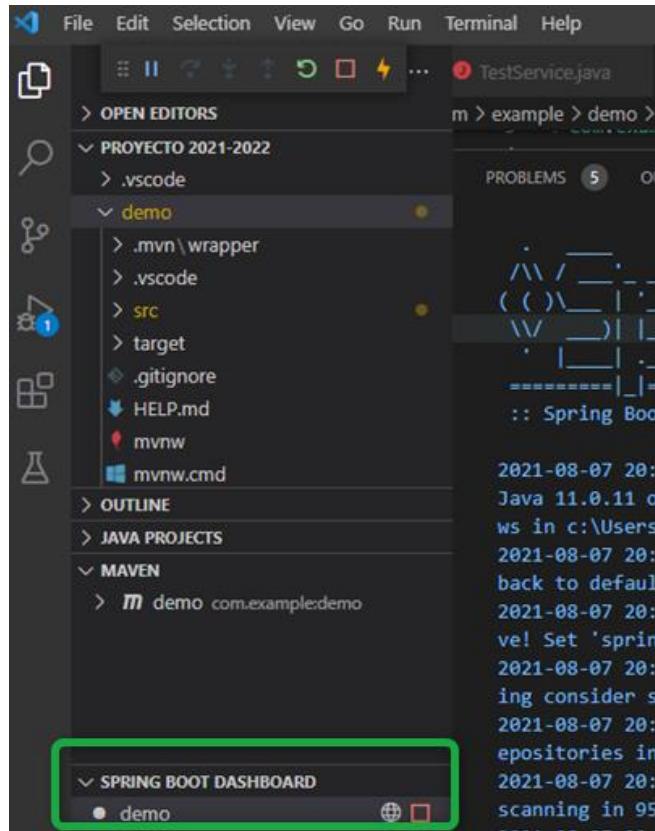


Fig. 2.3: Servidor Tomcat ejecutándose, del proyecto Maven desplegado con framework SpringBoot en IDE Visual Studio Code

En conclusión, respecto a Visual Studio Code, al haber elegido como IDE a Visual Studio Code:

- No necesitaremos descargar una versión especial del IDE (como en el caso de Eclipse con la versión “Spring Tool Suite” o STS).
- Solo requiere bajarse el paquete ZIP pre-fabricado de Spring Boot de una web concreta, y abrir dicho directorio del proyecto desde Visual Studio Code.
- Una vez hecho esto, instalar diversos plugins y aplicaciones necesarias en Visual Studio Code (desde su propia interfaz).
- En pocos minutos podremos empezar a crear nuestra aplicación Spring Boot desde Visual Studio Code sin variantes del IDE ni complejidad adicional.

2.3 Spring Boot

Spring es un framework para desarrollo de servicios y microservicios (Fig. 2.4) y aplicaciones Java, principalmente para la web.

¿Cuál es el secreto para que Spring se siga usando tanto hoy día, desde su creación en el año 2004? Daría 3 motivos, centrados en su funcionalidad:

- 1) Es un framework para aplicaciones, que soporta diversos lenguajes como: XML, JavaScript, JSON, diversas versiones Java, etc.
- 2) Permite inversión de control (IoC).
- 3) Permite inyección de código y de dependencias mediante “etiquetas Spring”.



Fig. 2.4: Logo Spring Framework

Posteriormente a Spring, surge “Spring Boot”.

¿Por qué surge Spring Boot?

Se necesitaba mantener lo mejor de Spring acelerando los procesos de desarrollo, pruebas unitarias, pruebas integradas en entorno DES, entornos PRE y, por último, lanzamiento a entornos PRO (producción).

¿Y cómo logra Spring Boot acelerar los tiempos comprendidos entre desarrollo y producción?

- Cuando desarrollemos con Spring Boot una aplicación, se necesitará menos configuración que si esta misma aplicación la desarrolláramos con Spring.
- Con Spring Boot tendremos las mismas librerías de terceros que tendríamos con Spring.
- Spring Boot incluye diversas herramientas que Spring no tiene para agilizar el proceso productivo.

Más detalles de Spring Boot (Fig. 2.5):

- Permite incrustación servidores Tomcat, Jett o Undertow en el proyecto software (todo ello sin necesidad de implementar WAR's).
- Simplificación la configuración de dependencias.
- Configuración automática de las dependencias que tendríamos en Spring, siempre que sea posible.

- Proporciona funciones que agilizan el paso de los desarrollos a producción, tales como: métricas, comprobación de estado, configuración externa, etc.
- La adaptación con Java no requiere configuración adicional ni más código XML (de lo que necesitaba con Spring puro).



Fig. 2.5: Logo Spring Boot

Por todo ello utilizaremos el **Framework Spring Boot** en este tutorial.

Adicionalmente, Spring Boot permite un servidor Tomcat (HTTP) local y embebido:

- El proyecto que descargaremos prefabricado de Spring Boot ya contiene este servidor Tomcat embebido.

Resumiendo...

- Usaremos el Framework Spring Boot.
- Usaremos un servidor local HTTP haciendo uso de un Servidor Tomcat que llevará embebido nuestro proyecto Spring Boot desde el comienzo del desarrollo (nosotros solo nos tendremos que preocupar de descargar el proyecto Spring Boot con la configuración deseada y ese paquete descargado contendrá el Servidor Local Tomcat).

2.4 Maven

En este proyecto desarrollado con Java, la herramienta Maven cumple una función esencial en la gestión de dependencias de nuestro software.

Lo primero que deberíamos preguntarnos es: ¿Qué es Maven?

- En pocas palabras:

“Maven es una herramienta que permite evitar cualquier problema de portabilidad que pueda ocurrir con una aplicación desarrollada en lenguaje Java y ejecutar la aplicación en cualquier ordenador y sistema operativo, siempre que éste contenga una JVM instalada (Java Virtual Machine)”.

Pero Maven es mucho más que esa breve definición:

- Nos ayudará a crear y gestionar proyectos Java.
- Si necesitas descargar librerías externas, tú no tendrás que descargarte los JAR's o dependencias.
- Esas librerías externas se descargan automáticamente desde “Maven Center”, MVNRepository u otros repositorios cloud vinculados con Maven (de forma fácil y sencilla, usando código XML).
- Máxima portabilidad: no encontrarás problemas de incompatibilidad software ni de los JAR's con otra máquina, ya que al descargarse de Maven Center estas librerías se adecuarán a mi proyecto perfectamente.
- Podrás compilar proyectos Java, incluso si nuestro proyecto depende de otros proyectos en cascadas o se cuentan con aplicaciones estructuras en capas.
- Podrás reestructurar los proyectos de forma ordenada y fácil.
- Toda la configuración de las dependencias externas necesarias, será centralizada en un único fichero por proyecto Maven que tengamos, llamado “pom.xml”.

Existen otras alternativas similares a “Maven”, siendo una de las alternativas más conocida el caso de “*Gradle*”.

Pero, ¿cómo integramos la funcionalidad de Spring Boot con servidor Tomcat incrustado en un proyecto Maven (Fig. 2.4)?

- El proyecto software que descargaremos, en esencia es un proyecto Maven, contenedor a su vez del Framework Spring Boot y el servidor embebido Tomcat (que nos hará el papel de servidor HTTP).
- Para lograr esto, el proyecto software que descargamos inicialmente contiene los ficheros necesarios para lo siguiente, con una estructura de archivos mínimos por defecto (Fig. 2.5)::
 - Usar el Framework Spring Boot y la integración entre todos los lenguajes necesarios para el desarrollo de la aplicación (Java, XML, JSON, js, etc).
 - Un fichero “Pom.xml” para poder indicar al proyecto Maven qué dependencias necesito descargarse de MavenCenter, y después usarlas en el proyecto que estoy desarrollando.
 - Los ficheros necesarios para comunicarse con MavenCenter.
 - Ficheros de configuración usados para “git”, tales como “.gitignore” (código que deseo ignorar por Git) y “Help.md” (mensaje de descripción que deseo leer en git).
 - Directorio “target”, que contendrá los ficheros compilados “.class”.
- Maven creará una carpeta en el directorio personal de mi equipo, y la llamará “.m2”. En esta carpeta descargará las dependencias y las usará en mi proyecto Maven.

En la siguiente figura, la Fig.2.6, tenemos una imagen que representa la imagen del logo de Apache Maven:



Fig. 2.6: Logo Apache Maven

Ahora podemos hacernos una idea de cómo se integran Spring Boot y el servidor Tomcat, dentro de un proyecto Maven.

¿Y qué necesitamos saber de la estructura de un proyecto Maven?

Primer concepto clave de proyectos Maven:

- El fichero “pom.xml” se llama así proveniendo sus siglas de las palabras “Project Object Model”.

Segundo concepto clave de un proyecto Maven:

- En el fichero “pom.xml” puedes configurar, con lenguaje XML, diversas cosas. Se podría destacar:
 - 1 Las librerías/dependencias/JAR’s para que Maven las descargue por ti.
 - 2 Versión de Java que el proyecto Maven necesita.
 - 3 Versión del servidor de aplicaciones: JBoss, Tomcat o cualquier otro.

Tercer concepto clave de un proyecto Maven:

- Un fichero “pom.xml” presenta como mínimo las siguientes propiedades en su interior:
 1. “*ModelVersion*”
 2. “*groupID*”
 3. “*artifactID*”
 4. “*versión*”
- Otras propiedades que el proyecto Maven puede contener... y que estarán dentro de la etiqueta xml llamada “*properties*”, son:
 1. Versión máxima de Java de los JAR que usaré.

2. Versión máxima de Java del código fuente que usaré para desarrollar mi aplicación final
3. Algunas librerías personalizadas que deseas usar tú para tu proyecto.

Cuarto concepto clave de un proyecto Maven:

- En proyectos Maven especialmente grandes, existe un fichero “pom.xml” especial, conocido como SUPERPOM.
- El fichero conocido como “superPom” es un XML donde se indica una descripción sobre cómo Maven internamente deberá hacer la compilación, empaquetado del programa, carpeta targets, resources, etc, de todos los proyectos Maven que integran el macro-proyecto Maven que engloba a la aplicación final.

Quinto concepto clave de un proyecto Maven:

- Podemos usar nuestra herramienta Maven de 2 formas distintas (alternativas):
 1. Descargar del sitio oficial la herramienta Maven y crear un proyecto Maven por línea de comandos dentro de un directorio “x” (el que nosotros deseemos). Posteriormente, montar un proyecto Java en dicho directorio “x”.
 2. Descargar un proyecto Maven ya creado:
 - a. Como si de una dependencia de Maven se tratara, desde un servidor Cloud gestor de versiones el proyecto Maven (de GitBucker, GitHub, etc).
 - b. Después configurar la carpeta “.m2” de Maven y su fichero “settings.xml”, para conectarnos a servidores NEXUS y así bajar las dependencias de software que sean necesarias manualmente (de Nexus).
 - c. Por último, desde IDE’s como Eclipse Spring Boot, Visual Studio Code, u otro, hacer los procesos “Clean-Install” para que quede todo completamente integrado.

Sexto concepto clave de un proyecto Maven:

- Respecto a las dependencias del código, destacar como sitios web preferidos a “Nexus”, “MavenCenter” y “MVNRepository”:
 1. **MVNRepository** es un indexador de artefactos de Maven, en MavenCenter y otros repositorios.
 2. **MavenCenter** es el repositorio oficial.
 3. **NEXUS**: es un repositorio de Maven de código abierto, para que cualquier usuario se lo pueda montar dicho repositorio de una forma privada. Es otro indexador de artefactos Maven, similar a MVNRepository. Es muy importante su carpeta

“.m2” y su fichero interno “settings.xml” donde incluiremos las direcciones de los repositorios de descarga, usuarios y passwords.

Séptimo concepto clave de un proyecto Maven:

- Instrucciones finales para la gestión de dependencias Maven:
 1. Para indicarle a Maven que se descargue una cierta dependencia concreta, ir a buscar esa **dependencia por su nombre en el buscador de la web de MVNRepository**.
 2. Despues localizar el código XML asociado a esta dependencia y pegarlo en el fichero “pom.xml” dentro de la sección de “properties” del “pom.xml” de nuestro proyecto Maven.
 3. Compilar el proyecto Maven en local (realizar el “install”). Esto hará que se descarguen las dependencias necesarias desde MAVEN CENTER de forma automática usando el XML que acabamos de pegar al buscar la librería, conseguir el XML y pegarlo desde MVNRepository en el “pom.xml”.

Si es de interés, podrá verse una imagen de la estructura de un proyecto Maven a continuación, en la Fig. 2.7:

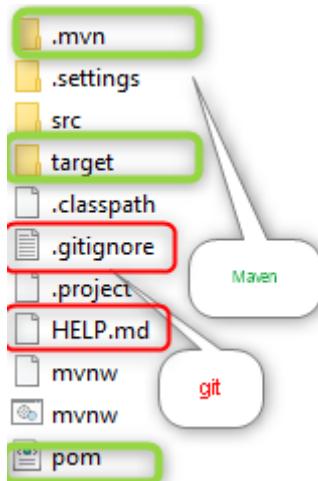


Fig. 2.7: Estructura de un proyecto Maven

2.5 Servidor Tomcat y cliente Http.

- **Servidor HTTP**

Y ¿qué es un servidor Tomcat? Se destacan los siguientes puntos:

- Es propiedad de “Apache Software Foundation”.

- Es un contenedor Web, el cual soporta Servlets y JSP (lenguaje Java).
- Es un servidor que incluye un compilador Jasper, el cual compila código JSP y lo convierte en servlets.
- No es un servidor de aplicación, como sí lo sería por ejemplo JBoss.
- Es un servidor que utiliza el protocolo HTTP.
- El desarrollo software correspondiente al servidor Tomcat, escucha peticiones de uno o más navegadores web.
- Los navegadores web, que harán peticiones al servidor Tomcat, podremos decir que son los clientes de este servidor.
- El servidor Tomcat responde, a una petición del navegador web cliente, con el código necesario para que el navegador logre cargar la página web solicitada (o parte de ella).
- Por tanto, un servidor Tomcat sirve código HTML al navegador web cliente.

Los servidores Tomcat no son los únicos que existen para trabajar con Java. Además de los servidores Tomcat, existen otros servidores que trabajan con el lenguaje Java, con otras ventajas y contras.

Algunas alternativas similares y el motivo para cada elección, por poner algunos ejemplos, serían las siguientes:

1. Si necesitas un servidor ligero con tecnología Apache, entonces elegiremos servidores **Tomcat** (son simplemente un servidor HTTP con un contenedor Servlets de Java).
2. Para soluciones de servidores más completas, optar por otras alternativas de código abierto (las más conocidas):
 - a. Servidores de aplicaciones **JBoss** (de RedHat).
 - b. Servidores **GlassFish**.

¿Qué ofrece JBoss o GlassFish que no ofrece Tomcat? Ventajas:

- Básicamente, Tomcat es solo un servidor de HTTP. Por el contrario, los servidores JBoss y GlassFish hacen más cosas que sólo servir peticiones HTTP.
- Los servidores JBoss y GlassFish pueden llegar a servir aplicaciones Java completas, y además en versión Empresarial (Java EE).

¿Pero qué ventajas tiene un servidor Tomcat frente a otro más robusto?

- La principal ventaja de los servidores Tomcat es que son mucho más ligeros que un servidor de aplicaciones (como JBoss o GlassFish).
- El peso de un servidor Tomcat en MB podría ser de unos 60-70 MB de memoria solamente (con JBoss o GlassFish necesitamos varios cientos de MB).

Respecto a las características de Edición Enterprise, ¿Qué opción es preferible?

- Respuesta rápida: GlassFish.
- Motivo principal: GlassFish está muy por encima en prestaciones Enterprise respecto a las que nos podría ofrecer JBoss.

Hasta el momento se había decidido lo siguiente:

- Uso de Java con lenguaje de la parte del servidor.
- Usar el Framework Spring Boot, para agilizar el proceso desde desarrollo a producción, de una aplicación.
- Protocolo HTTP para comunicar Cliente y Servidor.
- Usaremos Maven en la gestión de librerías y dependencias.

¿Qué servidor vamos elegir y que soporta los 4 puntos anteriores?

- Un servidor Tomcat.

La aplicación que vamos a desarrollar podrá descargarse prefabricada, con un servidor Tomcat embebido (o incrustado) en su interior.

Entonces, podría resultar interesante considerar la jerarquía de directorios que se incluye con la instalación de Tomcat. Podemos servirnos de la siguiente imagen (Fig. 2.8):

La jerarquía de directorios de instalación de Tomcat incluye:

- bin - arranque, cierre, y otros scripts y ejecutables.
- common - clases comunes que pueden utilizar Catalina y las aplicaciones web.
- conf - ficheros XML y los correspondientes DTD para la configuración de Tomcat.
- logs - logs de Catalina y de las aplicaciones.
- server - clases utilizadas solamente por Catalina.
- shared - clases compartidas por todas las aplicaciones web.
- webapps - directorio que contiene las aplicaciones web.
- work - almacenamiento temporal de ficheros y directorios

Fig. 2.8: Resumen de la jerarquía de archivos en un proyecto Maven.

● Cliente HTTP

Como cliente HTTP, usaremos una herramienta muy ligera y portable, llamada "Insomnia Core" (Fig. 2.9):

- Nos servirá para realizar pruebas del desarrollo Back-End.
- Podremos definir si queremos ejecutar peticiones con métodos POST, GET, etc de manera muy simple.
- La herramienta es un ejecutable de Windows.



Fig. 2.9: Logo instalador programa Insomnia Core.

• Proceso de debug

El proceso de pruebas entre cliente/servidor tiene los siguientes aspectos:

1. Nos decidimos por un servidor HTTP tipo Tomcat.
2. Uso de framework de tecnología Back, con servidor Tomcat incrustado: Spring Boot 2.
3. Clientes HTTP: “Insomnia Core” (fase inicial del desarrollo) y “Angular” (fase final).
4. Ejecución y debug con Tomcat: al ejecutar nuestra aplicación back en nuestro IDE, esta correrá sobre el servidor Tomcat.
5. En el cliente HTTP Insomnia podremos ver la respuesta que nos devuelve el servidor Tomcat, a las llamadas hechas a nuestra API REST.

2.6 Angular.

Como indica el nombre de este apartado, se va a utilizar el Framework de Javascript llamado “Angular” para desarrollar la aplicación de este tutorial.

A continuación, se va a justificar esa elección, a través de un desarrollo progresivo y comparativo entre las diferentes alternativas, hasta llegar a la conclusión elegida: Angular.

Actualmente, en lo relacionado con el desarrollo Front-End, el foco se pone en el lenguaje JavaScript (y todos los frameworks que se nutren de JavaScript).

Por una parte, tendríamos JavaScript puro. El desarrollo front-end con JavaScript puro presenta una serie de ventajas y desventajas.

Ventajas:

1. Lenguaje muy sencillo.
2. Rápido en cuanto a tiempos de ejecución.
3. Dispone de varias opciones en cuanto a efectos visuales.
4. Lo soportan los navegadores más populares y los principales sistemas operativos móviles.
5. Lenguaje muy versatilidad.,
6. Multiplataforma y permite ejecución híbrida en cualquier sistema operativo móvil.
7. Único lenguaje que permite trabajar con desarrollos de cualquier tipo en modo Full-Stack.

Desventajas:

1. Los códigos Front-End son legibles por cualquier usuario.
2. Introduce fragmentos de código (legible) en los sitios web.
3. Opciones de 3D limitadas: no se puede crear un juego con JavaScript puro.
4. No compatible con todos los navegadores del mercado de una manera uniforme.
5. El usuario puede desactivar el intérprete JavaScript de su navegador, dejando inhabilitadas toda la funcionalidad aportada con código JS puro.
6. Ficheros JavaScript de tamaño limitado por razones de seguridad, aunque se complementa con otros lenguajes más seguros (como Java 11).

Por otra parte, podríamos usar en este tutorial 3 alternativas distintas a Javascript puro:

- a) Librería jQuery.
- b) Framework Angular JS.
- c) Framework React JS.

Por la parte de jQuery, considerar lo siguiente:

1. jQuery depende de JavaScript, ya que JavaScript está escrito en C, mientras que jQuery está escrito sobre JavaScript.
2. jQuery es una API, no un lenguaje de programación: JavaScript es un lenguaje de programación, mientras que jQuery es una API construida sobre JavaScript.
3. jQuery reduce la codificación necesaria para ciertas operaciones que antes se hacían en JavaScript.
4. En JavaScript se escriben más líneas de código para conseguir lo mismo que con jQuery.
5. jQuery está pensado para funcionar igual en todos los navegadores: en jQuery no necesitamos tener cuidado con las características especiales de cada navegador, con JavaScript sí.
6. jQuery no tiene acceso directo al árbol DOM (como lo teníamos en JavaScript) sin usar una capa extra por encima, lo que puede hacer que perdamos algo de rendimiento respecto a JavaScript.
7. jQuery no permite proyectos software de gran tamaño escalables.

Hasta el momento, no parece buena alternativa usar JavaScript puro si deseamos desarrollar una Web profesional rápidamente, y jQuery tiene importantes limitaciones

de rendimiento. Parece que como últimas opciones solo quedan “Angular” vs “React JS”.

¿Por qué elegimos “Angular” como herramienta Front-End en este tutorial?

- Por un lado, “React JS” es solo una biblioteca para hacer UIs (Interfaces de Usuario) y, aunque React pueda dar resultados más fluidos frente a los de Angular, React JS resulta más complejo de aprender que si usáramos Angular partiendo de Java.
- Por otro, “Angular” es un framework completo para aplicaciones web, que implementa la capa de comunicación cliente-servidor y usa el estándar de codificación TypeScript.
- Además, Angular permite diseñar sitios web SPA (Single Page Application): la página completa se carga al inicio, y las sucesivas actualizaciones se cargan sin actualizar la página completa (solo actualiza el elemento).

Nota Final:

TypeScript permite usar el tipado estático, las clases e interfaces de desarrollo software y algunas cosas más que puede asemejarse con Java conceptualmente (sin dejar de ser orientado a Javascript).

3. Instalación y configuración del entorno de trabajo.

En esta sección se puede encontrar todos los tutoriales de instalación de las 6 herramientas descritas en el capítulo anterior (capítulo 2).

Además, el orden de instalación de las 6 herramientas será similar al orden que se siguió en la sección anterior.

3.1 Instalando Java con Open JDK 11 (Back-end)

Paso 1:

Buscar en Google lo siguiente: “AdoptOpenJDK”.

Paso 2:

Clicar en una búsqueda que nos lleve al siguiente link (o ir desde el siguiente link directamente): <https://adoptopenjdk.net/>.

Paso 3:

Si no encontramos la búsqueda anteriormente indicada, o el link anterior falla, ir a este otro link para su descarga: <https://adoptium.net/> .

Nota Informativa:

- *Desde julio de 2021, la Fundación Eclipse apadrinó el proyecto “AdoptOpenJDK”, y le cambió el nombre de “AdoptOpenJDK” por el de “Adoptium”.*
- *Así se creó una nueva marca y ahora están en proceso de cambio de sitio web.*

Paso 4:

Si funciona la página web del proyecto “AdoptOpenJDK”, debería verse una imagen similar a la Fig. 3.1.

Sino, clicar en el 2º link y nos debe aparecer una imagen más similar a la Fig. 3.2:



Fig. 3.1: web de AdoptOpenJDK, sitio web anterior



Fig. 3.2: web de Adoptium, sitio web actual

Paso 5:

Si hemos elegido el link del paso 2 (del proyecto AdoptOpenJDK) y vemos Fig. 3.1, entonces configurar nuestra descarga como se aprecia en las imágenes de Fig. 3.3 y Fig. 3.4.

Si se eligió el link del paso 3 (del proyecto *Adoptium*) y vemos la imagen de Fig. 3.2 directamente, entonces configurar nuestra descarga como se aprecia en la imagen de Fig. 3.4 (únicamente).

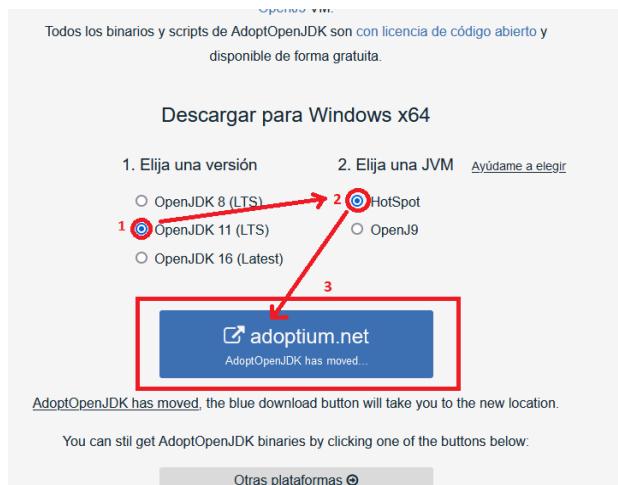


Fig.3.3: configuración proyecto Maven, desde el sitio web anterior



Fig. 3.4: configuración proyecto Maven, desde el sitio web actual

Paso 6:

Descargar el archivo EXE para Windows que contiene el Open JDK. Ver Fig. 3.5.

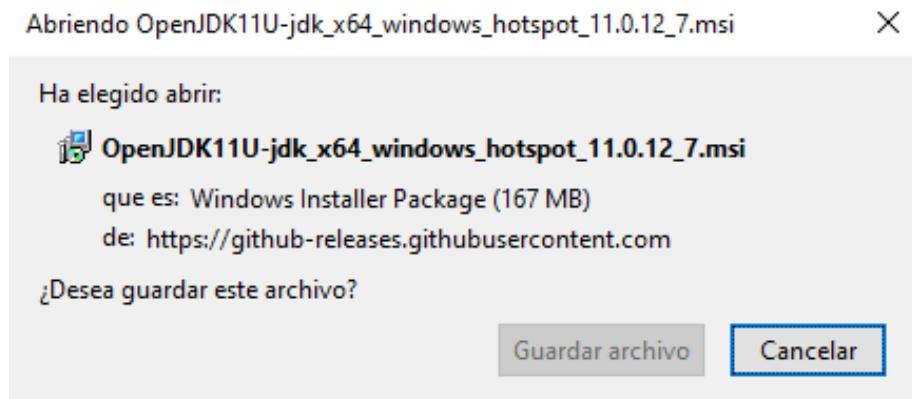


Fig. 3.5: Descarga del instalador del Open JDK 11

Paso 7:

Cuando termine la descarga veremos el siguiente archivo en nuestra carpeta de descargas (Fig. 3.6), de peso aprox. del archivo de 169Mb:

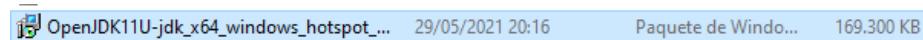


Fig. 3.6: Instalador Windows del Open JDK

Paso 8:

Para empezar con la instalación del JDK, abrir el ejecutable.

Paso 9:

Durante el proceso de instalación, en la primera ventana que aparezca (Fig. 3.7), dar a ACEPTAR:

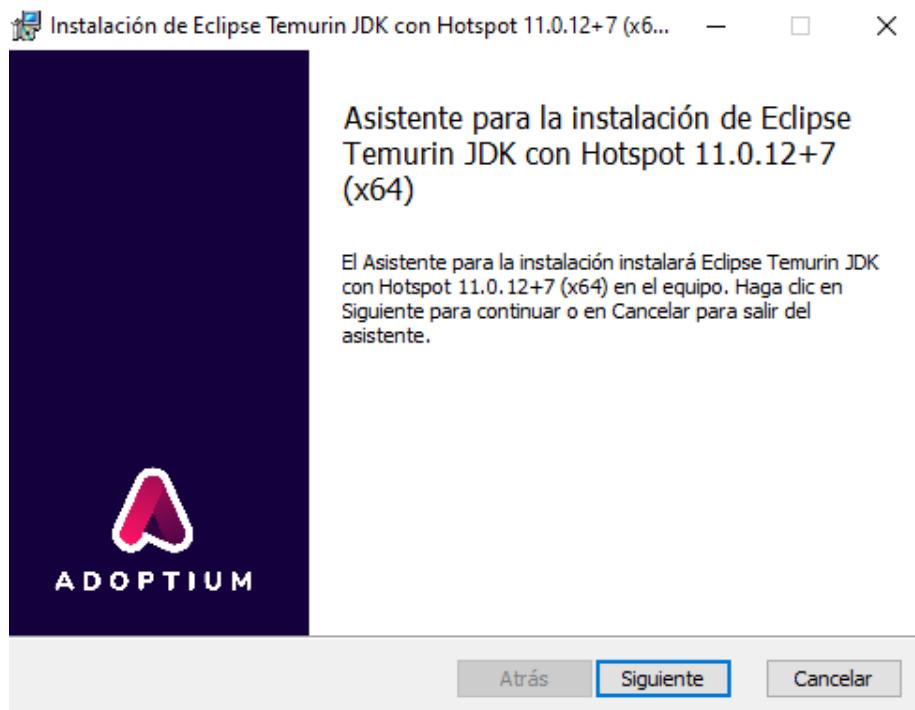


Fig. 3.7: Primera ventana del proceso de instalación del Open JDK

Paso 10:

En la siguiente pantalla dejaremos configuradas las 4 características como en forma de dispositivo, y no con algún tipo de CRUZ ROJA.

¿Cómo SÍ debe hacerse? Ver Fig. 3.8.

¿Cómo NO debe configurarse? Ver Fig. 3.9.

Aclaraciones:

Si nos aseguramos de configurar la instalación como se ilustra en Fig. 3.8, estaremos haciendo las siguientes 3 cosas:

- 1- Poniendo una variable de entorno al nuevo JDK.
- 2- Asignando un PATH de instalación.
- 3- Asociando un fichero .JAR para compilar los ficheros de librería JAVA.

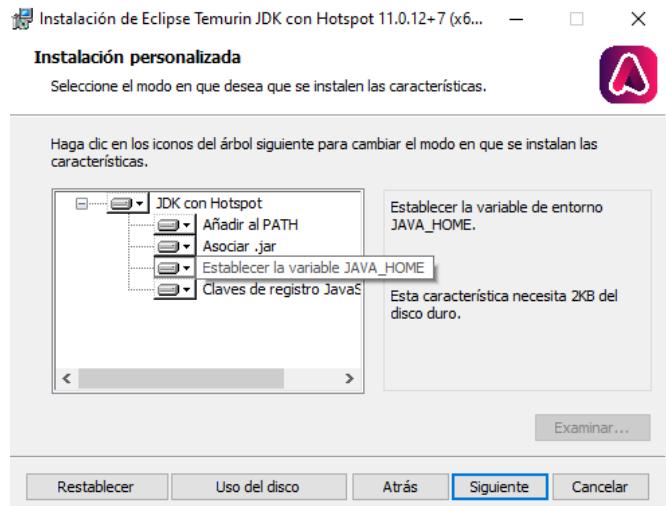


Fig. 3.8: Instalación Correcta

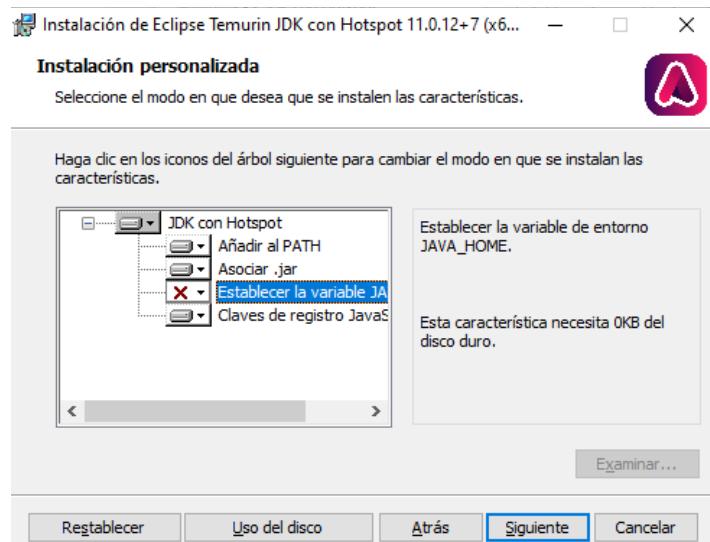


Fig. 3.9: Instalación Incorrecta

Paso 11:

Con la configuración adecuada, dar a SIGUIENTE.

Luego dar al botón INSTALAR.

Paso 12:

Una vez acabado el proceso de instalación, aparecerá la última pantalla del proceso de instalación (Fig. 3.10). Dar al botón FINALIZAR.

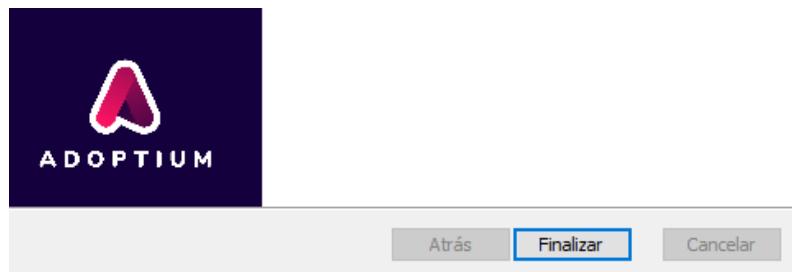


Fig. 3.10: Última pantalla del proceso de instalación

Paso 13:

Si se desea, ir al “Panel de Control” (Fig. 3.11) para comprobar que NO tenemos el JDK 11 correctamente instalado: ni se verá la aplicación Java Virtual Machine (JVM), ni el JDK en la lista de programadas instalados.

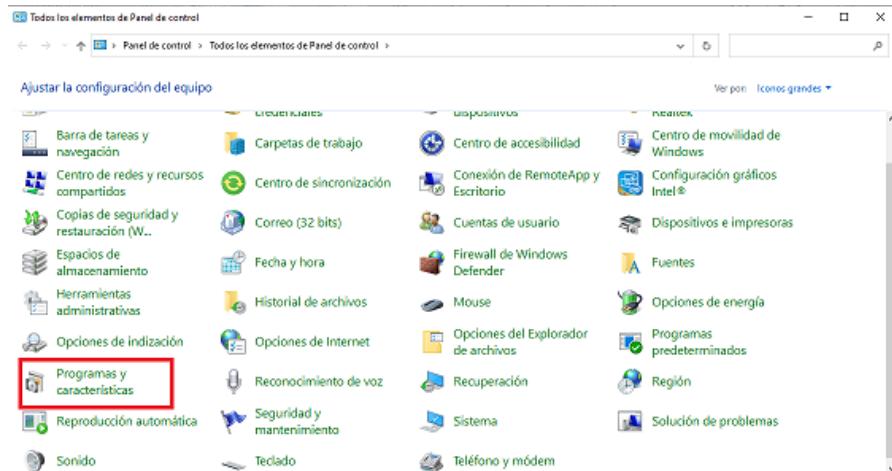


Fig. 3.11: Panel de Control, programas y características

Paso 14:

Ahora vamos a ir, desde “Programas de Windows”, a las “Variables de Entorno de la Cuenta” (Fig. 3.12), y notaremos que SÍ se encuentra el JDK instalado como variable de entorno.

Para ello, una vez entramos a las variables de entorno “de cuenta” o “de sistema” (ambas opciones son válidas), debemos ir al cuadro inferior de la ventana y comprobar que tenemos ya puesta la variable “JAVA_HOME”, en una ruta similar a la que indico en la siguiente imagen de ejemplo (Fig. 3.13).

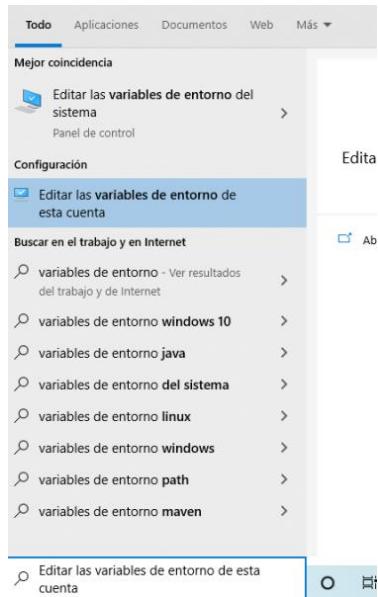


Fig. 3.12: Abrir “Editar Variables de Entorno de esta cuenta” desde botón de Windows.

Variables del sistema	
Variable	Valor
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
JAVA_HOME	C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin;C:\Ana... .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PATHEXT	

Nueva... Editar... Eliminar

Fig. 3.13: Visualización de las variables de entorno del sistema y detección de instalación de la variable de entorno llamada “JAVA_HOME” en ruta donde se ubicó el Open JDK 11.

Paso 15:

Si no contamos con la variable JAVA HOME incluida:

- Incluir la variable de entorno, buscando el programa instalado en una ruta similar a la señalada (Fig. 3.14) y luego añadiendo la ruta en dicha variable de entorno nueva (ruta de Fig. 3.15).

En caso de contar con JAVA_HOME configurada en la lista de variables de entorno:

- Comprobemos que la ruta que indica la variable JAVA_HOME existe (accediendo a la ruta de la variable de entorno), y que contiene todos los archivos necesarios para que el JDK puede funcionar.

> Este equipo > Disco local (C:) > Archivos de programa > AdoptOpenJDK		
Nombre	Fecha de modificación	Tipo
jdk-11.0.11.9-hotspot	29/05/2021 20:20	Carpeta

Fig. 3.14: Ruta donde está instalado mi Open JDK 11.

> Este equipo > Disco local (C:) > Archivos de programa > AdoptOpenJDK > jdk-11.0.11.9-hotspot			
Nombre	Fecha de modificación	Tipo	Tamaño
bin	29/05/2021 20:20	Carpeta de archivos	
conf	29/05/2021 20:20	Carpeta de archivos	
include	29/05/2021 20:20	Carpeta de archivos	
jmods	29/05/2021 20:20	Carpeta de archivos	
legal	29/05/2021 20:20	Carpeta de archivos	
lib	29/05/2021 20:20	Carpeta de archivos	
release	21/04/2021 6:00	Archivo	2 KB

Fig. 3.15: Archivo básicos que debe tener el Open JDK 11 en su interior.

Paso 16:

Por último, para asegurarnos de que nos reconoce las variables de entorno correctamente y el propio JDK, por favor abrir una terminal de windows pulsando Teclas “Windows + R” y en el cuadro de diálogo poner “cmd”, para posteriormente darle a ACEPTAR (Fig. 3.16).

En el nuevo terminal de comandos que hemos abierto, introducir los siguiente: “java –version”. Con esto debe salir información de la misma versión de nuestro JDK que acabamos de instalar (Fig. 3.17).

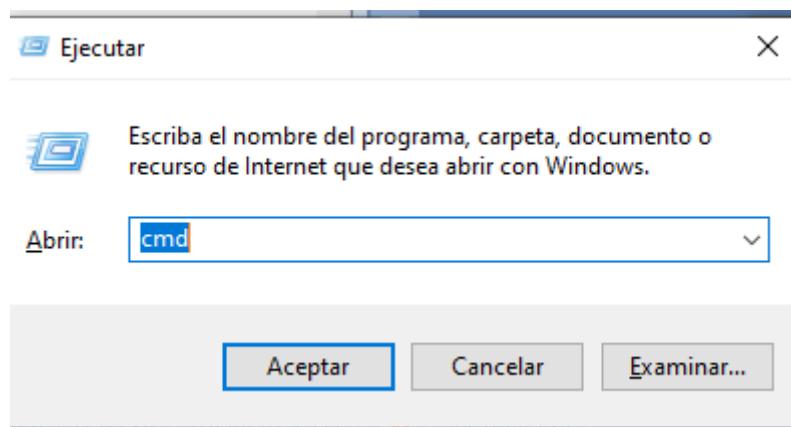


Fig. 3.16: Cuadro de diálogo que abre CMD.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\windows>java --version
openjdk 11.0.11 2021-04-20
OpenJDK Runtime Environment AdoptOpenJDK-11.0.11+9 (build 11.0.11+9)
OpenJDK 64-Bit Server VM AdoptOpenJDK-11.0.11+9 (build 11.0.11+9, mixed mode)

C:\Users\windows>
```

Fig. 3.17: Comprobación de la versión de Java instalado en el sistema, a través de la nueva venta de comandos de Windows.

Si en tu caso apareció algo similar a nuestra figura 3.17, podremos estar completamente seguros de que el JDK 11 de Java Open está instalado correctamente.

3.2 Instalación del IDE Visual Studio Code

Visual Studio Code será el IDE (Entorno de Desarrollo Integrado) que usaremos para crear nuestra página web desarrollada en Java y frontend con Angular.

Para instalar Visual Studio Code y configurarlo para nuestra puesta en marcha debemos hacer los siguientes pasos en el orden descritos a continuación en este apartado:

Paso 1: Accede a la web oficial de Visual Studio Code siguiendo este link

<https://code.visualstudio.com/>

Paso 2: Se mostrará una imagen similar a la siguiente (Fig. 3.18), donde aparecerá un botón de descarga con opción de desplegarlo. Si se hace clic en la flecha despegable

tendrá la opción de descargar el instalador para macOS, Windows x64 o Linux x64. Como el entorno de trabajo se está montando en un Windows 10 de 64 bits, se seleccionará, en este caso, la opción de Windows x64 Stable (selecciona la opción que mejor se adapte a tu caso en el momento que estés instalando tu Visual Studio Code).

Importante: en caso de tener varias versiones disponibles, la Stable de número de subversión más elevado siempre es preferible frente a cualquier otra.

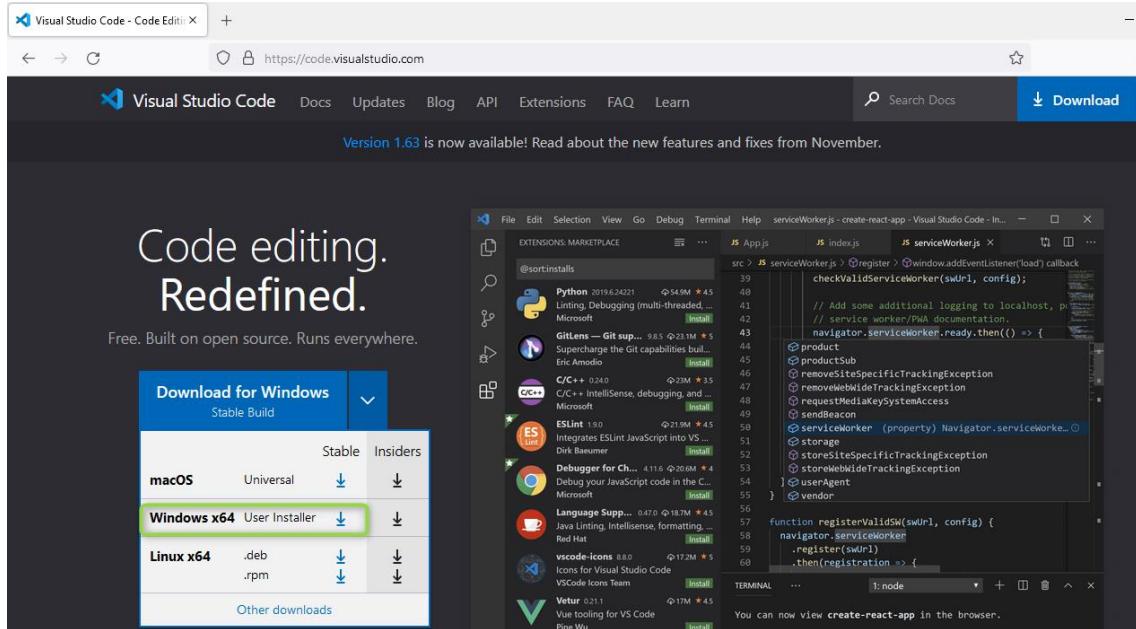


Fig. 3.18: Web oficial para descarga de Visual Studio Code.

Paso 3: Cuando descargamos el ejecutable instalador de Visual Studio Code, nos aparece un tutorial de primeros pasos. Puede verse el tutorial oficial que se nos abre en la Fig. 3.19.

The screenshot shows the official Visual Studio Code website. At the top, there's a dark header with the "Visual Studio Code" logo and links for "Docs", "Updates", "Blog", "API", "Extensions", "FAQ", and "Learn". To the right is a search bar with the placeholder "Search Docs". Below the header, a message says "Version 1.63 is now available! Read about the new features and fixes from November." In the main content area, there's a green box with the text "Thanks for downloading VS Code for Windows!". It includes a link to a direct download and a call to action to take a survey. To the left, a sidebar lists various "Getting Started" topics: Overview, SETUP, GET STARTED, USER GUIDE, LANGUAGES, NODEJS / JAVASCRIPT, TYPESCRIPT, PYTHON, JAVA, C++, CONTAINERS, DATA SCIENCE, AZURE, and REMOTE.

Getting Started

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity). Begin your journey with VS Code with these [introductory videos](#).

Visual Studio Code in Action

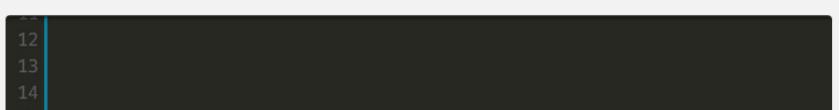


Fig. 3.19: Tutorial de primeros pasos para Visual Studio Code, cuando se ejecuta la descarga del instalador oficial.

Paso 4: Ir a nuestro directorio de “Descargas”, de nuestro equipo local, y ejecutar el instalador de Visual Studio Code que acabamos de descargar (Fig. 3.20).

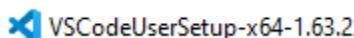


Fig. 3.20: Instalador descargado de Visual Studio Code.

Paso 5: Cuando empieza la instalación, se deben aceptar los términos de instalación como se ve en las imágenes, y luego dar a botón “siguiente” e “instalar” en varias ocasiones (proceso de instalación que puede encontrarse visualmente desde la Fig. 3.21 hasta la Fig. 3.27).

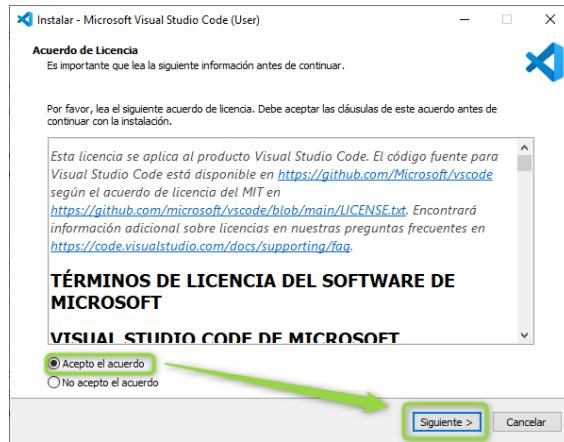


Fig. 3.21: Aceptación de los términos y condiciones del programa Visual Studio Code.

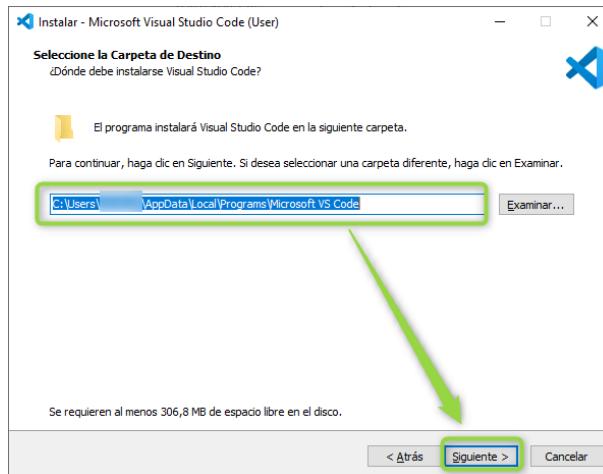


Fig. 3.22: Selección la ruta de instalación donde ubicaremos el programa (dejar por defecto).

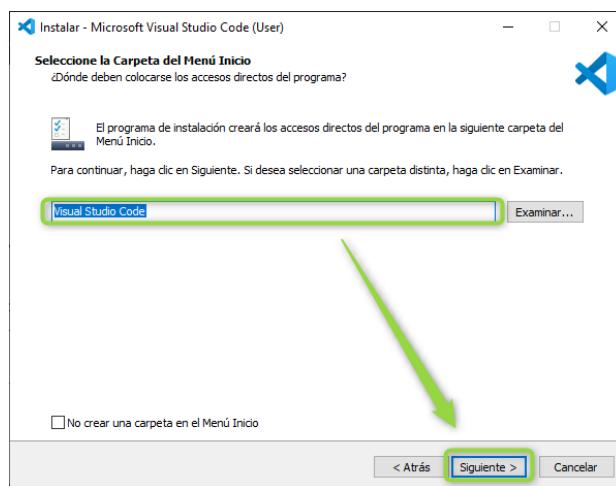


Fig. 3.23: Selección de carpeta donde colocaré los accesos directos (dejar por defecto).

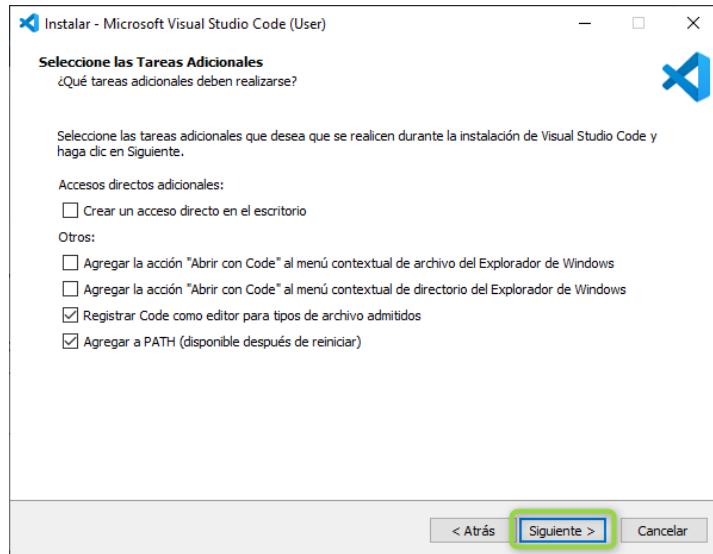


Fig. 3.24: Pantalla de tareas adicionales previa a la instalación del programa.

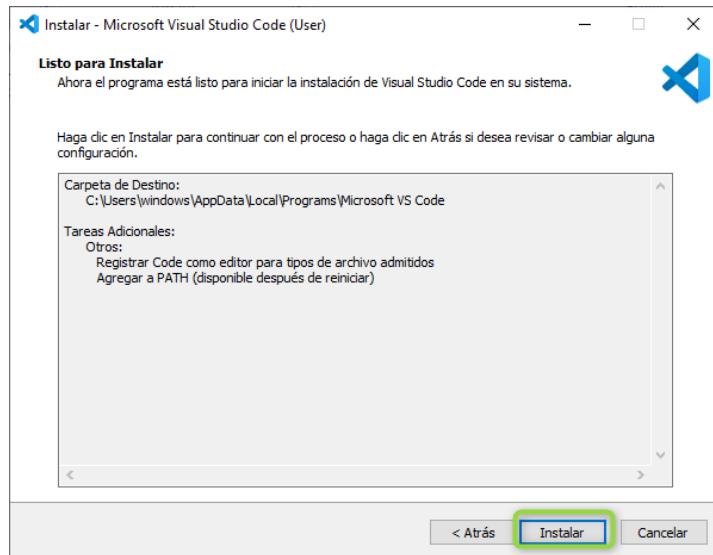


Fig. 3.25: Pantalla de visualización de tareas e instalables configurados para la instalación.

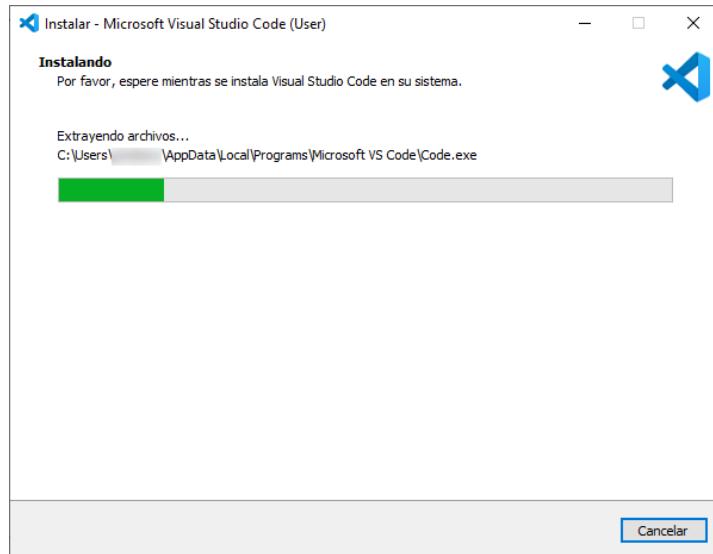


Fig. 3.26: Proceso de instalación en curso.

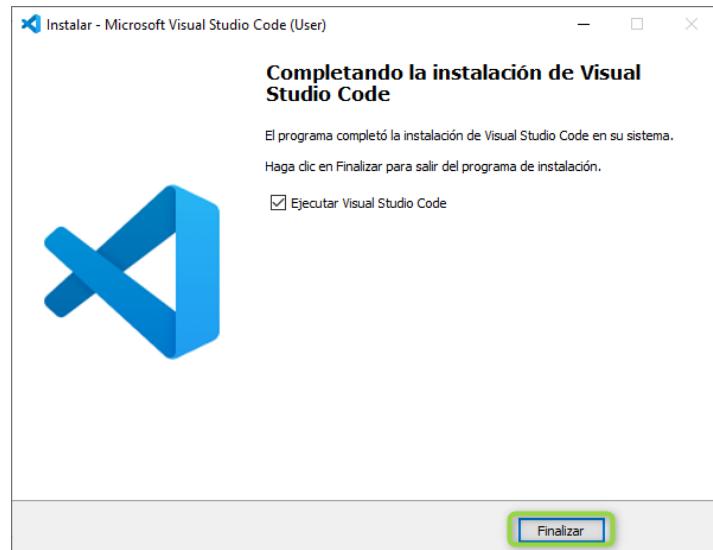


Fig. 3.27: Instalación finalizada y pantalla final.

Paso 6: Una vez hemos instalado el entorno de desarrollo integrado Visual Studio Code correctamente, lo abriremos por primera vez.

Primero, instalar el paquete de idiomas cuando nos salga abajo a la derecha la sugerencia (ver Fig. 3.28). Esperaremos a que termine de instalar el paquete de idiomas. Sabremos que el paquete de idiomas está instalado porque después de instalarlo se verá todas las opciones de la barra superior del programada en idioma Español/Castellano.

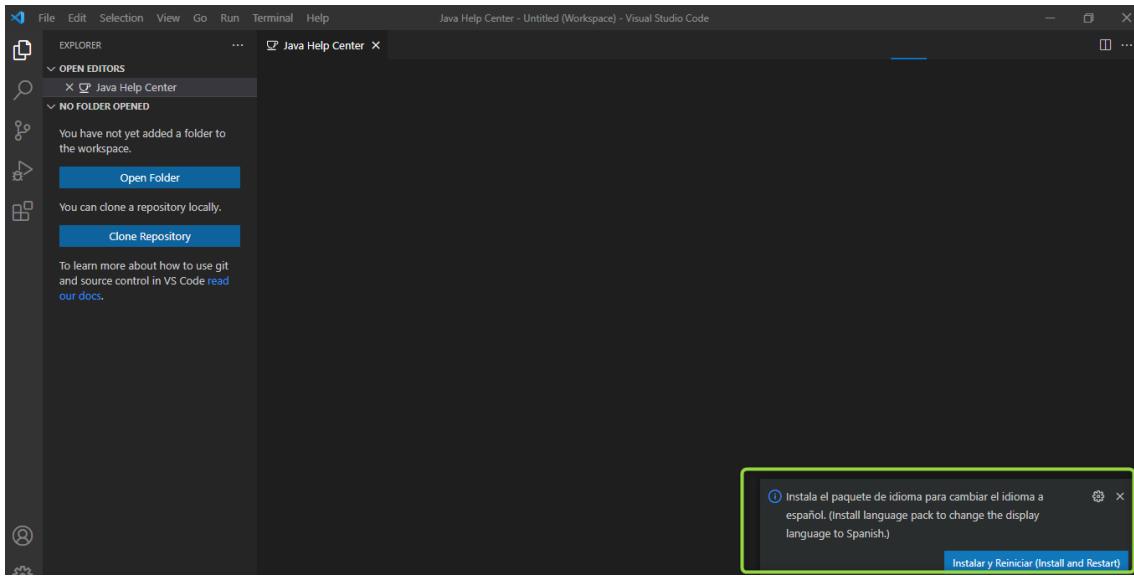


Fig. 3.28: Salto de sugerencia para instalar paquete de idioma en español, e instalación del paquete en español, tras aceptar la sugerencia emergente.

Tras la instalación vemos los menús superiores en español (Fig. 3.29).

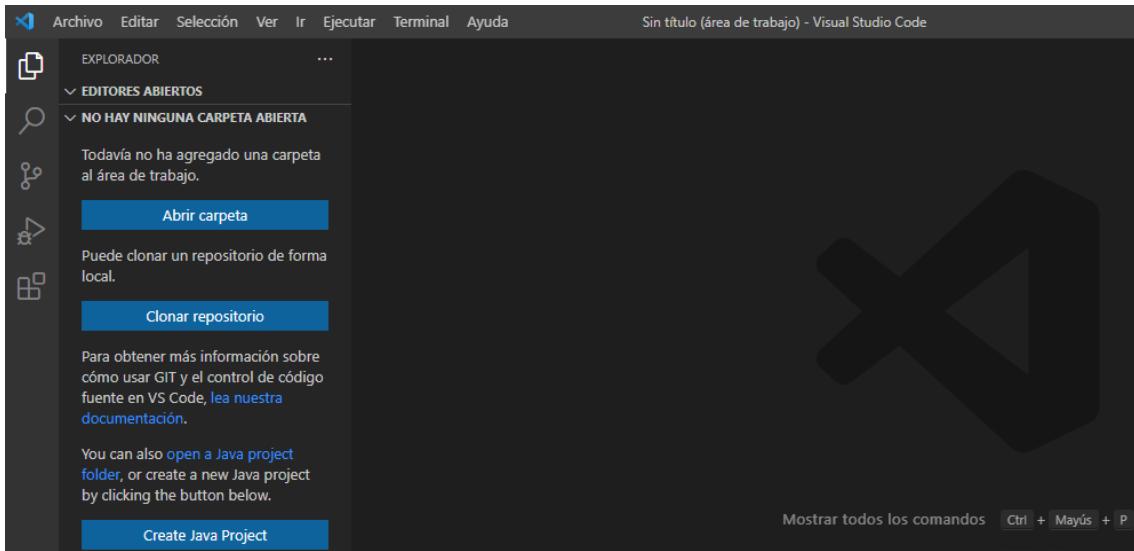


Fig. 3.29: Cambio de idioma realizado correctamente a español.

Paso 7: Ahora que tenemos el IDE Visual Studio Code abierto y configurado en español, lo primero que recomiendo es anclar el acceso directo del programa a la barra de herramientas inferior de Windows 10 (en mi caso). Para ello clicar con el botón derecho del ratón en el ícono de Visual Studio Code y seleccionar en la lista “Anclar a la barra de Tareas” (podrá ver el proceso en las Fig. 3.30 y Fig. 3.31).

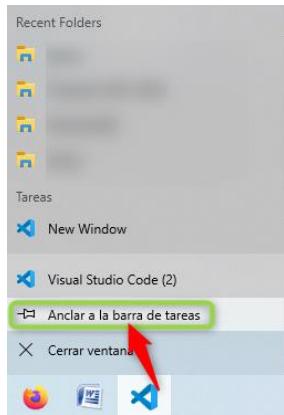


Fig. 3.30: Botón derecho sobre el ícono de programa abierto en la barra de tareas y seleccionar “Anclar a la barra de tareas”.

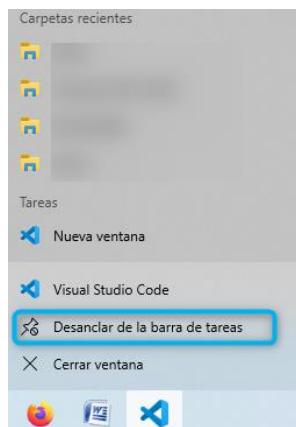


Fig. 3.31: Se vuelve a dar a botón derecho del ratón para comprobar que esta opción ahora permite “desanclar”, pero sin clicar en nada (solo comprobamos visualmente).

Paso 8: Ahora tenemos que instalar el pack de extensiones para Java (Fig. 3.32) y para Spring Boot (Fig. 3.33) en Visual Studio Code, y así adaptar correctamente el IDE al desarrollo con estas herramientas.

Clicar, para cada caso, en el botón de INSTALAR



Fig. 3.32: Instalación de la extensión “Extension Pack for Java” en Visual Studio Code.



Fig. 3.33: Instalación de la extensión “Spring Boot Extension Pack” en Visual Studio Code.

Paso 9: También seleccionamos una extensión para lenguaje XML, y cuando nos aparezca la lista para elegir, seleccionamos la que tenga como fabricante a “Red Hat” (ver Fig. 3.34):



Fig. 3.34: Extensión sin instalar de Visual Studio Code, para lenguaje XML.

Cuando queda instalada la extensión para XML, el plugin se visualiza como vemos a continuación (Fig. 3.35), y esta nos servirá para visualizar correctamente el código del POM (sin errores).



Fig. 3.35: Extensión ya instalada en Visual Studio Code, para lenguaje XML.

Paso 10: Una vez cargado el workspace de Visual Studio Code sin presentar errores en el código, ir a la sección de “extensiones” de este IDE.

Se instalarán una serie de 32 Plugin's aprox. (entro del IDE) para poder trabajar, en el desarrollo Front-end y Back-end, de una forma mucho más cómoda.

NOTA: se van a mostrar las imágenes de las instalaciones a pares: antes de instalar plugging (izq) vs instalación OK (der).

- PHPStorm Formatter + PHP Snippets From PHPStorm (Fig. 3.36 y Fig. 3.37):

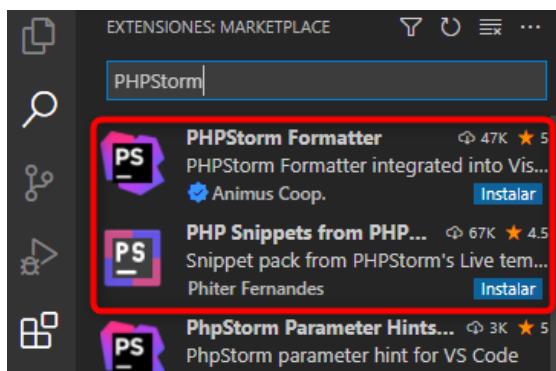


Fig. 3.36: Plugin's 1 y 2 no instalados.

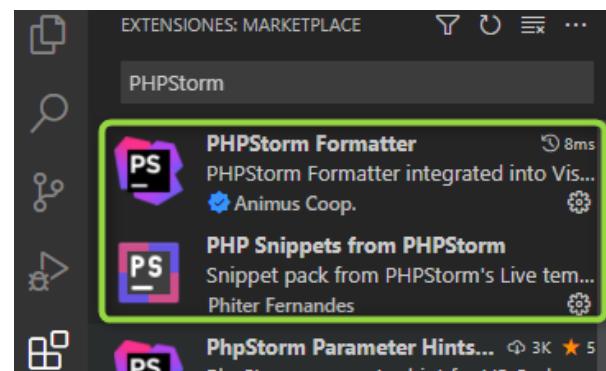


Fig. 3.37: Plugin's 1 y 2 instalados correctamente.

- PHP Intelephense + PHP Debug (Fig. 3.38 y Fig 3.39):

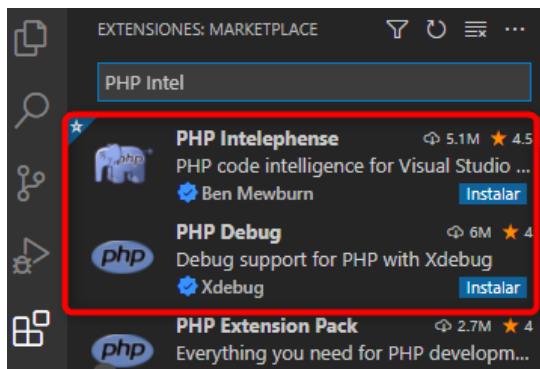


Fig. 3.38: Plugin's 3 y 4 no instalados.

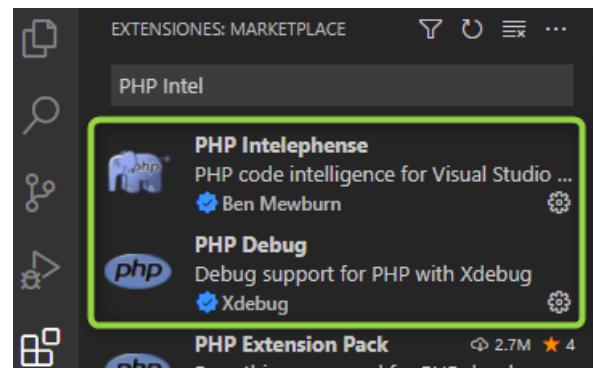


Fig. 3.39: Plugin's 3 y 4 instalados correctamente.

- PHP Getters & Setters (Fig. 3.40 y Fig. 3.41):

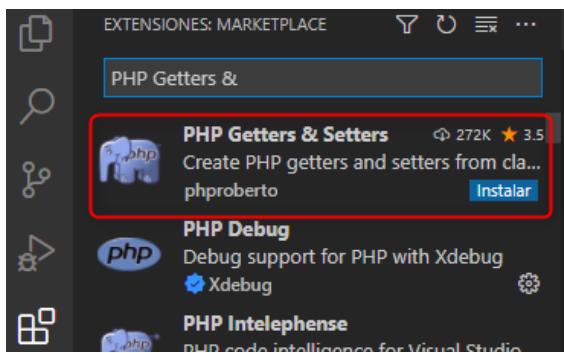


Fig. 3.40: Plugin 5 no instalado.

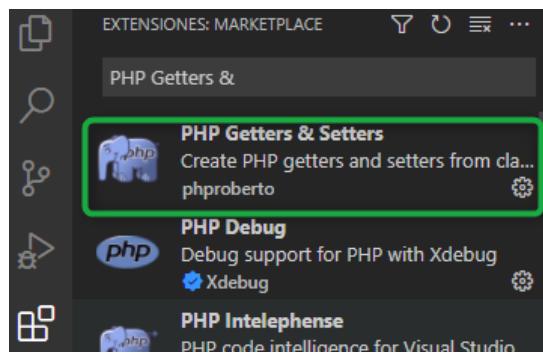


Fig. 3.41: Plugin 5 instalado correctamente.

- php cs fixer (Fig. 3.42 y Fig. 3.43):

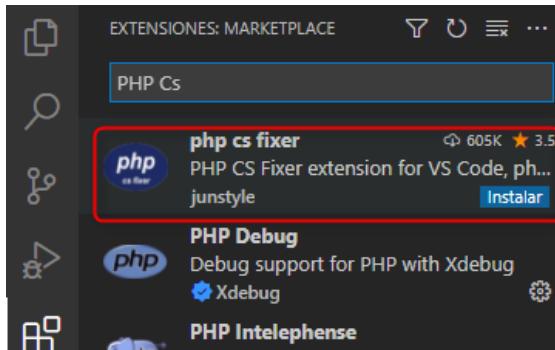


Fig. 3.42: Plugin 6 no instalado.

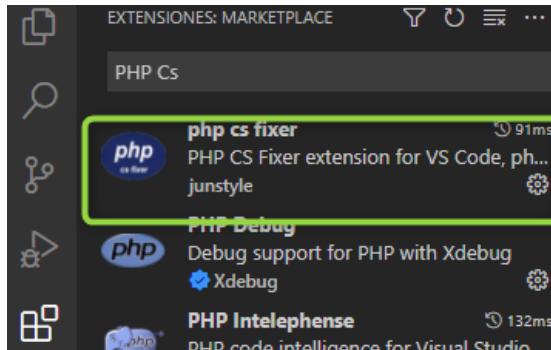


Fig. 3.43: Plugin 6 instalado correctamente.

- Twig Language 2 (Fig. 3.44 y Fig. 3.45):

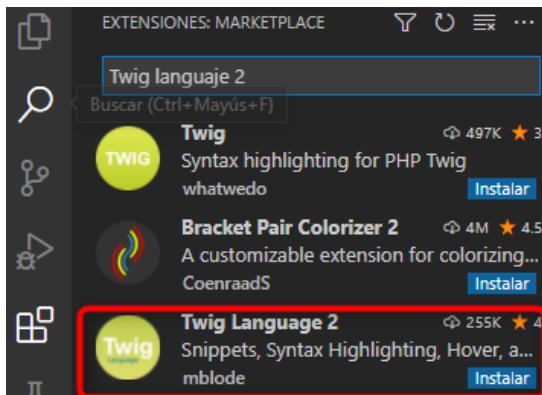


Fig. 3.44: Plugin 7 no instalado.

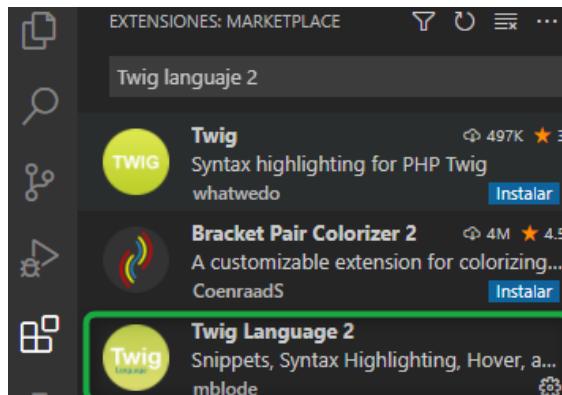


Fig. 3.45: Plugin 7 Instalado correctamente.

- SQLTools (Fig. 3.46 y Fig. 3.47):

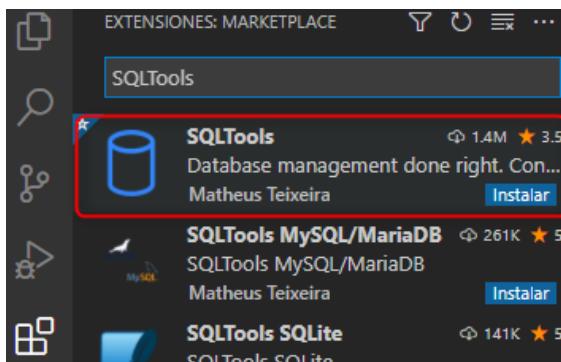


Fig. 3.46: Plugin 8 no instalado.

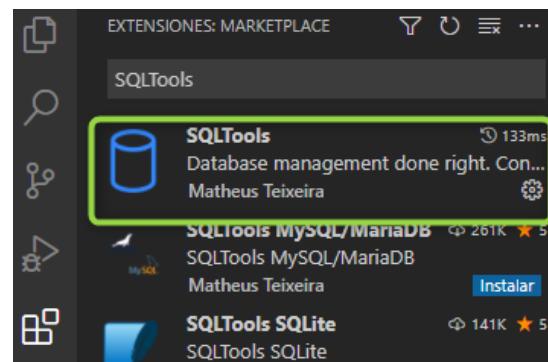


Fig. 3.47: Plugin 8 instalado correctamente.

- C# (Fig. 3.48 y Fig. 3.49):

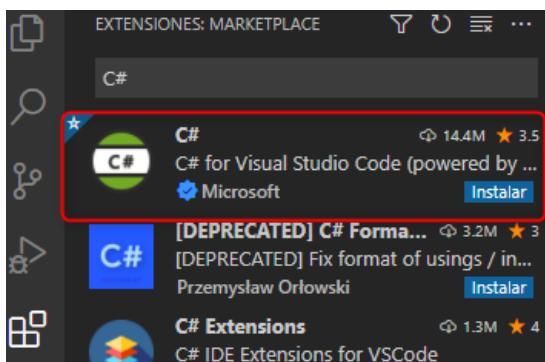


Fig. 3.48: Plugin 9 no instalado.

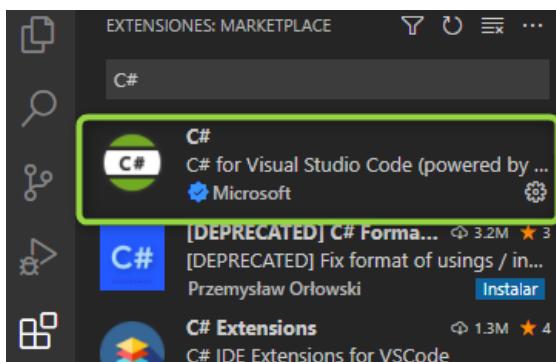


Fig. 3.49: Plugin 9 instalado correctamente.

- Path Autocomplete (Fig. 3.50 y Fig. 3.51):

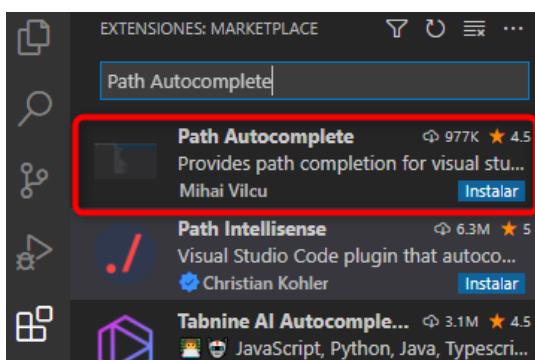


Fig. 3.50: Plugin 10 no instalado.

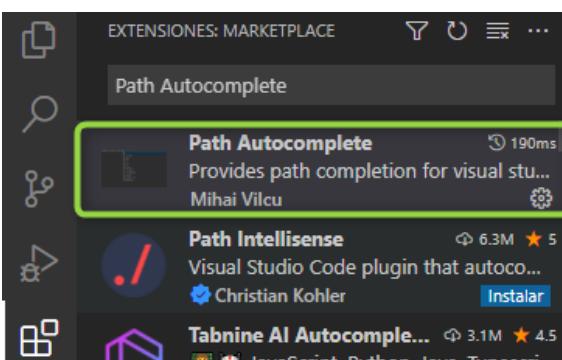


Fig. 3.51: Plugin 10 instalado correctamente.

- Php Namespace Resolver (Fig. 3.52 y Fig. 3.53):

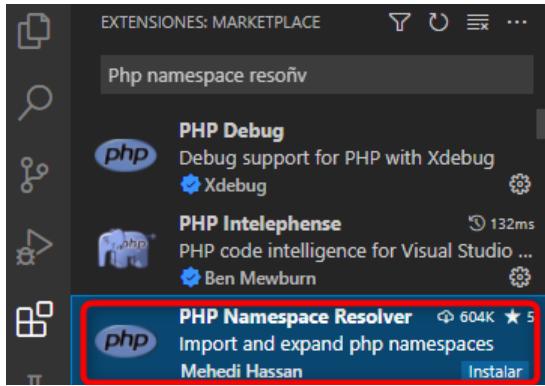


Fig. 3.52: Plugin 11 no instalado.

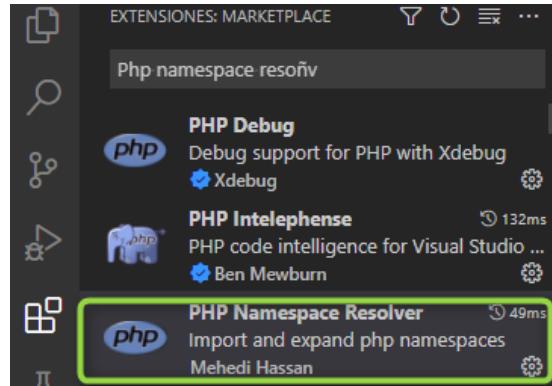


Fig. 1.53: Plugin 11 instalado correctamente.

- DotENV (Fig. 3.54 y Fig. 3.55):

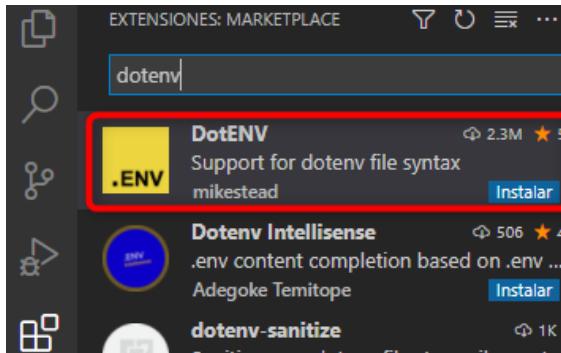


Fig. 3.54: Plugin 12 no instalada

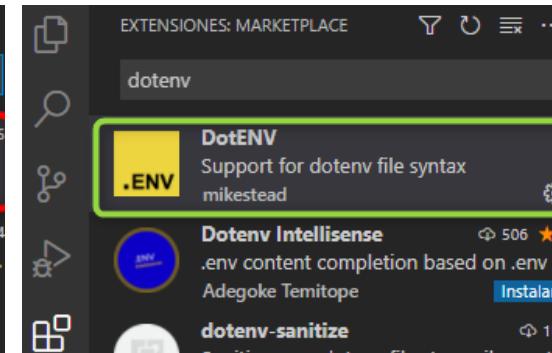


Fig. 3.55: Plugin 12 instalada correctamente.

- Bracket Select + Bracket Pair Colorizer (Fig. 3.56 y Fig. 3.57):

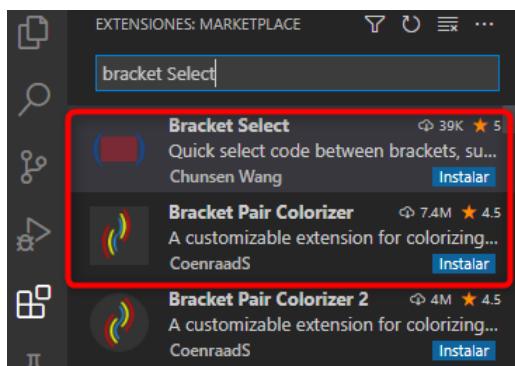


Fig. 3.56: Plugins 13 y 14 no instalados.

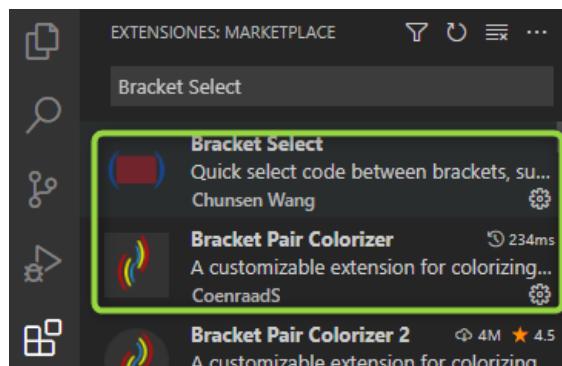


Fig. 3.57: Plugins 13 y 14 instalados correctamente.

- Beautify (Fig. 3.58 y Fig. 3.59):

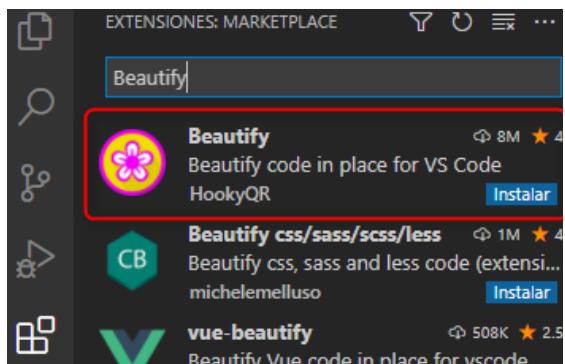


Fig. 3.58: Plugin 15 no instalado.

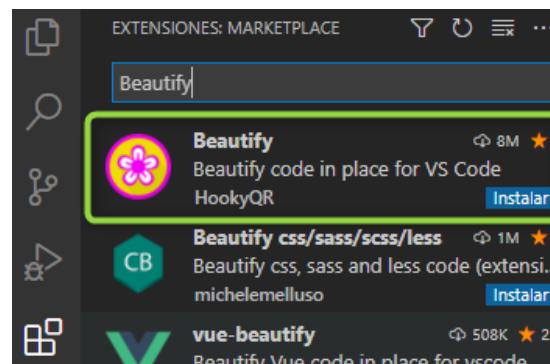


Fig. 3.59: Plugin 15 instalado correctamente.

- PHP Constructor (Fig. 3.60 y Fig. 3.61):

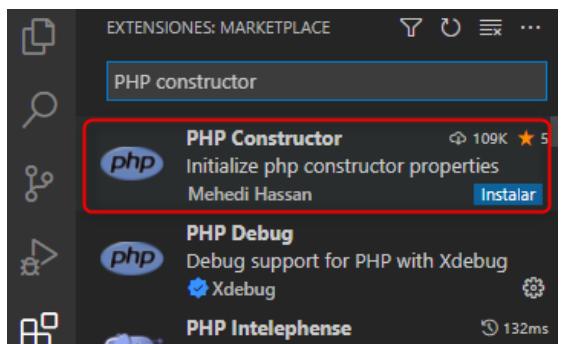


Fig. 3.60: Plugin 16 no instalado.

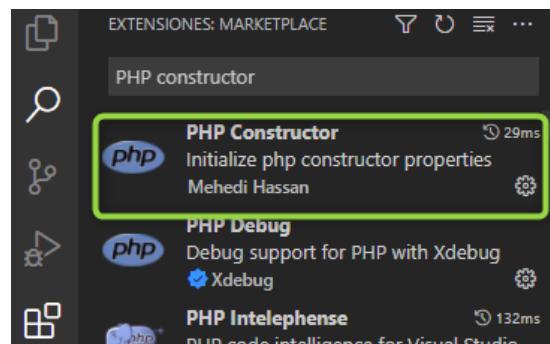


Fig. 3.61: Plugin 16 instalado correctamente

- Symfony for VSCode (Fig. 3.62 y Fig. 3.63):

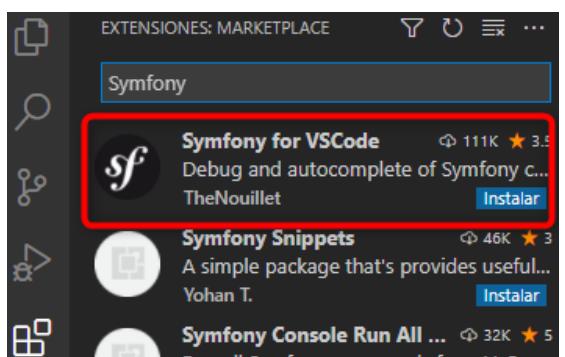


Fig. 3.62: Plugin 17 no instalado.

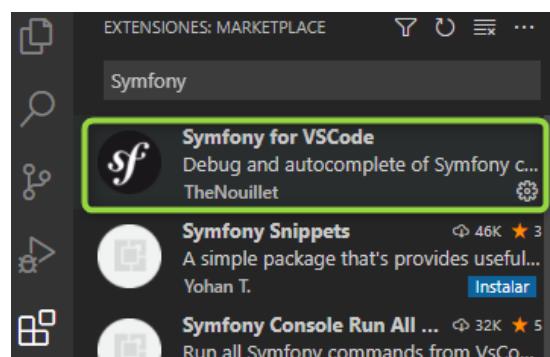


Fig. 3.63: Plugin 17 instalado correctamente

- ESLint (Fig. 3.64 y Fig. 3.65):

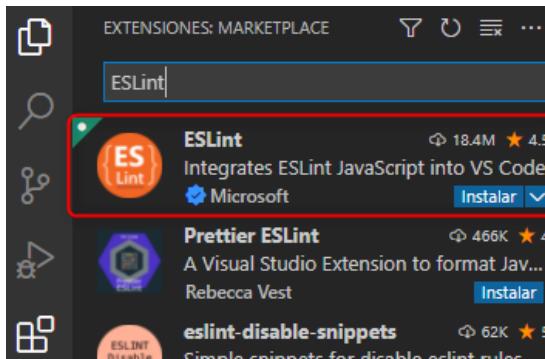


Fig. 3.64: Plugin 18 no instalado.

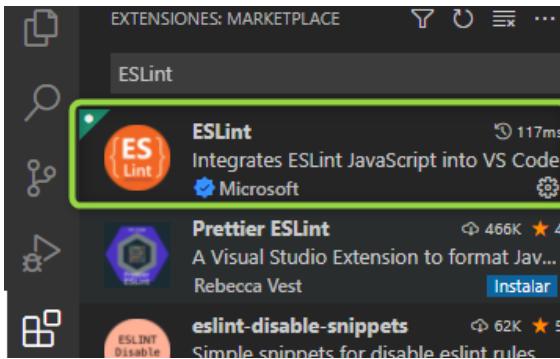


Fig. 3.65: Plugin 18 instalado correctamente

- Angular 10 Snippet + Angular Snippets (Fig. 3.66 y Fig. 3.67):

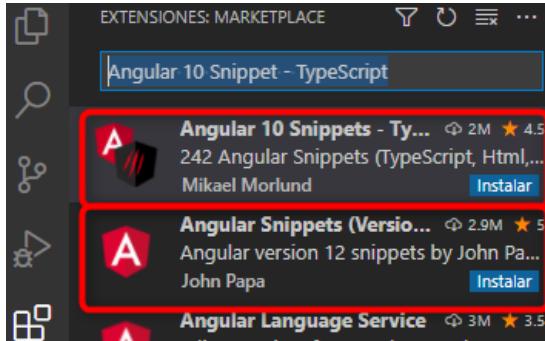


Fig. 3.66: Plugin's 19 y 20 no instalados.

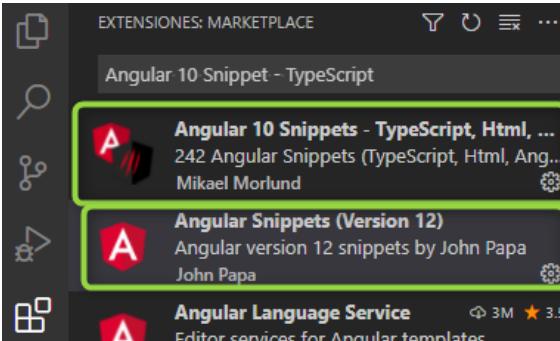


Fig. 3.67: Plugin's 19 y 20 instalados correctamente.

- HTML css Support (Fig. 3.68 y Fig. 3.69):

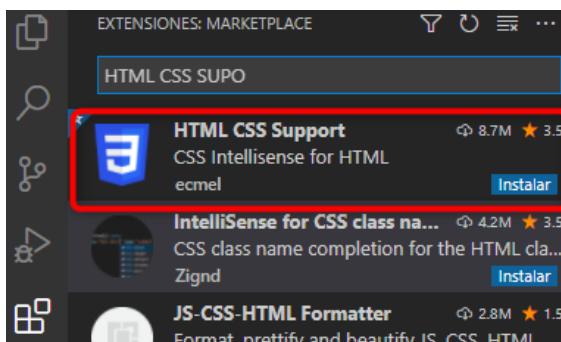


Fig. 3.68: Plugin 21 no instalado.

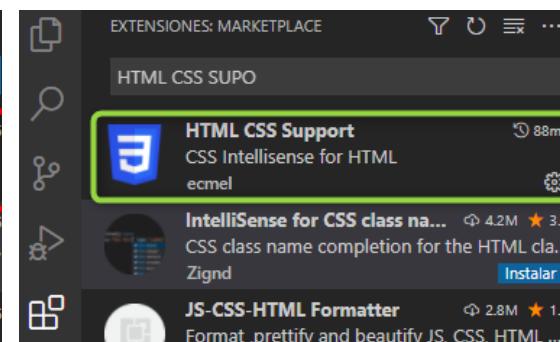


Fig. 3.69: Plugin 21 instalado correctamente.

- HTML Snippets (Fig. 3.70 y Fig. 3.71):

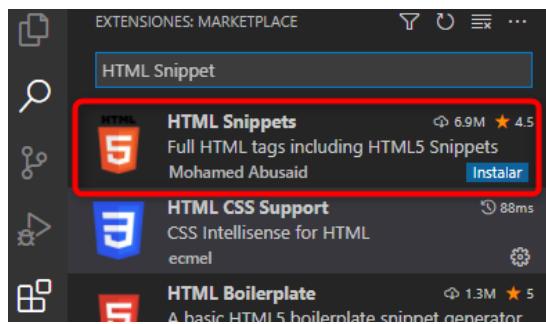


Fig. 3.70: Plugin 22 no instalado.

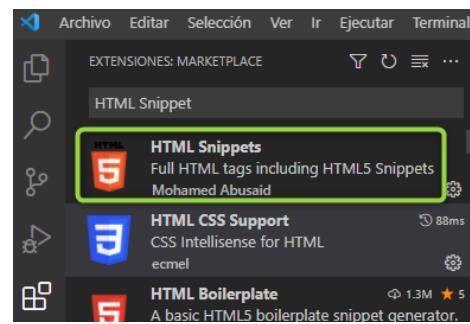


Fig. 3.71: Plugin 22 instalado correctamente.

- Javascript (ES26) code Snippets (Fig. 3.72 y Fig. 3.73):

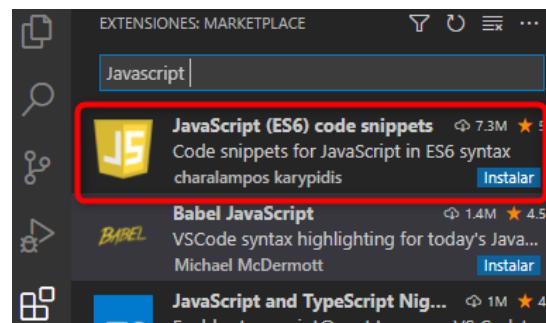


Fig. 3.72: Plugin 23 no instalado.

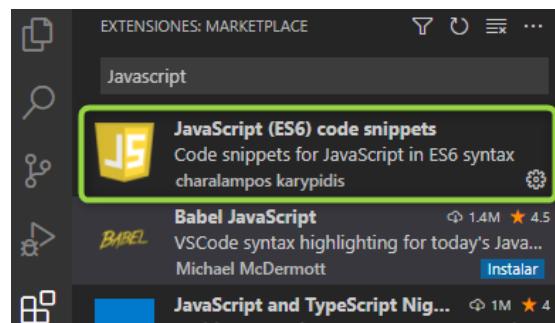


Fig. 3.73: Plugin 23 instalado correctamente.

- Sass (Fig. 3.74 y Fig. 3.75):

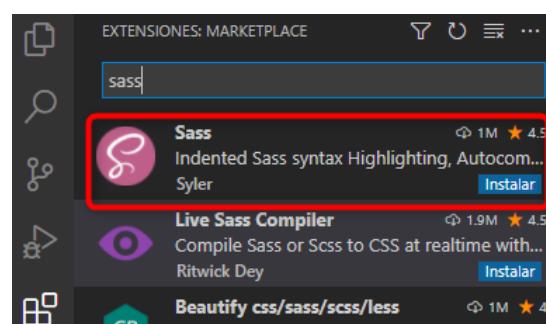


Fig. 3.74: Plugin 24 no instalado.

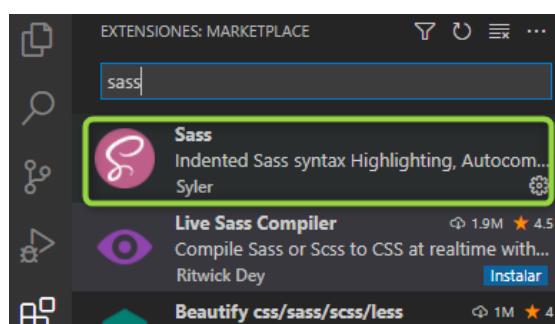


Fig. 3.75: Plugin 24 instalado correctamente.

- Vscode-icons (Fig. 3.76 y Fig. 3.77):

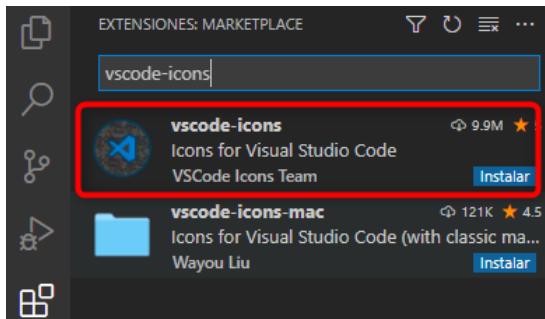


Fig. 3.76: Plugin 25 no instalado.

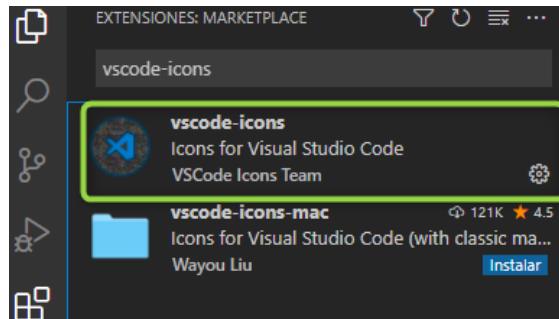


Fig. 3.77: Plugin 25 instalado correctamente.

- JQuery Code Snippet (Fig. 3.78 y Fig. 3.79):

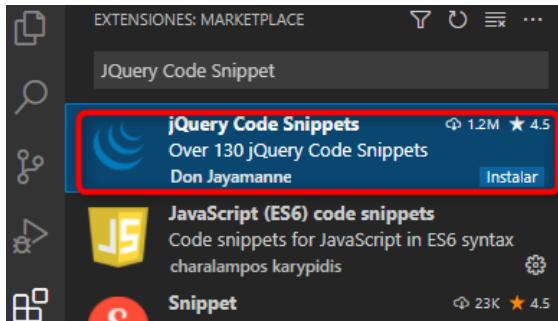


Fig. 3.78: Plugin 26 no instalado.

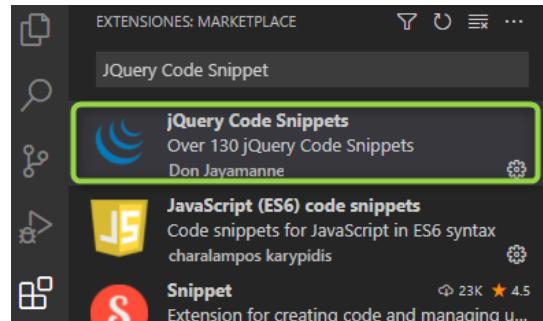


Fig. 3.79: Plugin 26 instalado correctamente.

- Language Support for Java(TM) by Red Hat (Fig. 3.80):

Debe venir preinstalado ya, o instalarlo al comiendo de la instalación de BakEnd o en instalaciones previas.

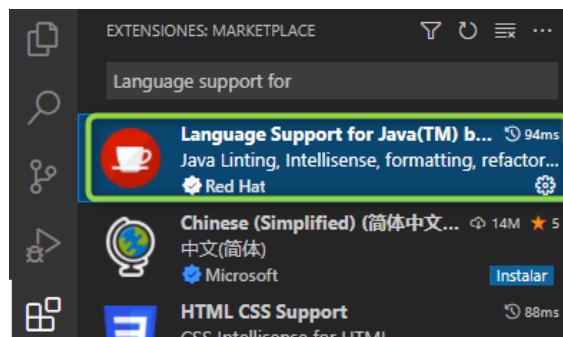


Fig. 3.80: Plugin 27 instalado correctamente.

- BootStrap4, Front Awesome 4 (Fig. 3.81 y Fig. 3.82):

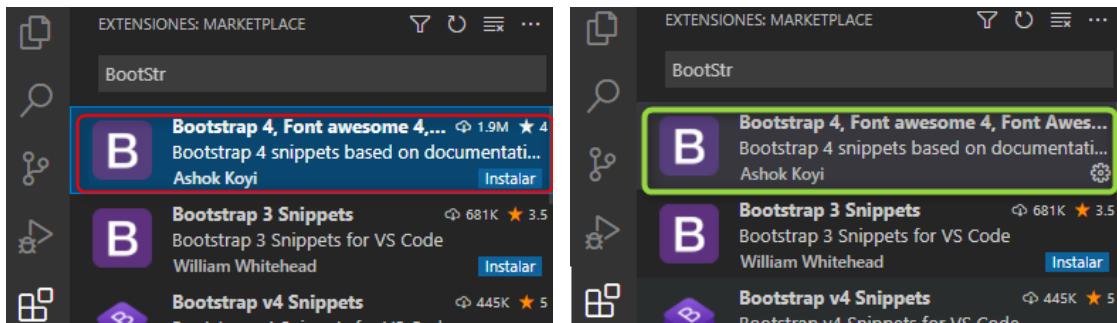


Fig. 3.81: Plugin 28 no instalado.

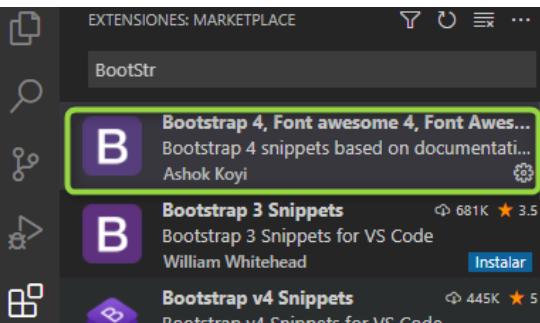


Fig. 3.82: Plugin 28 instalado correctamente.

- 14.29 – Node-Snippets (Fig. 3.83 y Fig. 3.84):

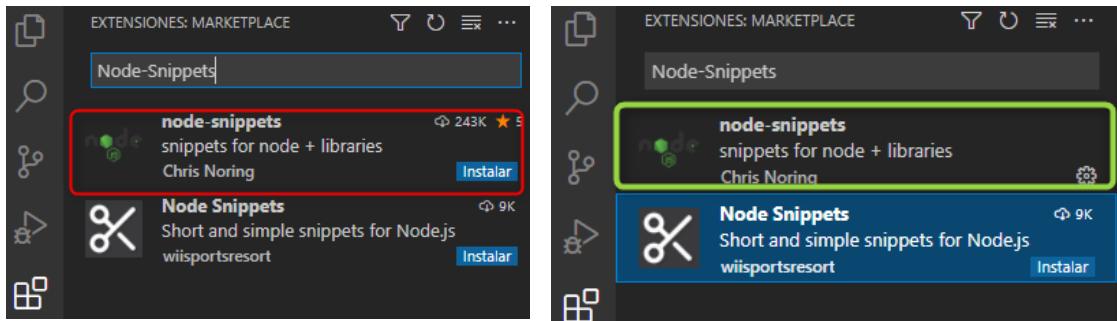


Fig. 3.83: Plugin 29 no instalado.

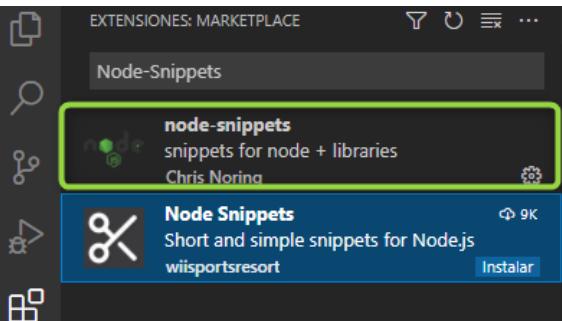


Fig. 3.84: Plugin 29 instalado correctamente.

- ES7 React/Redux/React-Native snippets (Fig. 3.85 y Fig. 3.86):

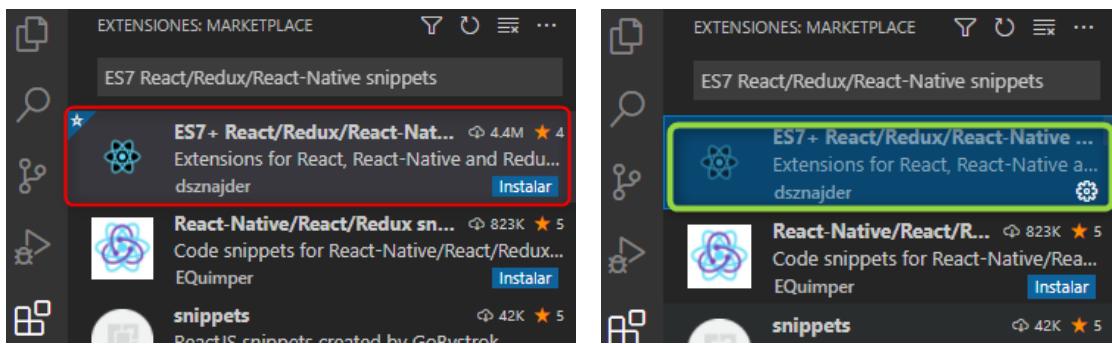


Fig. 3.85: Plugin 30 no instalado.

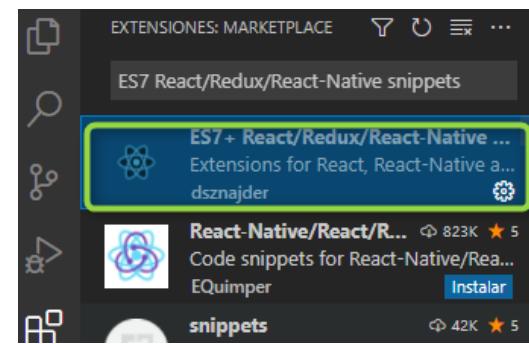


Fig. 3.86: Plugin 30 instalado correctamente.

- Moment.js snippets (Fig. 3.87 y Fig. 3.88):

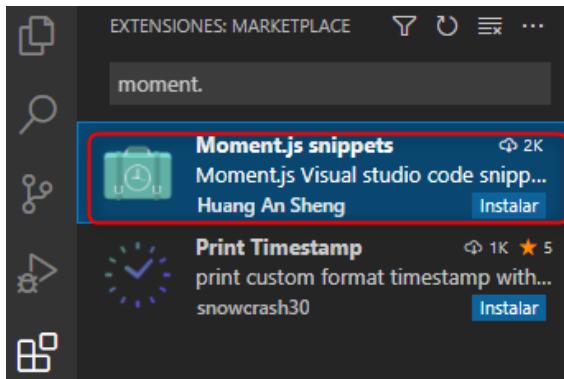


Fig. 3.87: Plugin 31 no instalado.

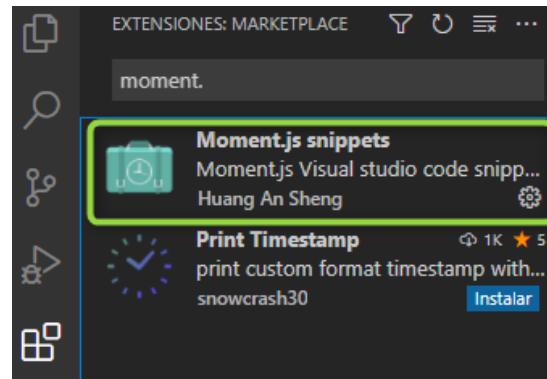


Fig. 3.88: Plugin 31 instalado correctamente.

- Node Debug (Fig. 3.89 y Fig. 3.90):

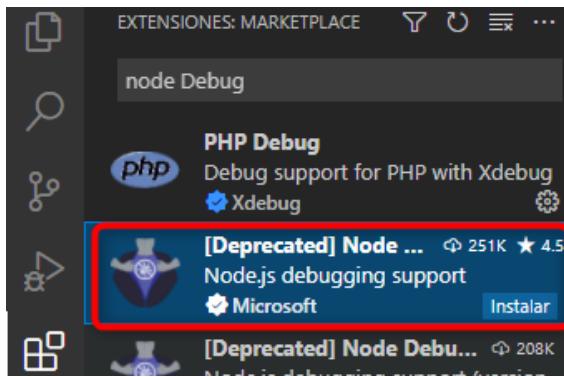


Fig. 3.89: Plugin 32 no instalado.

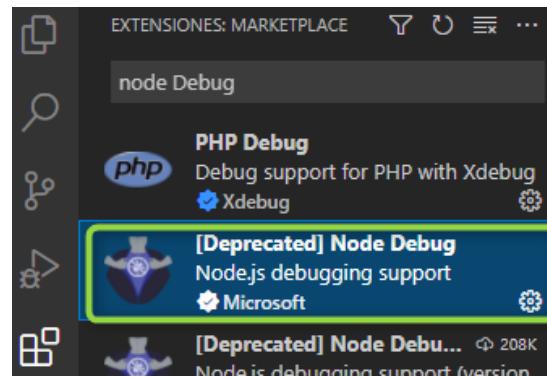


Fig. 3.90: Plugin 32 instalado correctamente.

Paso 11: Si has logrado realizar el tutorial hasta aquí sin presentar errores, entonces podemos decir que Visual Studio Code quedó correctamente instalado y configurado.

3.3 Descarga de proyecto prefabricado Spring Boot + Maven (Back-end)

Para este proyecto usaremos el framework de “Spring Boot”, con servidor Tomcat incrustado en su interior. Este servidor Tomcat integrado será el que hará las tareas de servidor HTTP.

El uso de Spring básico no se contempla como una solución eficiente: Spring Boot es una solución más fácil, rápida de montar y que tiene menos ficheros para configurar.

Entonces, descarguemos un paquete de Spring Boot con el servidor Tomcat embebido, siguiendo los siguientes pasos:

Paso 1:

Escribir en Google la búsqueda como “Spring Boot initializr”.

Paso 2:

De las opciones que salgan, clic en la coincidencia que coincide con esta web: <https://start.spring.io>

Paso 3:

Una vez llegué a la web anterior, verá algo similar a la Fig. 3.91.

A continuación, tendremos en cuenta la siguiente configuración básica:

- La versión más reciente de tipo NO SNAPSHOT (Stable) que haya disponible en esta web, de Spring Boot.
- El tipo de proyecto debe ser “Maven Project”.
- El lenguaje debe ser “Java”.
- El empaquetado (packagin) debe ser “Jar”.
- La versión Java será la de Java “11”.

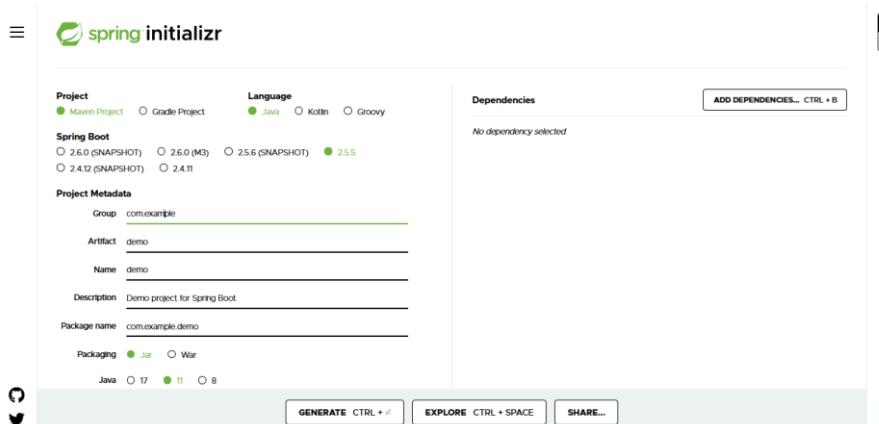


Fig. 3.91: Pantalla inicial de la página “Spring Boot Initializr”.

Paso 4:

Los detalles de la configuración serán los siguientes:

- | | |
|-------------------------|--|
| a. <i>Proyecto:</i> | <i>Maven Project</i> |
| b. <i>Spring Boot:</i> | <i>2.5.5 stable (o versión superior STABLE).</i> |
| c. <i>Language:</i> | <i>Java</i> |
| d. <i>Group:</i> | <i>com.miTutorialSpringBoot</i> |
| e. <i>Artefact:</i> | <i>demo</i> |
| f. <i>Name:</i> | <i>demo</i> |
| g. <i>Description:</i> | <i> proyecto demo de Spring Boot</i> |
| h. <i>Package name:</i> | <i>com.miTutorialSpringBoot.demo</i> |
| i. <i>Packaging:</i> | <i>Jar</i> |
| j. <i>Java:</i> | <i>11</i> |

Paso 5:

Cargar a la derecha la dependencia de BBDD “H2 Database”, desde botón “ADD DEPENDENCIES...”.

Pulsar en el botón inferior “GENERATE” y se nos descargará un ZIP.

La descarga del paquete Spring Boot, con las configuraciones anteriores, sería como se ve a continuación (Fig. 3.92):

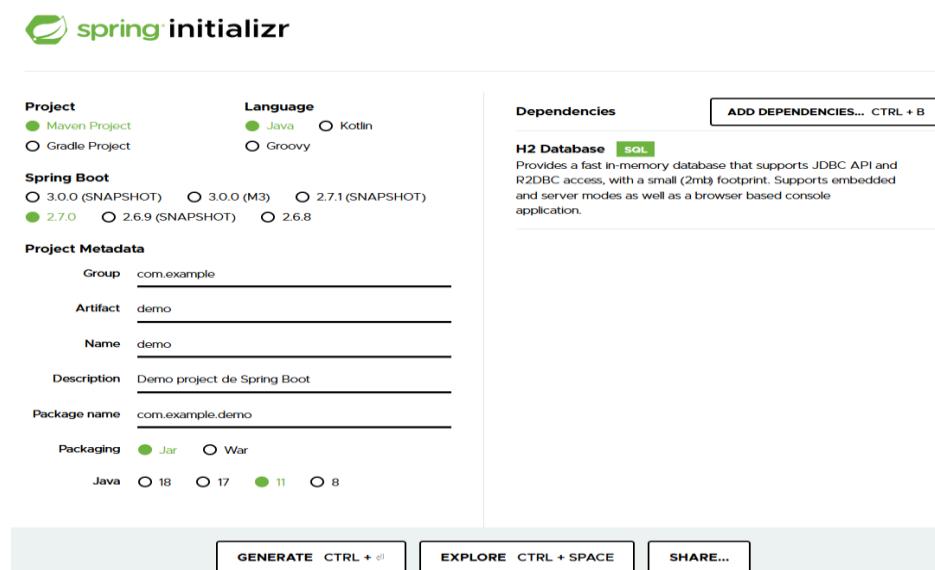


Fig. 3.92: Inicio de la Descarga del paquete en formato ZIP, desde navegador FireFox, con las configuraciones correctas.

Paso 6:

Descomprimir el fichero ZIP que se nos ha descargado en el directorio de “Descargar” (para caso de Windows 10).

Paso 7:

El fichero descomprimido se manda a la ruta donde se desee alojar permanentemente. En mi caso será en el directorio “Escritorio / TFM / ProyectoOficial” (Fig. 3.93).

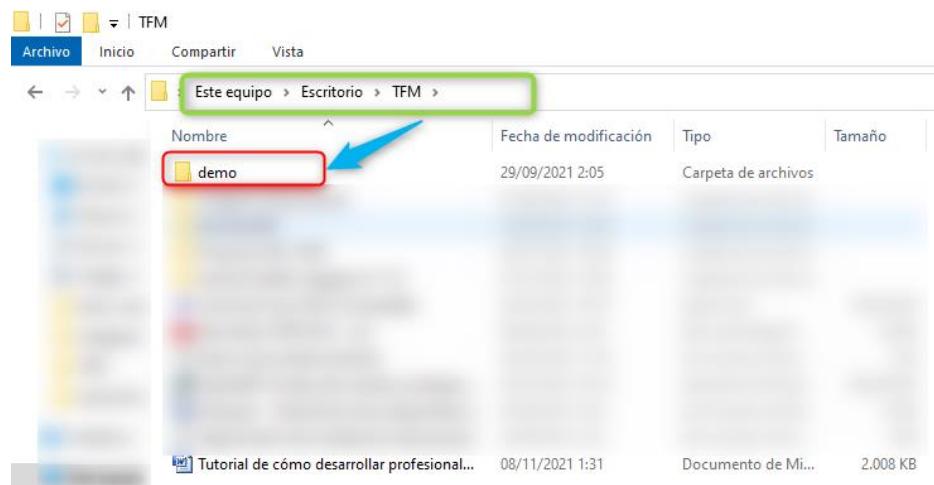


Fig. 3.93: Ruta donde moveremos el paquete del proyecto software con Maven y Spring Boot, recién descargado en formato ZIP y descomprimido.

Paso 8:

El proyecto alojado en “Escritorio / TFM” se abre desde Visual Studio Code, y deberemos tener en cuenta las 3 pantallas siguientes: (a) cargar el workspace de backend cuando no haya ningún directorio cargado aún; (b) pantalla inicial de Java Tool, pudiendo aparecer o no; (c) ir a sección de extensiones e instalar “Extension Pack for Java”.

- a. Cargar proyecto backend en el workspace cuando no haya ningún archivo aún:
 - Pulsar “Open Folder”, seleccionar carpeta “demo” antes descargada, y cargarla en el workspace (Fig. 3.94).

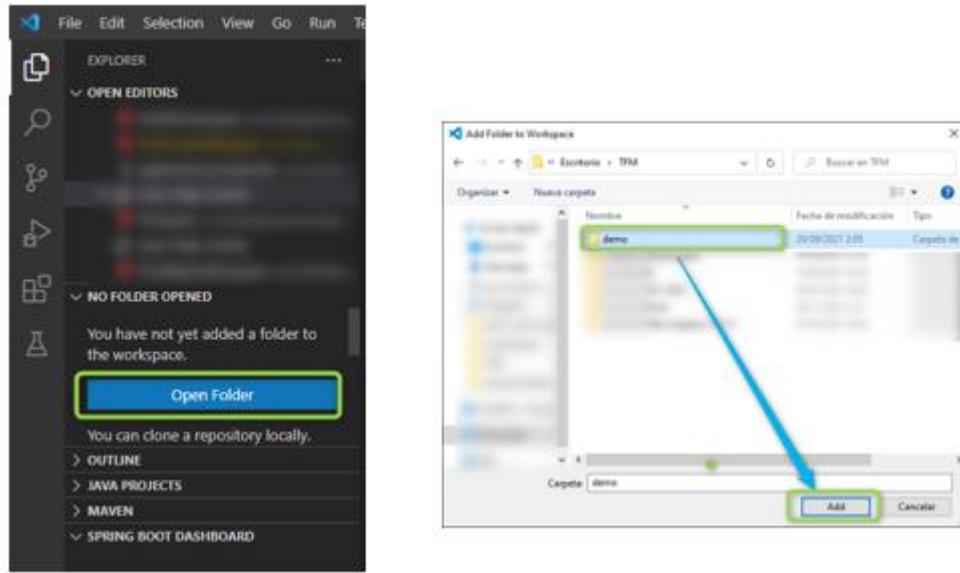


Fig. 3.94: Foto Izq.: Clicar en “Open Folder” o “Abrir Archivo” en la sección de archivos de Visual Studio Code. Foto Der.: Selección del directorio concreto del proyecto, y clicar en “Abrir”, operación que como resultado importará el proyecto en workspace de Visual Studio Code.

b. Pantalla inicial de Java Tool, pudiendo aparecer o no. Ver Fig. 3.95.

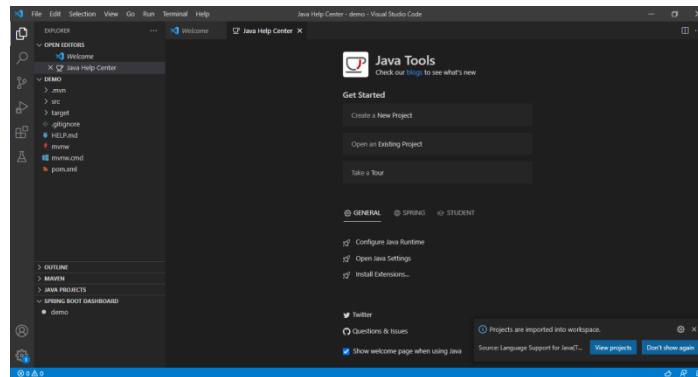


Fig. 3.95: Resultado de abrir el proyecto en Visual Studio Code.

c. Ir a sección de extensiones e instalar “Extension Pack for Java”.

Ver Fig. 3.96.

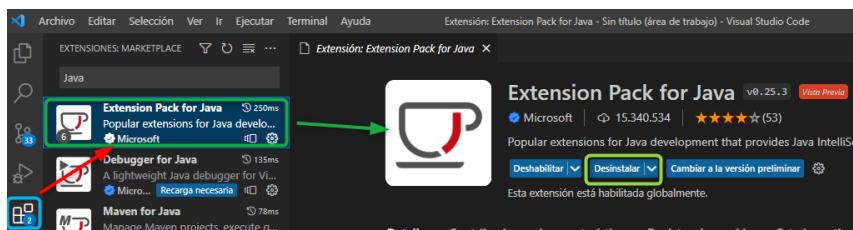


Fig. 3.96: Instalación de la herramienta que interpreta lenguaje Java en Visual Studio Code, llamada “Extensión Pack for Java”.

Paso 9:

Se necesitarán instalar en Visual Studio Code algunas extensiones, así como el intérprete de Java 11, etc.

- Eso será tratado en el apartado siguiente.

Paso 10:

Note en Visual Studio Code que se incorpora un menú en la parte izquierda, donde ya aparecen “MAVEN” y “SPRING BOOT DASHBOARD”. Eso significa que este paquete es el que lleva el Spring Boot (con HTTP embebido) y MAVEN.

Paso 11:

A continuación, en “Fig. 3.97”, puede verse un Zoom de la sección “SPRING BOOT DASHBOARD”, donde se resaltan 4 botones. Esta sección será usada para ejecutar en modo “Run” o en modo “Debug” la aplicación.

Las opciones que se resaltan son:

- “Start as...” en rojo
- “Start” en blanco
- “Refresh” en amarillo
- “Debuging” en rosa.

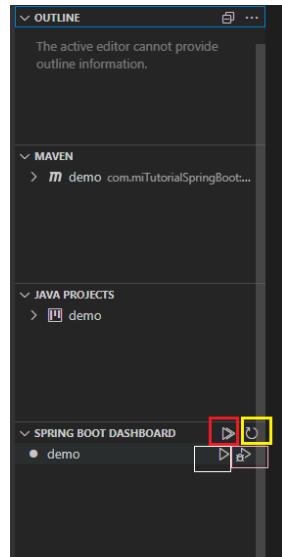


Fig. 3.97: Los 4 botones del debugger de Visual Studio Code, para modo PLAY y DEBUG.

Paso 12:

Revisamos que efectivamente coincidan los archivos que veo en el “Visual Studio Code” con el proyecto deseado desde el navegador de carpetas, comparando la imagen anterior (Fig. 3.96) con la siguiente (Fig. 3.98).

Este equipo > Escritorio > TFM > demo > demo			
Nombre	Fecha de modificación	Tipo	Tamaño
.mvn	29/09/2021 0:05	Carpetas de archivos	
.settings	29/09/2021 2:13	Carpetas de archivos	
src	29/09/2021 0:05	Carpetas de archivos	
target	29/09/2021 2:13	Carpetas de archivos	
.classpath	29/09/2021 2:13	Archivo CLASSPATH	2 KB
.gitignore	29/09/2021 0:05	Documento de te...	1 KB
.project	29/09/2021 2:13	Archivo PROJECT	1 KB
HELP.md	29/09/2021 0:05	Archivo MD	1 KB
mvnw	29/09/2021 0:05	Archivo	10 KB
mvnw	29/09/2021 0:05	Script de comand...	7 KB
pom	29/09/2021 0:05	Documento XML	3 KB

Fig. 3.98: Archivos del proyecto “demo”, recién cargado como workspace en Visual Studio Code.

Paso 13: Ahora cargamos nuestra área de trabajo, que será nuestro paquete prefabricado de Spring Boot Initializr. Si no sabemos cómo cargar nuestro workspace en Visual Studio Code puede verse las figuras Fig. 3.99, Fig. 3.100 y Fig. 3.101, donde tiene un tutorial del proceso guiado.

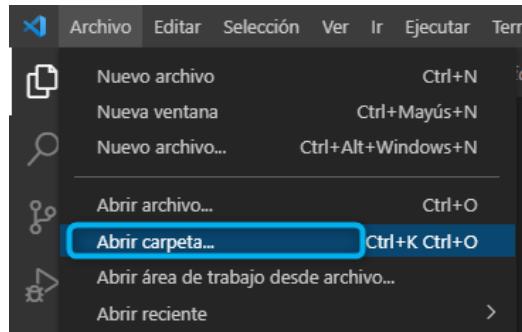


Fig. 3.99: Selección opción “Archivo” → “Abrir carpeta...”

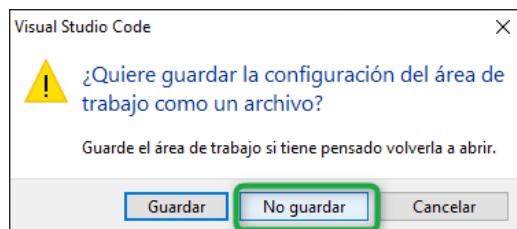


Fig. 3.100: No guardar como archivo el entorno de trabajo.

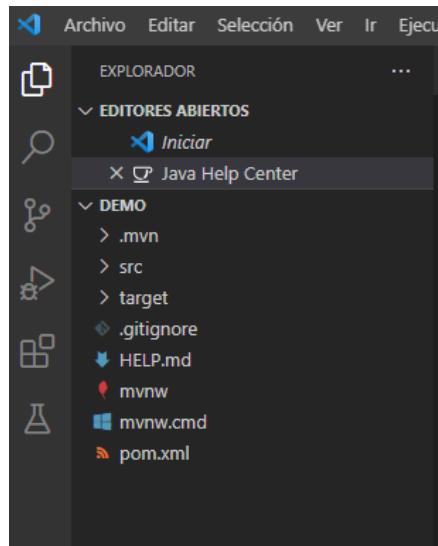


Fig. 3.101: Visualización del entorno de trabajo cargado en Visual Studio Code.

Paso 14: Ahora tenemos que ir al pom.xml y posiblemente nos salga una etiqueta de instalación dependencias para ANALITICS que nos autogenera el programa.

Si la etiqueta aparece (similar a la de Fig. 3.102), seleccionar “Install” y dejaremos esta herramienta instalada también en nuestro Visual Studio Code.

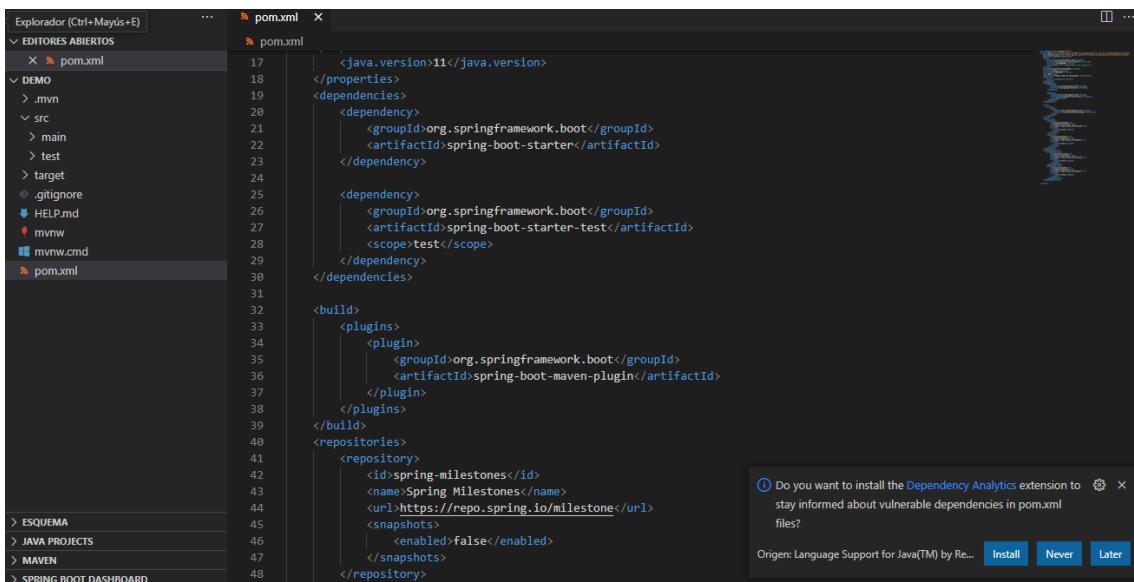


Fig. 3.102: Etiqueta de Analitycs autogenerada, para aceptar e instalar en Visual Studio Code.

Paso 15: Ahora debemos ir al fichero “pom.xml”, y en éste incluir 4 nuevas dependencias, conseguidas de la web: <https://mvnrepository.com>.

- La primera dependencia, se usará para temas del repositorio y RestController (permitir el servicio RESTFull en las peticiones). Podemos encontrar el código para esta dependencia en: “[Spring Boot Starter Web](#)” (ver Fig. 3.103).

License	Apache 2.0
Organization	Pivotal Software, Inc.
HomePage	https://spring.io/projects/spring-boot
Date	(Dec 22, 2021)
Files	pom (2 KB) jar (4 KB) View All
Repositories	Central
Used By	8,095 artifacts

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.6.2</version>
</dependency>
```

Fig. 3.103: Código oficial de la dependencia para Spring Boot Starter Web, versión 2.6.2.

- La segunda dependencia, se usará para que se pueda refrescar el servicio al instante cuando guarde una modificación en el código fuente. Podemos encontrar el código para esta dependencia en: “[Spring BootDevTools](#)” (ver Fig. 3.104).

The screenshot shows the Maven Repository website at <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools/2.6.2>. The page displays the following information:

- Indexed Artifacts (24.9M)**: A chart showing the number of indexed artifacts from 2006 to 2018, with a sharp increase starting around 2014.
- Popular Categories**: A sidebar listing various software categories such as Aspect Oriented, Actor Frameworks, Application Metrics, Build Tools, Bytecode Libraries, Command Line Parsers, Cache Implementations, Cloud Computing, Code Analyzers, Collections, Configuration Libraries, and Core Utilities.
- Spring Boot DevTools > 2.6.2**: The main section, featuring a power icon and the text "Spring Boot Developer Tools".
- License**: Apache 2.0
- Organization**: Pivotal Software, Inc.
- HomePage**: <https://spring.io/projects/spring-boot>
- Date**: (Dec 22, 2021)
- Files**: pom (2 KB) | jar (226 KB) | View All
- Repositories**: Central
- Used By**: 1,056 artifacts
- Dependency Coordinates**: Maven, Gradle, Gradle (Short), Gradle (Kotlin), SBT, Ivy, Grape, Leiningen, Buildr

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <version>2.6.2</version>
</dependency>
```

Fig. 3.104: Código oficial de la dependencia para Spring Boot DevTools, versión 2.6.2.

- La tercera dependencia, se usará para temas referentes a la BBDD. Nuestra Base de Datos será una Base de Datos incrustada (integrada) y volátil, y por eso cuando el servidor se pare o se apague, entonces también la BBDD perderá su memoria y los datos almacenados en ella.
Esta dependencia la podemos encontrar en: “[H2 DatabaseEngine](#)” (ver Fig. 3.105).

The screenshot shows the Maven Repository page for the artifact com.h2database/h2/2.0.206. The top navigation bar includes links for Maven, Gradle, Gradle (Short), Gradle (Kotlin), SBT, Ivy, Grape, Leiningen, and Buildr. Below the navigation is a table with the following data:

Categories	Embedded SQL Databases
HomePage	https://h2database.com
Date	(Jan 04, 2022)
Files	pom (1 KB) jar (2.4 MB) View All
Repositories	Central
Used By	6,808 artifacts

Below the table is a code editor window containing the XML dependency declaration:

```
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.0.206</version>
    <scope>test</scope>
</dependency>
```

A note at the bottom of the code editor says "Exclude comment with link to declaration".

Fig. 3.105: Código oficial de la dependencia para Base de Datos “H2”, versión 2.0.206.

- La cuarta dependencia, se va a usar para incluir temas de ORM's (mapeo de Objetos Relacionales) en nuestra aplicación, juntamente con el protocolo JPA.
- Esta dependencia la podemos encontrar en: “[Spring Boot Starter Data JPA](#)” (ver Fig. 3.106).

The screenshot shows the Maven Repository page for the artifact org.springframework.boot/spring-boot-starter-data-jpa/2.6.2. The top navigation bar includes links for Maven, Gradle, Gradle (Short), Gradle (Kotlin), SBT, Ivy, Grape, Leiningen, and Buildr. Below the navigation is a table with the following data:

Indexed Artifacts (25.2M)	
Popular Categories	Aspect Oriented, Actor Frameworks, Application Metrics, Build Tools, Bytecode Libraries, Command Line Parsers, Cache Implementations, Cloud Computing, Code Analyzers, Collections, Configuration Libraries, Core Utilities
Spring Boot Starter Data JPA > 2.6.2	Starter for using Spring Data JPA with Hibernate
License	Apache 2.0
Organization	Pivotal Software, Inc.
HomePage	https://spring.io/projects/spring-boot
Date	(Dec 22, 2021)
Files	pom (3 KB) jar (4 KB) View All
Repositories	Central
Used By	1,602 artifacts

Below the table is a code editor window containing the XML dependency declaration:

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.6.2</version>
</dependency>
```

Fig. 3.106: Código oficial de la dependencia para Spring Boot Starter Data JPA, v. 2.6.2.

Paso 16: Si hemos incluido las porciones de código que hemos encontrado en cada una de las 4 dependencias del paso 11, en el fichero “pom.xml”, habrán sido incluidos junto al resto de código XML. Por si hubiera alguna duda, revisar las 2 siguientes Figuras (Fig. 3.107 y Fig. 3.108) para comprobar la correcta integración de las 4 nuevas dependencias:

- Inclusión de las dependencias de “Spring Boot Starter” y “Spring Boot Devtools” (Fig. 3.107):

```

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.6.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <version>2.6.2</version>
</dependency>

```

Fig. 3.107: Código incluido en el pom.xml de las dependencias “Spring Boot Starter” y “Spring Boot Devtools”.

- Inclusión de las 2 dependencias restantes, que afectan a temas de persistencia de los datos, para el protocolo JPA y la Bases de Datos volátil H2 (Fig. 3.108):

```

<!-- 2 DEPENDENCIAS USADAS PARA EL TEMA DE PERSISTENCIA DE DATOS Y BASE DE DATOS -->
<!-- para la base de datos h2 en memoria -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- el JPA -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

```

Fig. 3.108: Código incluido en el pom.xml de las dependencias “Spring Boot Starter Data JPA” y “h2”.

Paso 17: Ahora cargamos las nuevas dependencias juntas a las ya incluidas, en nuestro proyecto Maven. Para ello, por favor, vaya en Visual Studio Code a la sección “SPRING BOOT DASHBOARD” → botón PLAY, como se muestra en la figura (Fig. 3.109).

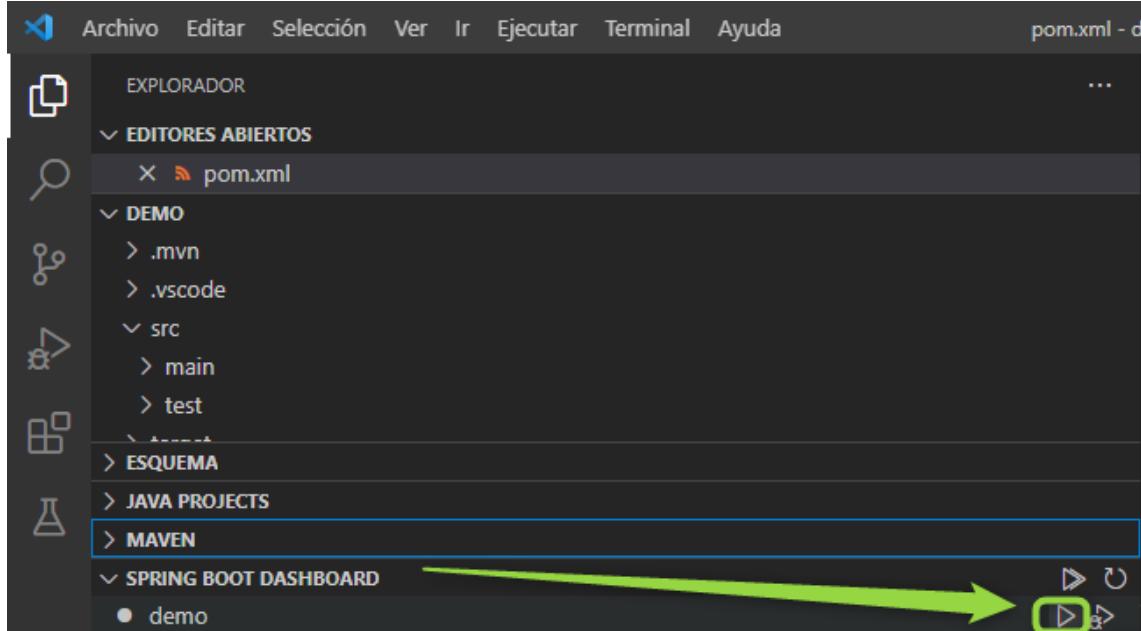


Fig. 3.109: Primera ejecución del servidor Tomcat integrado y carga de las nuevas dependencias.

Paso 18: Comprobar que, el simple hecho de ejecutar el servidor y tenerlo arrancado habilitará la sección MAVEN. También, en Fig. 3.110, estarán cargadas las nuevas dependencias (parte izquierda) y podremos ver las trazas de las descargas recién realizadas de las nuevas dependencias incluidas (parte derecha).

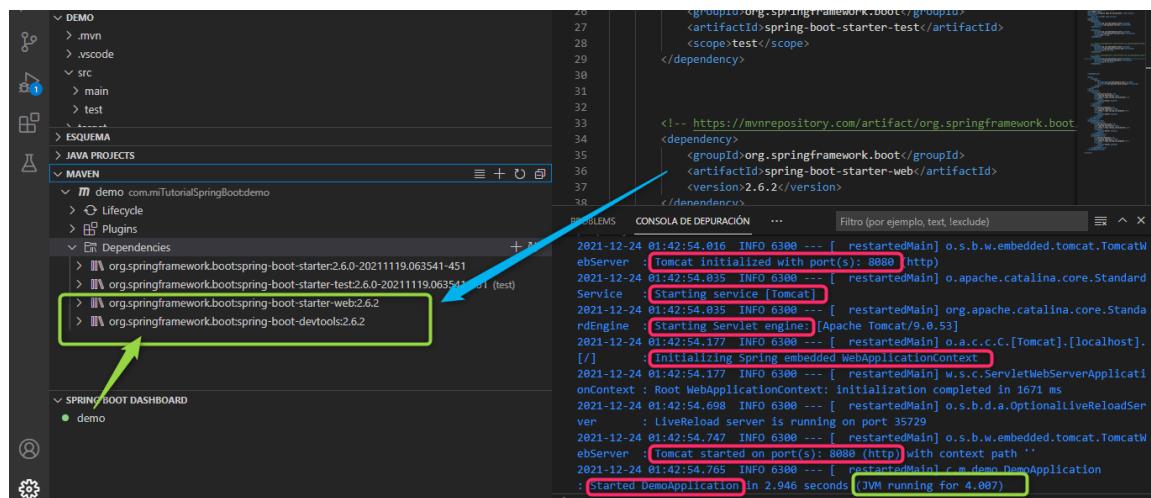


Fig. 3.110: Nuevas dependencias incluidas y descarga en tiempo real de las mismas.

Paso 19: Una vez llegamos a este punto, tenemos todo el IDE de Visual Studio Code preparado para usarlo para nuestro desarrollo de software. Solamente faltaría montar la estructura del proyecto y codificar el mismo.

Por tanto, por último, pararemos la ejecución del servidor Tomcat (Fig. 3.111):

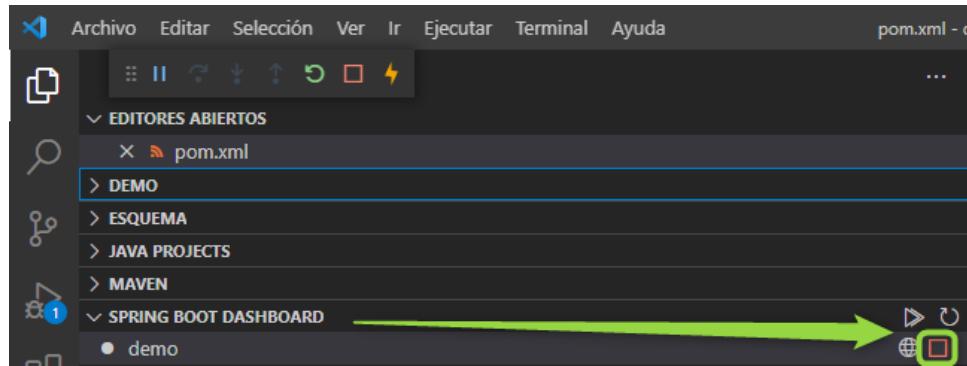


Fig. 3.111: Parada de nuestro servidor Tomcat.

Paso 20: Comprobamos que el servidor Tomcat está parado definitivamente cuando los iconos de la imagen anterior cambien, como por ejemplo, desapareciendo el botón de STOP y reapareciendo los botones de PLAY y DEBUG (ver Fig. 3.112).

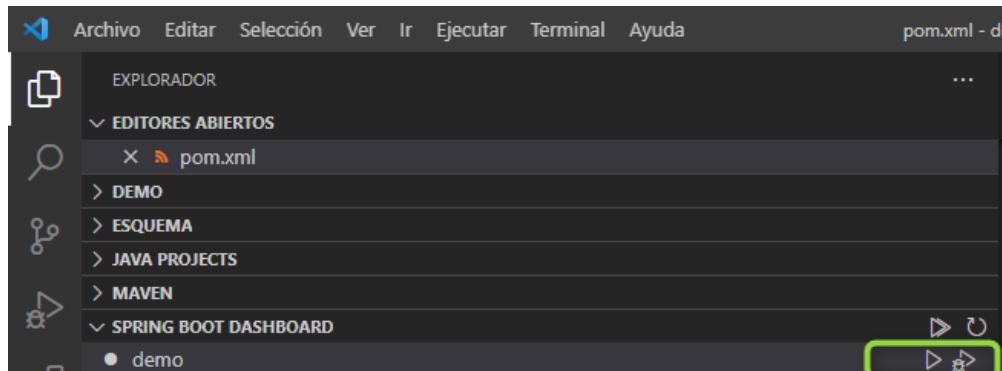


Fig. 3.112: Recuperación de algunos botones, en sección SPRING BOOT DASHBOARD, como Play o Debug cuando volvemos a tener el servidor Tomcat parado completamente.

3.4 Instalación de Maven en Windows O.S.

En este apartado se dará el tutorial de instalación completa de la herramienta MAVEN, utilizada para la gestión de dependencias de proyectos Java (como en nuestro caso).

En el apartado anterior se explicó que usaremos como framework la herramienta Spring Boot y que ésta tendrá incrustado (embebido) un servidor Tomcat que se usará como

servidor HTTP. Lo que no se mencionó es que tanto Spring Boot como el servidor Tomcat están contenidos en un proyecto Maven que actúa como contenedor de estos.

Por tanto, el fichero ZIP que hemos descargado durante el tutorial del apartado anterior, una vez descomprimido, en realidad era un proyecto Maven que venía prefabricado con las siguientes características en su interior:

1. Estructura básica de un proyecto Maven (como fichero “pom.xml” y otras características).
2. Framework de Spring Boot dentro del proyecto Maven.
3. Servidor Tomcat incrustado (embebido).

No obstante, con la sola descarga del proyecto prefabricado no tenemos todas las funcionalidades de Maven. Para completar todas las funcionalidades del proyecto Maven a nivel global de nuestro equipo (PC) se procede a explicar los siguientes pasos de este tutorial.

Paso 1:

Acceder al link oficial del proyecto MAVEN que se deja a continuación.

<https://maven.apache.org/>

Puede verse un ejemplo de la página web que encontrará si se revisa la siguiente imagen (Fig. 3.113).

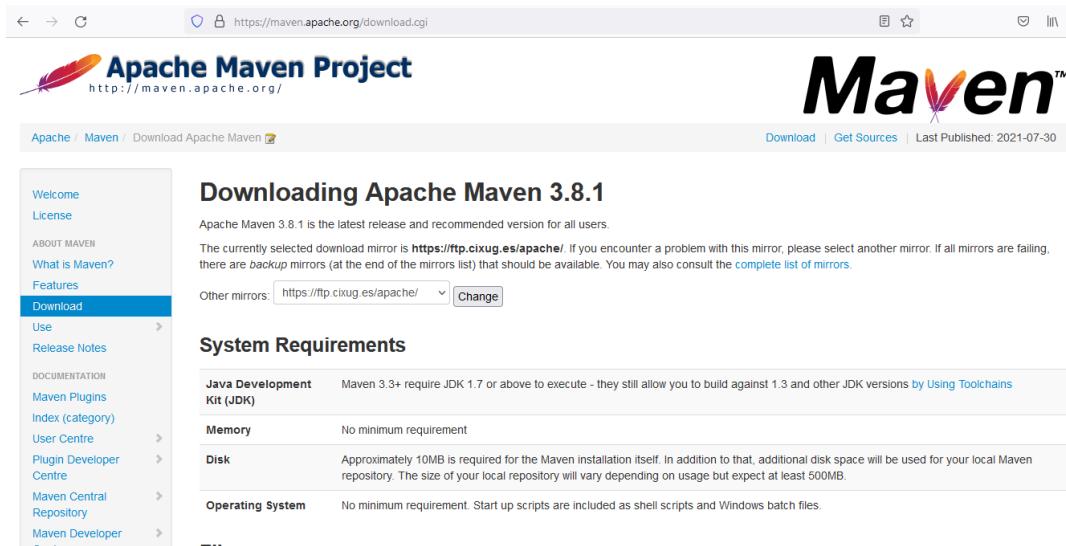


Fig. 3.113: Web oficial de Apache Maven.

Paso 2:

Realizar la descarga del fichero Apache-Maven más conveniente para nuestro caso:

Una vez accedamos a la web oficial correctamente, ir hacia abajo en esta misma página web y seleccionar el link de descarga del paquete ZIP para Windows (ver ejemplo en la Fig. 3.114, señalado en rojo):

Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [installation instructions](#). Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to [verify the signature](#) of the release bundles against the public [KEYS](#) used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.8.1-bin.tar.gz	apache-maven-3.8.1-bin.tar.gz.sha512	apache-maven-3.8.1-bin.tar.gz.asc
Binary zip archive	apache-maven-3.8.1-bin.zip	apache-maven-3.8.1-bin.zip.sha512	apache-maven-3.8.1-bin.zip.asc
Source tar.gz archive	apache-maven-3.8.1-src.tar.gz	apache-maven-3.8.1-src.tar.gz.sha512	apache-maven-3.8.1-src.tar.gz.asc
Source zip archive	apache-maven-3.8.1-src.zip	apache-maven-3.8.1-src.zip.sha512	apache-maven-3.8.1-src.zip.asc

Fig. 3.114: Selección de la descarga del paquete de Apache Maven adecuado para nuestro caso, y descarga del paquete en formato ZIP.

NOTA: para poder usar MAVEN necesitaremos Java 7 o superior. Por eso nosotros vamos a usar Java 11.

Paso 3:

Cuando nuestra descarga termina, vamos a nuestro directorio de “Descargas” en Windows, y descomprimir el ZIP recién descargado (Fig. 3.115).



Fig. 3.115: Descompresión del ZIP descargado. Imagen que muestra primero ZIP y después fichero descomprimido.

Paso 4:

La carpeta recién descomprimida en el directorio “Descargas”, la vamos copiar y pegar en el directorio siguiente: C:/Archivos de Programa.

Para más detalles y a modo de ejemplo puede verse la imagen de Fig. 3.116.

Este equipo > Disco local (C:) > Archivos de programa

Nombre	Fecha de modificación	Tipo
AdoptOpenJDK	29/05/2021 20:20	Carpeta de archivos
apache-maven-3.8.1-bin	31/07/2021 18:57	Carpeta de archivos
Common Files	09/05/2021 22:22	Carpeta de archivos
Git	31/07/2020 2:26	Carpeta de archivos

Fig. 3.116: Movimiento del archivo .bin de Apache Maven a la carpeta de “Archivos de Programa”.

Paso 5:

Entramos en la carpeta recién pegada (“apache-maven 3.81-bin” para nuestro caso).

Arriba nos aparece la ruta donde nos encontramos ubicados, así que copiamos la ruta que nos indique arriba (ver Fig. 3.117).

C:\Program Files\apache-maven-3.8.1-bin\bin

Nombre	Fecha de modificación	Tipo
m2.conf	07/11/2019 12:32	Archivo CONF
mvn	07/11/2019 12:32	Archivo
mvn	07/11/2019 12:32	Script de comando...
mvnDebug	07/11/2019 12:32	Archivo
mvnDebug	07/11/2019 12:32	Script de comando...
mvnyjp	07/11/2019 12:32	Archivo

Fig. 3.117: Archivos que contiene en el interior la carpeta que hemos pegado de Maven en Archivos de programa.

Paso 6:

Ahora, sobre un sistema operativo Windows 10 (o Windows 11), pulsa la tecla WINDOWS de tu teclado y escribe “Variables de”.

Deberían aparecer para elegir las opciones de “...variables de entorno del sistema” y “...variables de entorno de esta cuenta” (ver Fig. 3.118).

Elige la opción llamada “Editar las variables de entorno del sistema”.



Fig. 3.118: Búsqueda de “Variables de entorno del sistema”, desde el explorador de programas de Windows.

Nota: Las variables del sistema son las que nos van a interesar, porque una vez configuradas estas variables, estas afectan a todos los usuarios del sistema a la vez.

Paso 7:

Una vez seleccionado “Editar las variables de entorno del sistema”, vamos a añadir la ruta donde está nuestra información de Maven a las variables del sistema.

Debemos hacer las siguientes 5 cosas (pasos “a”, “b”, “c”, “d” y “e”).

- Nos aparece una nueva ventana.

Debemos seleccionar la pestaña llamada “Opciones Avanzadas”.

Pulsar el botón “Variables de entorno...”.

Ejemplo visual en Fig. 3.119.

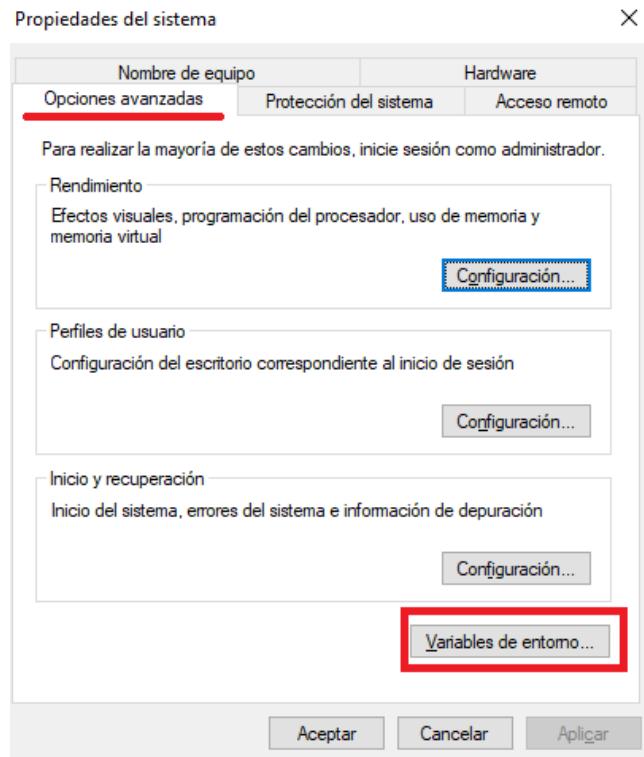


Fig. 3.119: Nueva ventana, donde nos dirigimos a “Opciones Avanzadas” → Variables de entorno...”

- b) Se nos abre otra ventana al clicar en el botón “Variables de entorno...”.
En esta nueva ventana, en la sección “Variables de usuario para Windows”, elegimos la opción “Path”, como se indica en Fig. 3.120.
Dar doble clic sobre la variable “Path”.

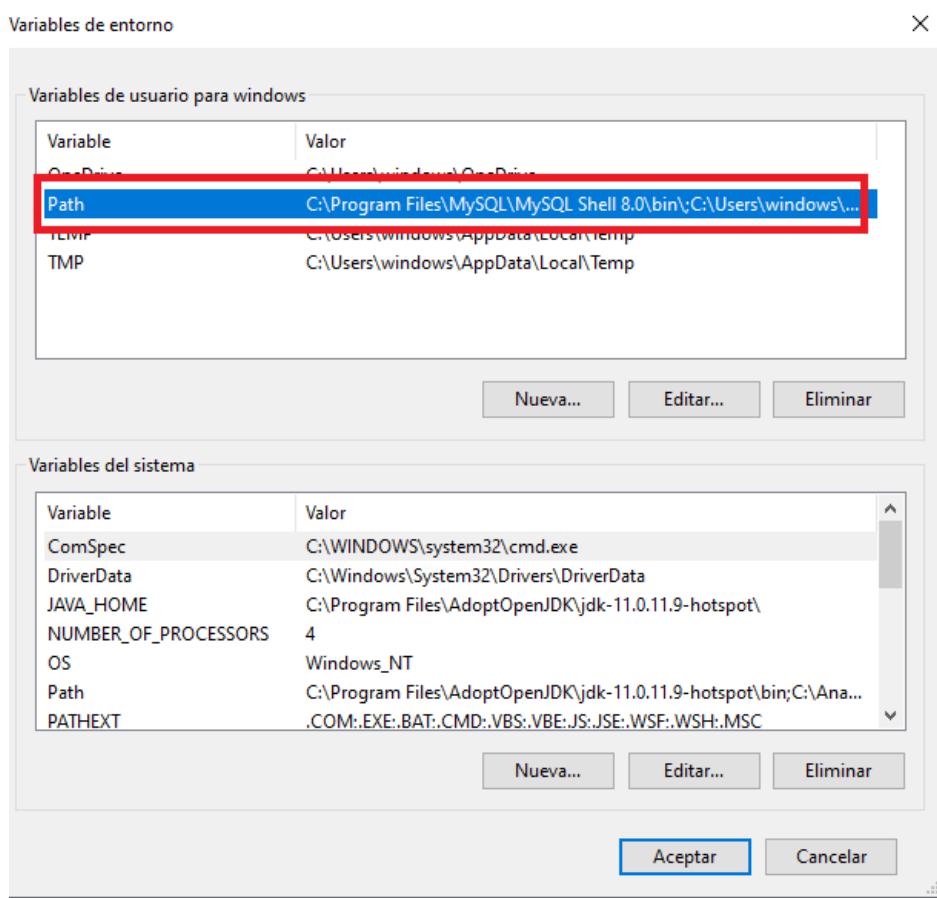


Fig. 3.120: Elección de variable de usuario llamada “Path”.

- c) Al hacer doble clic en la variable Path, se abre otra ventana.
Debemos clicar en la primera línea en blanco disponible, de la lista de rutas que se encuentran ya incluidas para esta variable (Path).
Ahora, en esta nueva línea en blanco pegamos la ruta de la carpeta Maven que habíamos copiado antes (y que tenemos en el portapapeles).
Una vez terminemos la edición, pulsar botón “Aceptar” y la ventana se cerrará (ver Fig. 3.121).

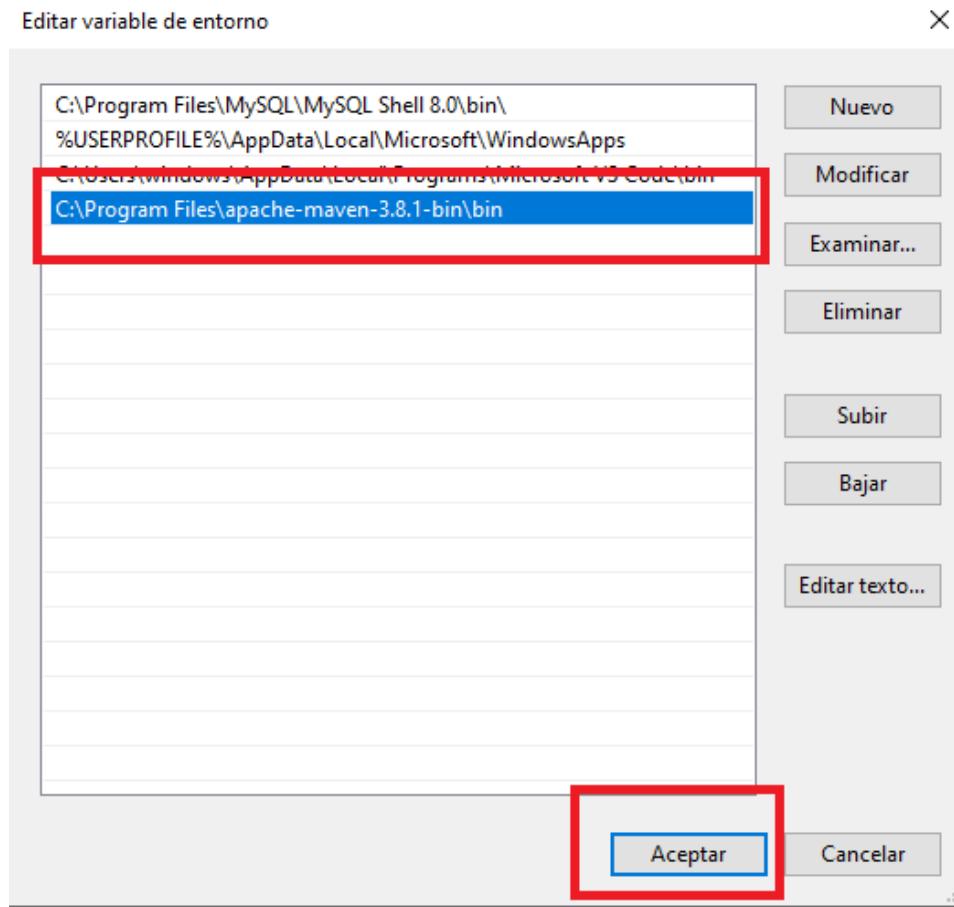


Fig. 3.121: Configuración de la variable de usuario llamada “Path”, incluyendo la nueva ruta señalada en la imagen (o similar).

NOTA: acabamos de configurar la ruta de ejecución de Maven.

- d) Ahora crearé la nueva variable de entorno que ejecutará Maven cuando realicemos la llamada correspondiente por comandos.
Para ello, dar a “Nueva en la zona de variables de la cuenta” (o zona superior) y se nos abrirá una nueva ventana que debemos configurar como se indica en la siguiente figura (ver Fig. 3.122).

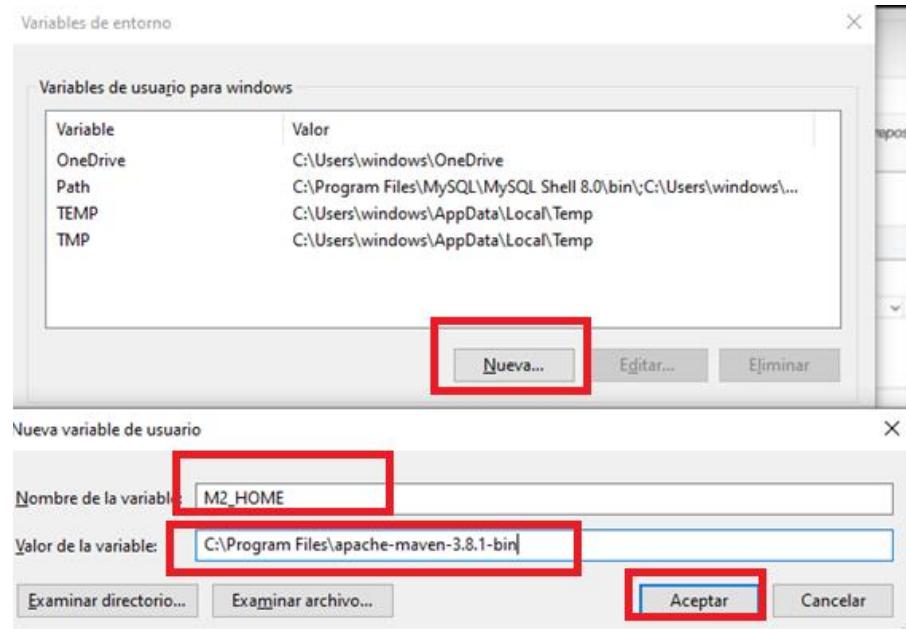


Fig. 3.122: Creación de nueva variable de entorno, a nivel de usuario, para ejecutar Maven cuando usemos el comando correspondiente en una consola de comandos Windows o cualquier otra alternativa.

e) El resultado con la nueva variable, queda así (Fig. 3.123):

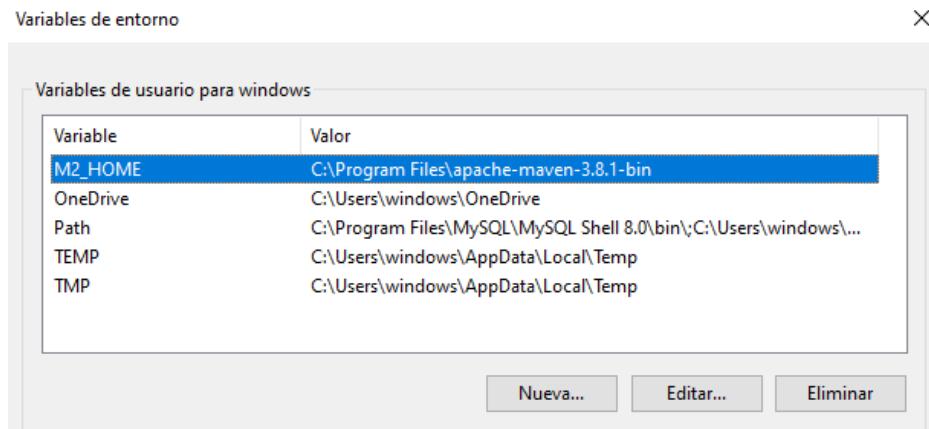


Fig. 3.123: Nueva variable de entorno creada.

NOTA: La configuración de las variables de entorno y de sistema se terminó correctamente. Para cerrar todas las ventanas guardando los cambios de las variables de entorno, aceptamos todas las ventanas necesarias y aplicamos cambios hasta cerrar cada una de las posibles ventanas anteriores.

Paso 8:

Abrimos la consola de comandos de Windows:

- Ahora podremos usar Maven desde la línea de comandos de Windows.
- Para COMPILEAR un proyecto de MAVEN por comandos:
 - 1º) Abrir una consola de comandos en Windows pulsando Windows + R y en la ventana emergente llamada “Abrir” deberemos escribir “cmd” y dar en ACEPTAR (Fig. 3.124).

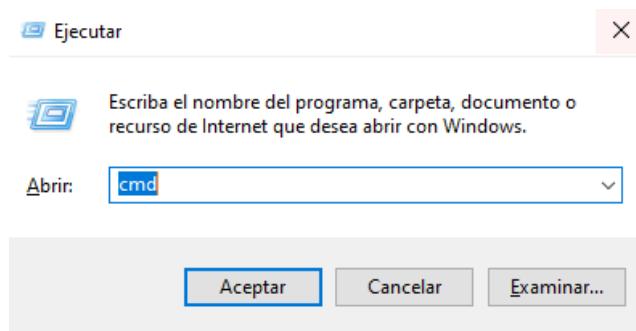


Fig. 3.124: Nueva ventana de ejecución, escritura de “cmd” y Aceptar para abrir nueva consola de comandos Windows.

2º) Se nos abrirá una consola de Windows en negro (Fig. 3.125).

A screenshot of a Windows Command Prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The window itself displays the following text:
Microsoft Windows [Versión 10.0.19042.1110]
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\windows>

Fig. 3.125: Nueva consola de comandos Windows abierta.

3º) Comprobamos que Maven se reconoce como comando, desde línea de comandos.

Para ello usaremos el comando “mvn --version”, y al usarlo deberemos ver lo siguiente (Fig. 3.126):

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.19042.1110]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\windows>mvn --version
Apache Maven 3.8.1 (05c21c65bdfed0f71a2f2ada8b84da59348c4c5d)
Maven home: C:\Program Files\apache-maven-3.8.1-bin\bin\
Java version: 11.0.11, vendor: AdoptOpenJDK, runtime: C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\windows>
```

Fig. 3.126: Ejecución del comando que nos permite conocer versión del Maven instalado en nuestro equipo.

4º) Ahora usaremos el comando “cd” en esta consola, y vamos a ir a la ruta del directorio donde tengamos el workSpace de nuestro proyecto software. Después entrar a la carpeta “demo” como se muestra en la siguiente imagen (Fig. 3.127).

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.19042.1110]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\windows> cd "Desktop\TFM\Proyecto 2021-2022"
C:\Users\windows\Desktop\TFM\Proyecto 2021-2022> cd demo
C:\Users\windows\Desktop\TFM\Proyecto 2021-2022\demo>
```

Fig. 3.127: Movimiento dentro del CMD de Windows al interior de la carpeta “demo” del proyecto.

5º) Probaremos que efectivamente tenemos los archivos de las rutas indicadas en CMD, visualizando esta vez desde la navegación de carpetas de Windows (véase Fig. 3.128 y Fig. 3.129).

TFM > Proyecto 2021-2022			
Nombre	Fecha de modificación	Tipo	Tamaño
.vscode	04/07/2021 19:28	Carpeta de archivos	
demo	29/05/2021 20:30	Carpeta de archivos	

Fig. 3.128: Abrimos la ruta del workspace del proyecto Maven al que nos estamos moviendo en CMD.

Nombre	Fecha de modificación	Tipo	Tamaño
.mvn	29/05/2021 20:21	Carpeta de archivos	
.settings	29/05/2021 20:23	Carpeta de archivos	
.vscode	29/05/2021 20:41	Carpeta de archivos	
src	29/05/2021 20:21	Carpeta de archivos	
target	29/05/2021 20:23	Carpeta de archivos	
.classpath	29/05/2021 20:23	Archivo CLASSPATH	2 KB
.gitignore	29/05/2021 20:21	Documento de te...	1 KB
.project	29/05/2021 20:23	Archivo PROJECT	1 KB
HELP.md	29/05/2021 20:21	Archivo MD	1 KB
mvnw	29/05/2021 20:21	Archivo	10 KB
mvnw	29/05/2021 20:21	Script de comand...	7 KB
pom	04/07/2021 18:29	Documento XML	3 KB

Fig. 3.129: Visualizamos todos los archivos del proyecto Maven al que nos hemos movido desde CMD.

Paso 9:

Con todo ya configurado e instalado, referente a MAVEN...

- 9.1) Acceder desde “cmd” al directorio “.../demo” (el cual contiene el proyecto compilado de Maven) > Fig. 3.127.
- 9.2) Realizar una operación CLEAN para eliminar el subdirectorio y los archivos que este contiene, de nuestro proyecto Maven. Para ello, véase Fig. 3.130.
- 9.3) Después de ejecutar la operación CLEAN sobre el proyecto Maven a travé de comandos de consola. El subdirectorio Target ya no estará. Véase Fig. 3.131.

```
C:\Users\windows\Desktop\TFM\Proyecto 2021-2022\demo>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:demo >-----
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/3.1.0/maven-clean-plugin-3.1.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/3.1.0/maven-clean-plugin-3.1.0.pom (5.2 kB at 11 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/31/maven-plugins-31.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/31/maven-plugins-31.pom (10 kB at 104 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/31/maven-parent-31.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/31/maven-parent-31.pom (43 kB at 314 kB/s)
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ demo ---
[INFO] Deleting C:\Users\windows\Desktop\TFM\Proyecto 2021-2022\demo\target
[INFO]
[INFO] -----[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.990 s
[INFO] Finished at: 2021-07-31T19:37:21+02:00
[INFO] -----
```

Fig. 3.130: Ejecución de comando “mvn clean”, dentro del proyecto Maven y eliminación de los compilados y Target’s.

Nombre	Fecha de modificación	Tipo	Tamaño
.mvn	29/05/2021 20:21	Carpeta de archivos	
.settings	29/05/2021 20:23	Carpeta de archivos	
.vscode	29/05/2021 20:41	Carpeta de archivos	
src	29/05/2021 20:21	Carpeta de archivos	
.classpath	29/05/2021 20:23	Archivo CLASSPATH	2 KB
.gitignore	29/05/2021 20:21	Documento de te...	1 KB
.project	29/05/2021 20:23	Archivo PROJECT	1 KB
HELP.md	29/05/2021 20:21	Archivo MD	1 KB
mvnw	29/05/2021 20:21	Archivo	10 KB
mvnw	29/05/2021 20:21	Script de comand...	7 KB
pom	04/07/2021 18:29	Documento XML	3 KB

Fig. 3.131: Visualización de nuevo de los archivos que contiene el directorio Maven, este vez sin el subdirectorio Target, como consecuencia de la ejecución del comando “mvn clean”.

- 9.4) Por último, volvemos a compilar el proyecto Maven:
 Usaremos el comando desde “cmd” llamado “**mvn compile**”. Esto
 compilará el proyecto Maven y se volverá a ver la carpeta TARGET en
 nuestro directorio “demo”.

3.5 Instalación del cliente HTTP

El cliente HTTP no tiene mayor dificultad, ya que tan solo consiste en descargarse el programa portable de “Insomnia.Core-2021.5.3” en el directorio donde guardes el proyecto de Visual Studio Code.

La web oficial de descargas para el cliente HTTP RestFull es la siguiente (Fig. 3.132):

<https://insomnia.rest/products/insomnia>

Podrás ver una web como esta:

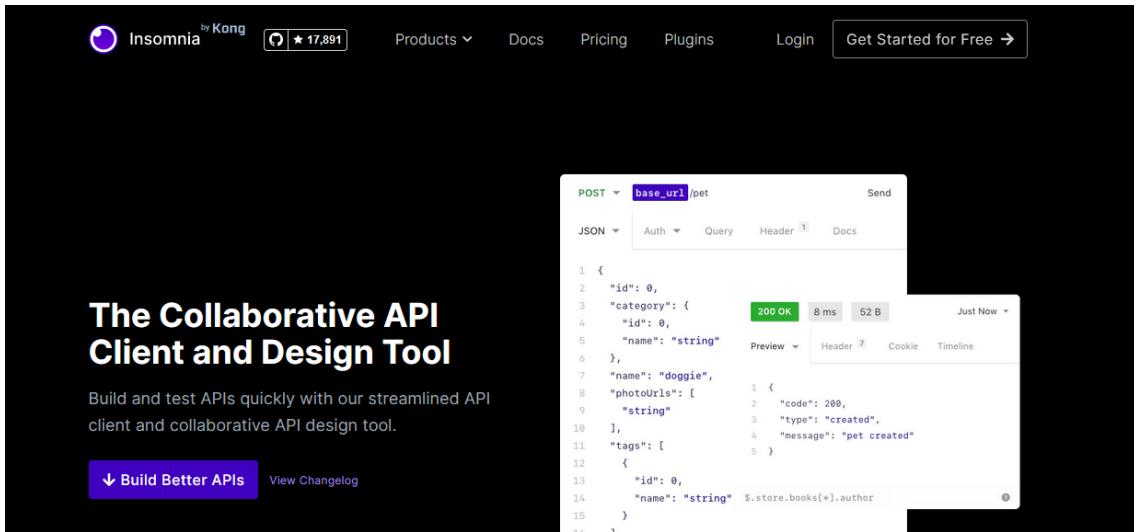


Fig. 3.132: Imagen de la web oficial del programa Insomnia Core, que usaremos como cliente HTTP.

Debemos ir ahora a la parte superior de esta web, botón “Pricing” (revisar Fig. 3.45). Nos aparecerá una web con las opciones de selección de tarifas disponibles, se seleccionará la opción FREE \$0 e iniciaremos la descarga gratuita pulsando el botón “Download App” que se indica en la imagen (Fig. 3.133).

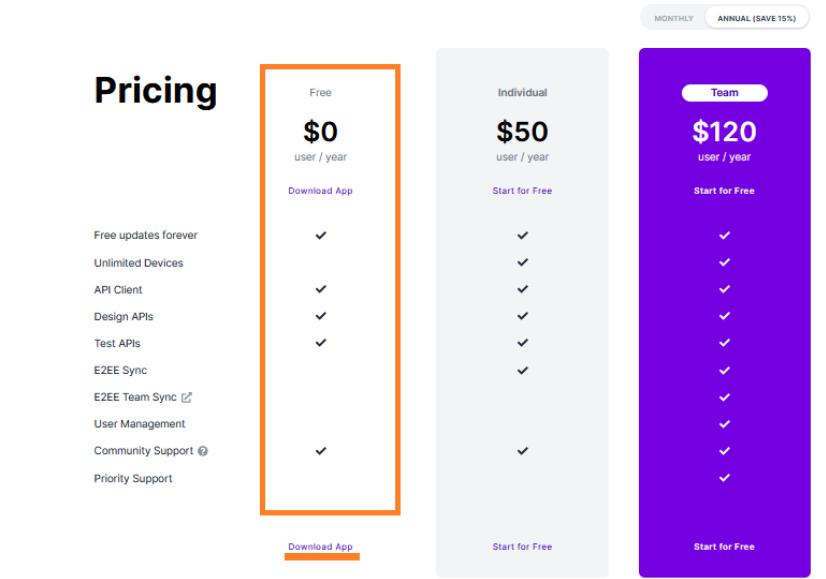


Fig. 3.133: Selección de tarifas disponibles para Insomnia Core y descarga gratuita.

Antes de iniciar la descarga nos lleva a la pantalla de “Download Insomnia” (Fig. 3.134), y una vez que pulsemos el botón “Download Insomnia for Windows” se nos descargará un fichero .exe con la aplicación cliente HTTP Rest portable (Fig. 3.135).

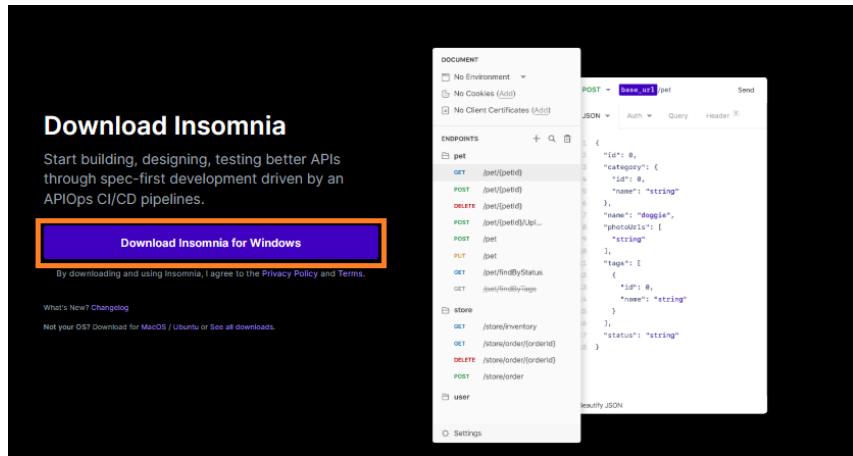


Fig. 3.134: Pantalla “Download Insomnia”.

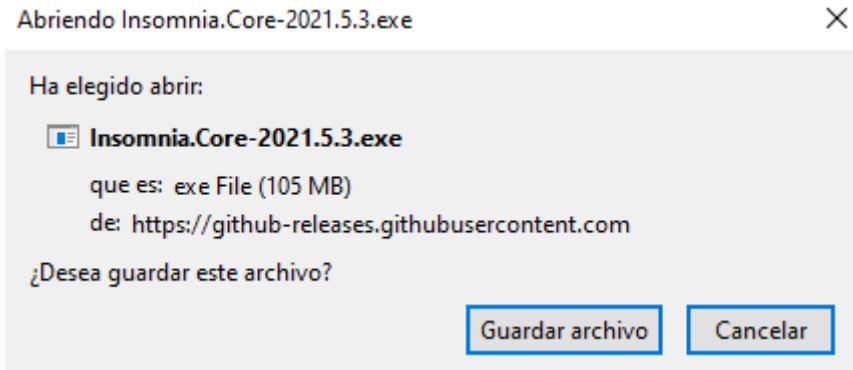


Fig. 3.135: Aceptación de descarga del ejecutable “Insomnia Core”.exe.

Abrir la aplicación portable haciendo doble clic sobre el ejecutable .exe que nos acabamos de descargar, y la aplicación estará ya lista para usarse.

3.6 Instalación de Angular.

Nótese que durante el proceso de explicación teórica de las herramientas Front-end del apartado 2, siempre se indica “Angular” y nunca se habla de “Angular JS”. Esto es importante aclararlo, puesto que “Angular JS” es la versión 1, y desde la versión 2 hasta la versión actual (versión 6) se llama solo “Angular”. Entonces cuando usemos Angular como framework Front-End estaremos usando un framework que será:

- De tipo JavaScript
- OpenSource
- Mantenido por Google

- Útil para crear páginas web SPA (simple page application): la página se cargará al inicio y el resto de elementos se recargan si son necesarios, pero no la página entera.

Aclarado todo lo anterior, en este apartado se va a mostrar la instalación de Angular, y la preparación del entorno de trabajo para el desarrollo front-end (que estará basado en Angular).

Para empezar a instalar el entorno de trabajo Front-End seguir los siguientes pasos:

Paso 1: Vamos al navegador, entramos en Google, y ponemos “node”. Entre las primeras búsquedas debe de aparecer el sitio oficial de “Node js”.

Paso 2: Ir a la sección de descargas del software “Node js” en la página oficial, y seleccionar la versión LTS más nueva que haya disponible.

Elegir siempre una versión LTS (Long-Time Support), porque es la versión que nos dará soporte a largo plazo (suele ser durante aprox. durante 4 años, desde su primer lanzamiento).

La siguiente figura (Fig. 3.136) muestra la opción de descarga elegida:



Fig. 3.136: página de descarga de la versión LTS para Node js.

Paso 3: Una vez tenemos descargado el instalador de la versión LTS de Node js, lo instalamos en nuestro equipo (ver figuras desde Fig. 3.137 hasta Fig. 3.144 para seguir el proceso de instalación).

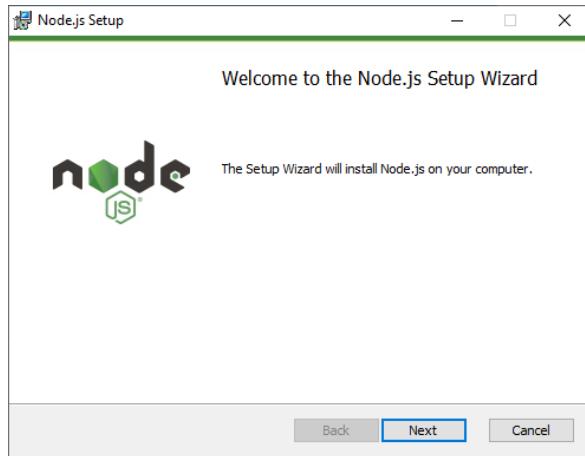


Fig. 3.137: Proceso de instalación Node js, parte 1.

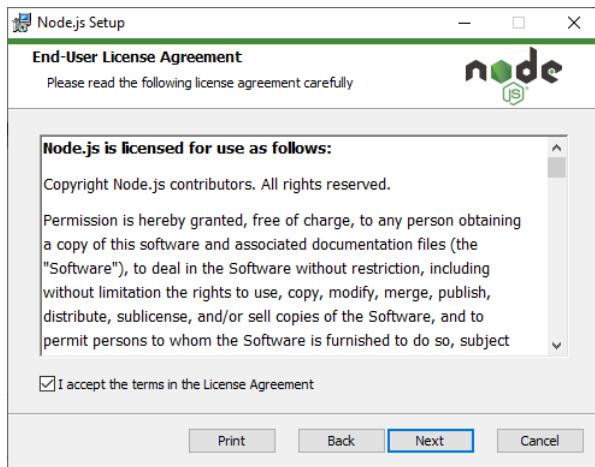


Fig. 3.138: Proceso de instalación Node js, parte 2.

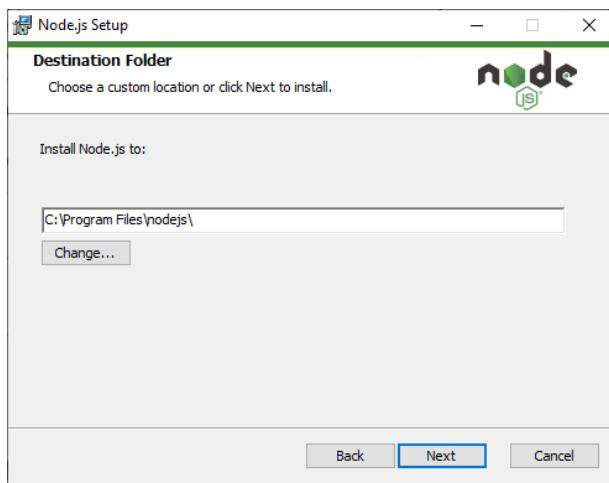


Fig. 3.139: Proceso de instalación Node js, parte 3.

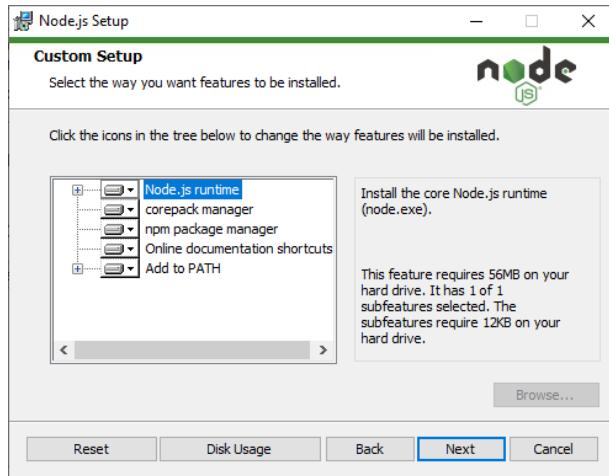


Fig. 3.140: Proceso de instalación Node js, parte 4.

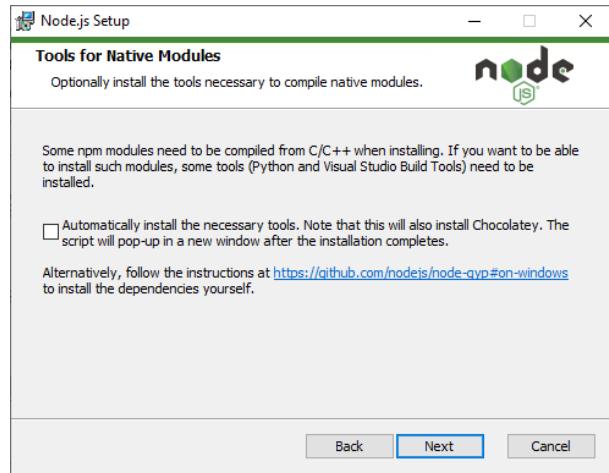


Fig. 3.141: Proceso de instalación Node js, parte 5.

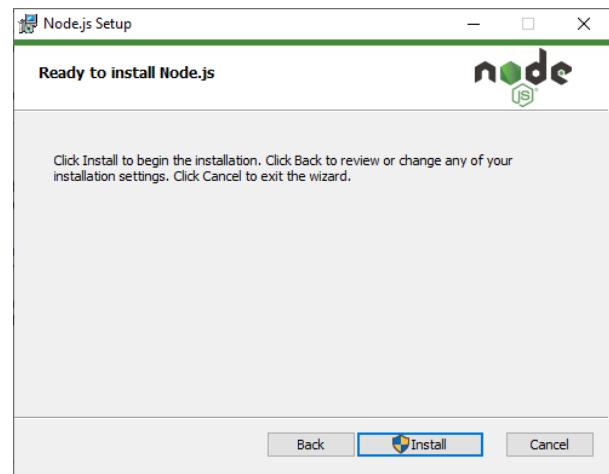


Fig. 3.142: Proceso de instalación Node js, parte 6.

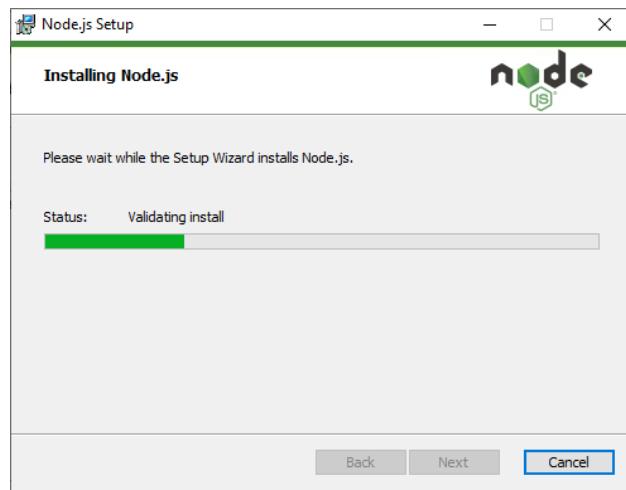


Fig. 3.143: Proceso de instalación Node js, parte 7.

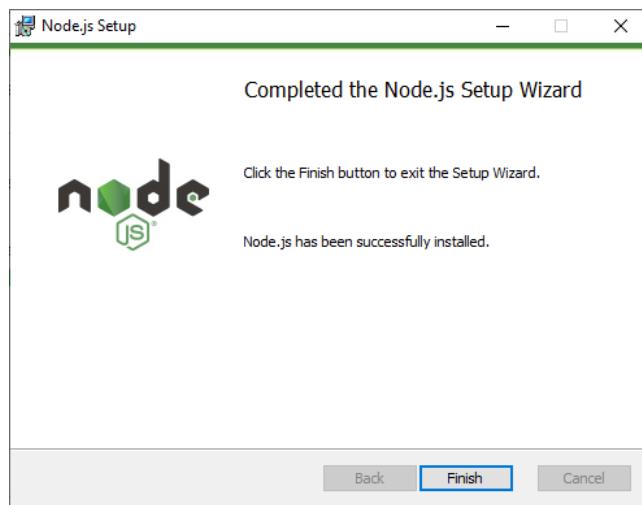


Fig. 3.144: Proceso de instalación Node js, parte 8.

Paso 4: Crear un directorio únicamente para Angular y otro directorio únicamente para TypeScript (ambos directorios al mismo nivel jerárquico). Ver Fig. 3.145.

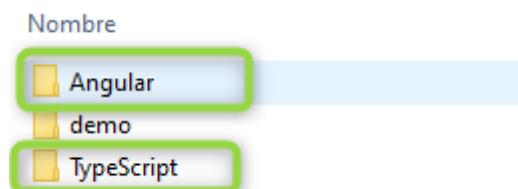


Fig. 3.145: Creación de 2 nuevos directorios, para Angular y TypeScript, al mismo nivel donde está la carpeta "demo" con el código Java.

Paso 5: Entrar al directorio de “Angular” y crear el proyecto “angular” como tal, desde una consola CMD. Para ello seguir se hacen 3 cosas (ver Fig. 3.146):

- 1) Entrar en la carpeta recién creada de ANGULAR.
- 2) Poner en el buscador de archivos superior del navegador de carpetas la palabra CMD.
- 3) De esta forma abrir la consola CMD de Windows con la dirección del directorio actual incorporada:

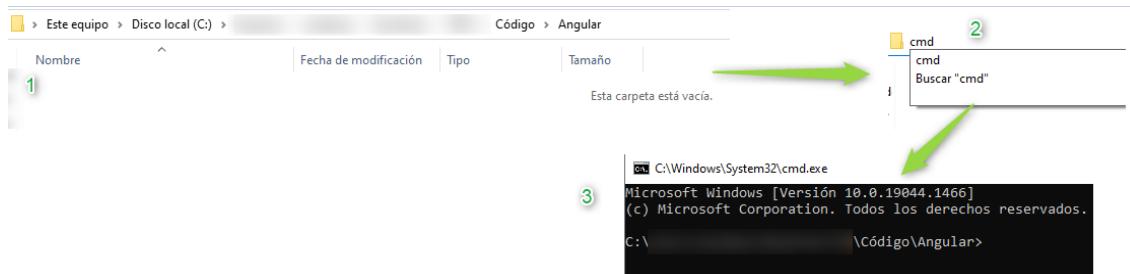


Fig. 3.146: Entrar en nueva carpeta Angular, abrir desde ahí CMD y visualización de CMD en directorio actual.

Paso 6: Instalar un cliente ANGULAR, con la ventana de comandos que se acaba de abrir, a través del comando “npm install –g @angular/cli”. Verá un ejemplo gráfico de lo que se está mencionando en la figura Fig. 3.147:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19044.1466]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Código\Angular>npm install -g @angular/cli
added 187 packages, and audited 188 packages in 22s
23 packages are looking for funding
  run 'npm fund' for details

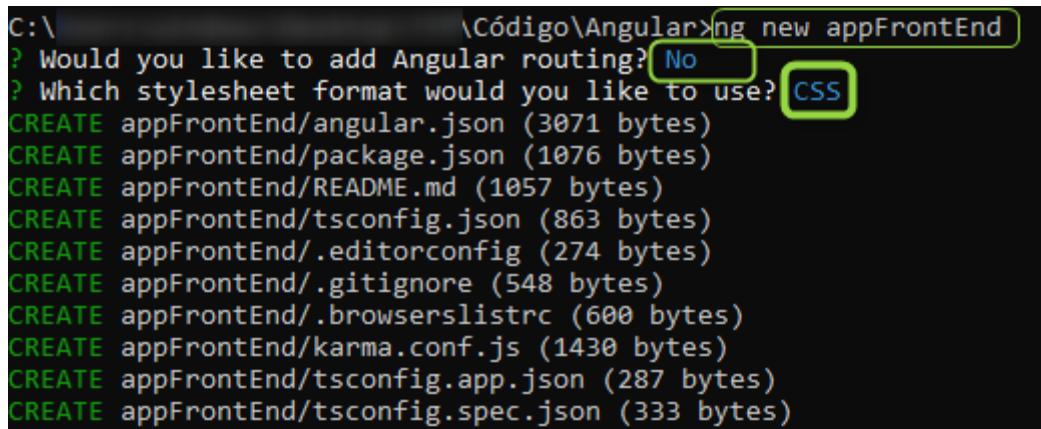
Found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 8.1.2 -> 8.4.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.4.0
npm notice Run npm install -g npm@8.4.0 to update!
npm notice
```

Fig. 3.147: Instalación del cliente Angular con el gestor de paquetes NPM y el cliente angular/cli.

Paso 7: Asegúrate de que se nos indica algo similar a la imagen anterior (Fig. 3.92) cuando instalamos el cliente de Angular: si se nos indica algo similar a lo que se ve en el recuadro naranja, la instalación del cliente Angular habrá sido un éxito.

Paso 8: Desde el mismo directorio que hemos instalado el cliente Angular, ahora ejecutamos el siguiente comando para crear nuestro primer proyecto Angular en el directorio que hemos creado para solo código Angular (“.../código/Angular/”).

Hasta que no creemos nuestro primer proyecto Angular, el directorio nuevo de Angular sigue vacío, tras crear el proyecto el directorio tendrá los archivos del proyecto recién creado (Fig. 3.148).



```
C:\ Código\Angular>ng new appFrontEnd
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE appFrontEnd/angular.json (3071 bytes)
CREATE appFrontEnd/package.json (1076 bytes)
CREATE appFrontEnd/README.md (1057 bytes)
CREATE appFrontEnd/tsconfig.json (863 bytes)
CREATE appFrontEnd/.editorconfig (274 bytes)
CREATE appFrontEnd/.gitignore (548 bytes)
CREATE appFrontEnd/.browserslistrc (600 bytes)
CREATE appFrontEnd/karma.conf.js (1430 bytes)
CREATE appFrontEnd/tsconfig.app.json (287 bytes)
CREATE appFrontEnd/tsconfig.spec.json (333 bytes)
```

Fig. 3.148: Creación de un nuevo proyecto completo en Angular, llamado “appFrontEnd”.

Paso 9: Cuando la instalación del proyecto haya terminado, y la terminal de comandos haya sido liberada, dejamos la terminal sin cerrar y nos dirigimos a la carpeta de Angular desde el navegador de archivos de Windows.

Debemos comprobar desde el navegador de carpetas de Windows, que en la carpeta “Angular” ahora sí están los archivos nuevos del proyecto Angular (y que antes estaba vacía).

Los nuevos archivos confirmarán 2 cosas:

- Que la descarga de los archivos del proyecto y la construcción del proyecto Angular fueron realizados.

La confirmación del nuevo proyecto (“appFrontEnd”) la podremos ver en la Fig. 3.149:

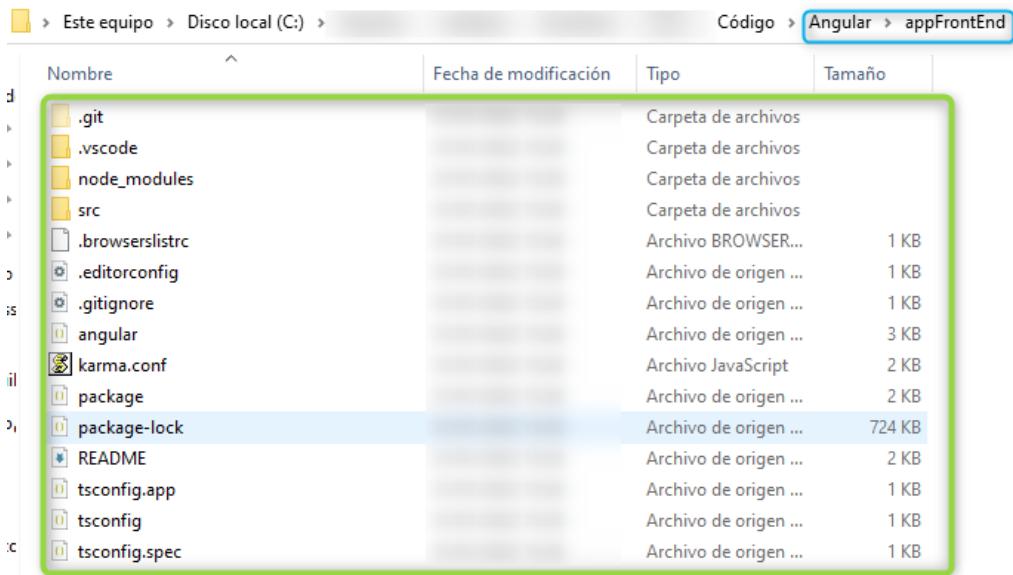


Fig. 3.149: Construcción del nuevo proyecto Angular “appFrontEnd”, visualizado desde el visor de archivos de Windows.

Paso 10: Ahora cerramos la terminal de comandos que estábamos usando.

Dentro de la carpeta “TypeScript”, que debe estar aún vacía, se abre una nueva terminal de comandos Windows (CMD) desde el buscador de archivos de windows en este directorio (ver paso 5 para abrir el CMD desde la carpeta “TypeScript”).

Cuando abra el CMD de esta forma, nos posicionará en la nueva ruta correspondiente de TypeScript (Fig. 3.150):

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19044.1466]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\ \Código\TypeScript>
```

Fig. 3.150: Apertura del nuevo CMD apuntando al directorio de TypeScript.

Paso 11: Ahora instalar el software correspondiente a TypeScript desde la nueva CMD (Fig. 3.151):

```
C:\ \Código\TypeScript>npm install -g typescript
added 1 package, and audited 2 packages in 2s
Found 0 vulnerabilities

C:\ \Código\TypeScript>
```

Fig. 3.151: Confirmación de la instalación del lenguaje TypeScript a nivel global, usando el gestor NPM.

Paso 12: Si tenemos una respuesta en el CMD similar a la imagen anterior (Fig. 3.96), cerramos este CMD.

Paso 13: Abrir el programa Visual Studio Code y cargar en el workspace los proyectos de Frontend y Typescript por separado en el área de trabajo del IDE.

El proyecto de Backend ya estaba incluido en el área de trabajo.

NOTA:

No cargar en el workspace el directorio padre una sola vez esperando que se carguen los diferentes proyectos en el área de trabajo. Esto generará que Visual Studio Code no detecte el servidor Spring Boot incrustado.

Mejor cargue cada proyecto por separado para que Visual Studio Code trate los proyectos de forma separada y no ocurran errores con el servidor Tomcat.

4. Desarrollo del proyecto

Esta sección el peso central del proyecto, y se divide en 3 subapartados.

Mientras que los ejemplos tutorizados más prácticos los encontramos dentro de los apartados 4.1 y 4.2 con el código back y front respectivamente, en el apartado 4.3 tendremos la programación de la web final con una redacción a más alto nivel y menor detalle.

Por tanto, se aconseja al lector de este tutorial que estudie primeramente los apartados 4.1 y 4.2 en detalle, y así poder seguir sin mayor dificultad el despliegue de la arquitectura propuesta en el apartado 4.3. También será en el apartado 4.3 donde se realizará la conexión del cliente Angular a la API REST desarrollada con Spring Boot.

4.1 Tutorial Back-end (Spring Boot).

En este apartado se explicará y se configurará todo lo esencial del proyecto Backend y la API REST.

El IDE que se utilizará será Visual Studio Code, y se dará por hecho que el lector ya cuenta con conocimientos básicos de Java.

Este apartado cubre como mínimas los siguientes puntos:

- 1) Montaje de la arquitectura básica de un proyecto backend, basado en Spring Boot y Maven, para desarrollar una API REST.
- 2) Conexión del proyecto Backend en Spring Boot a una base de Datos SQL.
- 3) Conexión del proyecto Backend en Spring Boot a una base de Datos No-SQL.
- 4) Uso de JPA para conectar con Base de Datos tipo SQL (relacionales).
- 5) Instalación y uso de base de datos No-SQL (colecciones de datos).
- 6) Realización de llamadas REST con clientes HTTP, mediante herramientas DevOp (como Postman).
- 7) Protección de API REST a través de Bearer Token.

a) ¿Qué hemos explicado ya de Java, Maven y Spring Boot?

En el apartado 3 se explicó la instalación de 6 herramientas, y para el Back-end nos interesarán las siguientes:

- Open JDK (apartado 3.1, pag. 22-30).
- Descarga del proyecto Maven prefabricado, con framework Spring Boot, Maven y Tomcat incrustado (apartado 3.3, pag. 48-61).
- Cliente HTTP (apartado 3.5, pag. 73-75).
- IDE Visual Studio Code.

Nota: El orden de aparición del contenido en este tutorial sigue el orden que debería seguir para instalarse cada herramienta (recomiendo seguir el mismo orden de instalación).

La primera herramienta que debería estar instalada es el intérprete de Java en modalidad “opensource”, y así lograremos que las herramientas que necesiten el JDK y JRE se integren correctamente más adelante.

Lo segundo que debemos tener instalado es el proyecto Maven prefabricado, con Spring Boot (lleva un servidor Tomcat incrustado). Luego tener cargado el proyecto Spring Boot descargado en el workspace de Visual Studio Code.

En Visual Studio Code se instalaron 2 cosas:

- El plugin para VSC que interpreta lenguaje Java (apartado 3.3 paso “8 c”).
- Las 4 las dependencias nuevas en el super “pom.xml” del proyecto Maven (dependencias en MavenRepository en apartado 3.3 paso 15, pág. 56-58; código incluido en el archivo “pom.xml” de las 4 dependencias en apartado 3.3 paso 16, pág. 59). Estas 4 dependencias fueron:
 - Spring Boot Starter Web
 - Spring BootDevTools
 - H2 DatabaseEngine
 - Spring Boot Starter Data JPA

Si se desea realizar test unitarios, añadir en el “pom.xml” la siguiente dependencia:

- Spring Boot Starter Test

Finalmente se instaló el software de “Insomnia Core” (es la herramienta Free sustituto de Postman), para conseguir las funciones de un cliente Web mientras no se tenga desarrollado el Frontend.

Por el momento, “Insomnia Core” hará las veces de cliente a la API REST.

b) Configuración del servidor local Backend.

Paso 1: Comprobaciones iniciales:

- El proyecto prefabricado y descargado del backend, que viene por defecto, se vería similar a lo que vemos en la siguiente Fig. 4.1.1.
- Aparece en Visual Studio Code un botón de “Spring Boot Dashboard”, bien como menú del lateral izquierdo o bien como subapartado de la ventana del explorador de archivos (debajo del workspace): véase en la Fig. 4.1.2.
- Comprobamos que se detecta el proyecto Spring Boot desde VSC, detectando que esté el botón “Spring Boot Dashboard” o el botón lateral de encendido sexagonal (véase Fig. 4.1.3). Desde este nueva sección debe poder ejecutarse y debugearse el proyecto.
- Al ejecutar el proyecto desde el Dashboard, comprobar que no ocurren errores por consola y que se cargan las nuevas dependencias (ver Fig. 4.1.4).

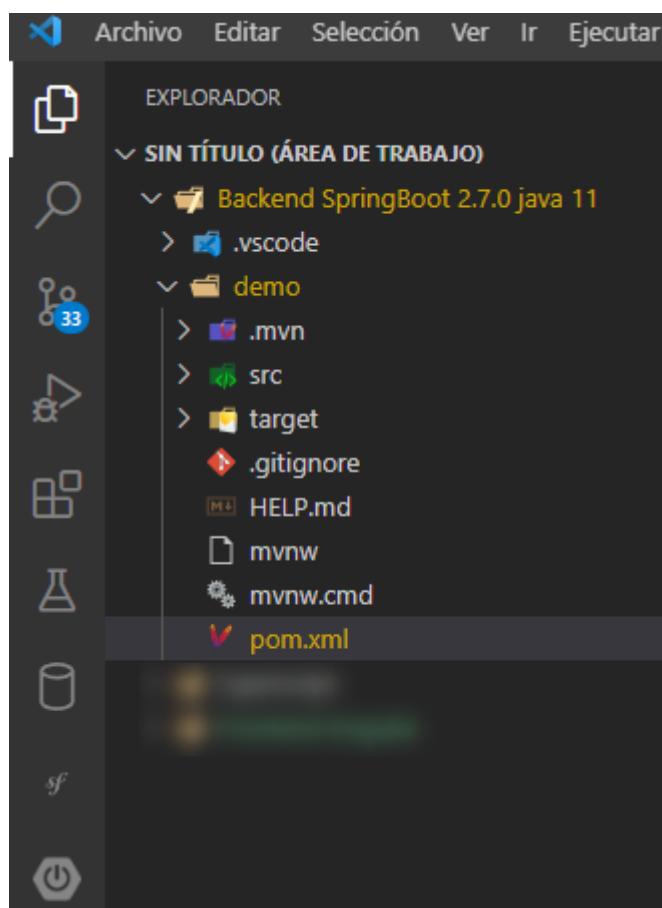


Fig. 4.1.1: Nuevo proyecto backend de Spring Boot cargado en workspace de Visual Studio Code.

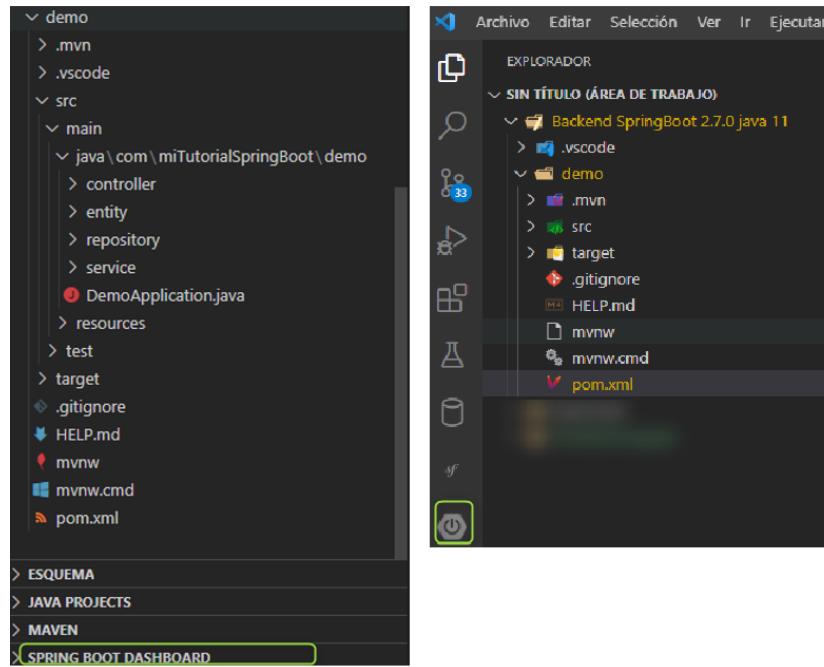


Fig. 4.1.2: Localización del botón “Spring Boot Dashboard”, ver recuadro verde.

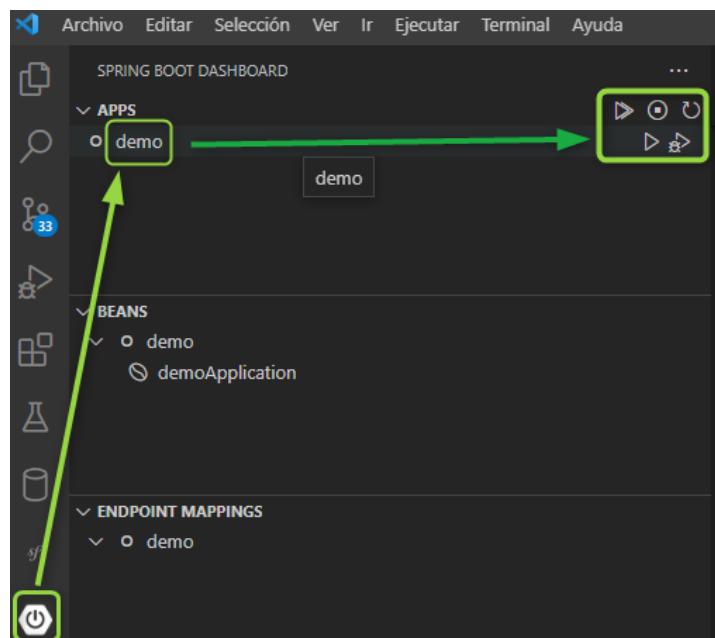


Fig. 4.1.3: Opciones para hacer Run o Debug del proyecto, desde el menú “Spring Boot”.

```

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.7.0</version>
</dependency>

<!-- librería para corregir el error de "Failed to Refresh live data from database" -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL SQL CONSOLE

formed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2022-11-09 01:15:09.392 INFO 12036 --- [restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-11-09 01:15:09.398 INFO 12036 --- [restartedMain] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actuator'
2022-11-09 01:15:09.445 INFO 12036 --- [restartedMain] o.e.j.s.h.ContextHandler.application : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-11-09 01:15:09.445 INFO 12036 --- [restartedMain] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-11-09 01:15:09.447 INFO 12036 --- [restartedMain] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-11-09 01:15:09.473 INFO 12036 --- [restartedMain] o.e.jetty.server.AbstractConnector : Started ServerConnector@499a810[HTTP/1.1, (http/1.1)]@{0.0.0.0:8080}
2022-11-09 01:15:09.474 INFO 12036 --- [restartedMain] o.s.b.web.embedded.jetty.JettyWebServer : Jetty started on port(s) 8080 (http/1.1) with context path '/'
2022-11-09 01:15:09.487 INFO 12036 --- [restartedMain] com.example.demo.DemoApplication : Started DemoApplication in 3.328 seconds (JVM running for 4.021)

Fig. 4.1.4: Visualizamos las dependencias que usa el proyecto.

Paso 2: Configuración del proyecto:

- Se propone usar como servidor incrustado un servidor Tomcat (lo lleva Spring Boot por defecto), o añadir manualmente un servidor Jetty como alternativa.
- Para modificar los servidores web, ir al archivo “pom.xml” y configurarlo como proceda según el tipo de servidor elegido:

- **Si usamos un servidor Tomcat (ya incluido):**

- 1º La dependencia que lo carga es la de “spring-boot-starter-web” y suele venir incluida en el pom.xml (ver Fig. 4.1.5).

- 2º Si ejecutamos esta configuración se nos cargará la dependencia de “Spring-boot-start-web” con un Tomcat ligero. Puede verse que está cargando la dependencia de Tomcat en la Fig. 4.1.6.

```

<?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 https://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>2.7.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo</name>
  <description>Demo project de Spring Boot</description>
  <properties>
    <java.version>11</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
    <!-- Usada para poder usar RESTFull y RESTController -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <version>2.7.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools -->
    <!-- Usada para refrescar el servicio al instante cuando guarde cambios en el código fuente -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <version>2.7.0</version>
    </dependency>
  </dependencies>

```

Líne 31, col 9 Tamaño de tabulación: 4 UTF-8 LF ☰ XML ⚡ Q

Fig. 4.1.5: Instalación, a través del “pom.xml”, de la dependencia “spring-boot-starter-web”, y así usar RESTFull con servidor Tomcat.

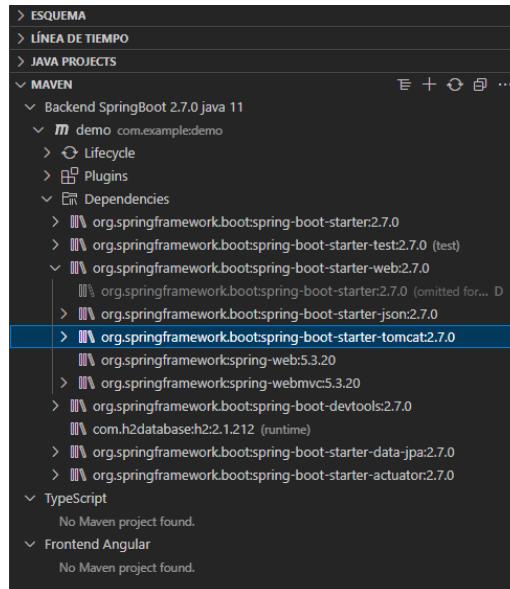
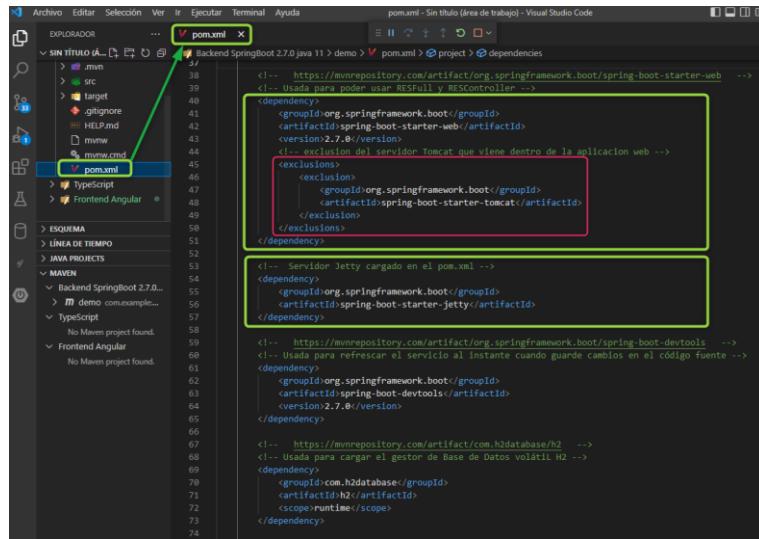


Fig. 4.1.6: Dependencias cargadas en el proyecto actualmente, contando entre ella con una con un servidor Tomcat. Puede verlo en el explorador de Visual Studio Code, debajo del workspace. Viene señalada la funcionalidad que indica que el servidor Tomcat ha sido cargado correctamente.

- Si usamos un servidor Jetty (requiere configuración manual):

1º) Necesitamos todavía la dependencia “spring-boot-starter-web” (para cargar la funcionalidad RESTFull), pero excluyendo el servidor Tomcat que lleva en su interior. Ver Fig. 4.1.7.

2º) Ahora añadir el servidor Jetty, instalando la dependencia “spring-boot-starter-jetty” en el fichero “pom.xml”. Esto instalará el nuevo servidor Jetty en el proyecto. Ver Fig. 4.1.8.



```

<!-- exclusion del servidor Tomcat que viene dentro de la aplicacion web -->
<exclusions>
    <exclusion>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
</exclusions>

<!-- Servidor Jetty cargado en el pom.xml -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>

```

Fig. 4.1.7: (1º) Excluyo el servidor Tomcat, que venía cargado automáticamente con la dependencia “Spring-boot-starter-web” (para usar RESTFull). (2º) Instalo la dependencia “spring-boot-starter-jetty” para usar un servidor Jetty en llamadas RESTFull.

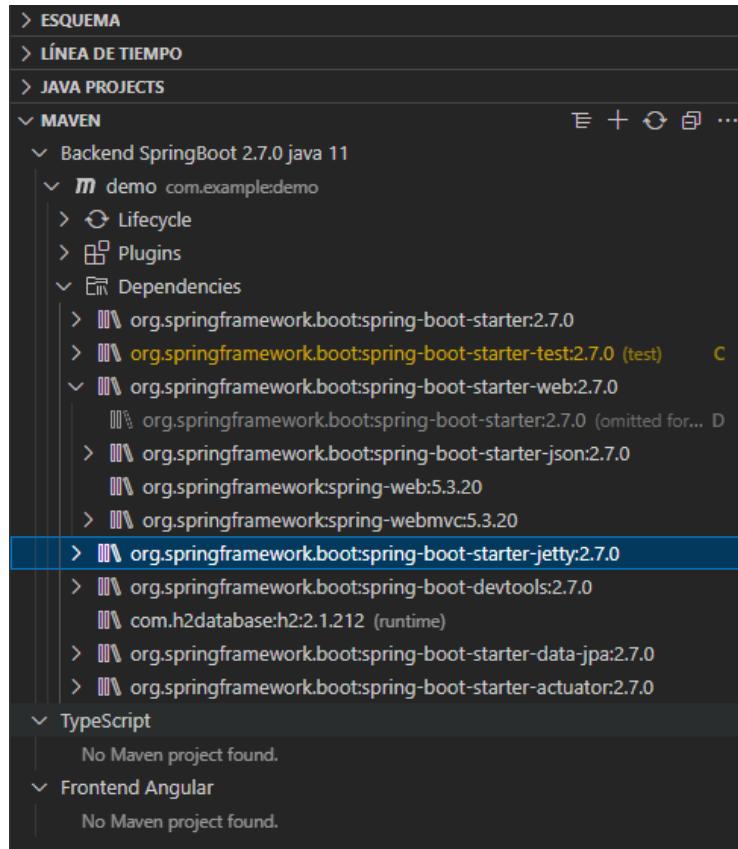


Fig. 4.1.8: Dependencias cargadas en el proyecto, en la parte inferior del Workspace, cuando se ejecuta con la configuración adicional del servidor Jetty. La funcionalidad que indica que el servidor Jetty ha sido cargado en nuestro proyecto es la señalada en la foto. También destacar que no se debe visualizar ya la funcionalidad del servidor Tomcat (funcionalidad eliminada).

Paso 3: En nuestro caso usaremos el servidor Tomcat que venía por defecto. En este punto habremos terminado de instalar todas las dependencias en el fichero “pom.xml” del apartado 3.3 (paso 8 en adelante). Estamos en disposición de empezar a crear la arquitectura básica del proyecto Back-end.

Por seguridad, comprobar que ejecuta la dependencia “spring-boot-starter-web”, como la siguiente Fig. 4.1.9.

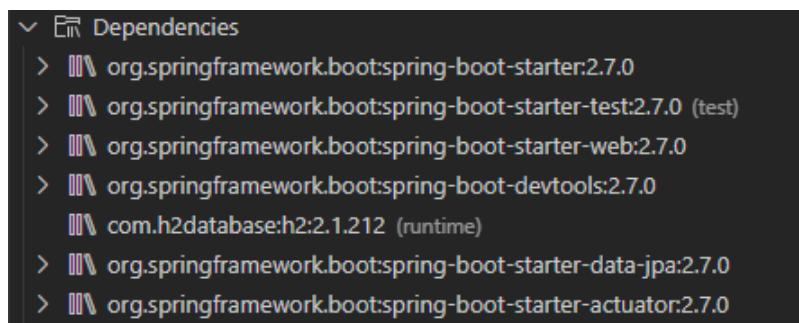


Fig. 4.1.9: Elección definitiva para las dependencias de nuestro proyecto backend.

c) Consultas CRUD con “Insomnia Core”.

En este apartado se presentan las llamadas HTTP típicas que se pueden realizar con una API REST al back desde un cliente web. Ahora usaremos como cliente Insomnia Core y más adelante usaremos Angular.

Las 4 llamadas básicas a bases de datos son: create, read, update y delete.

Antes de empezar, 2 cosas a tener en cuenta:

- Para probar las llamadas HTTP usaremos “Insomnia Core”.
- En todas las llamadas REST presentadas se define lo siguiente:
 - URL de la llamada.
 - Método HTTP usado.
 - PathVariables en la URL usadas, si procede.
 - RequestBody en formato JSON (añadiendo cabecera de “Content-Type” = “application/json”), si procede.
 - RequestParam (como parte del queryString de la URL), si procede.

La creación de todas las llamadas en “Insomnia Core” se hará manualmente (no se importa ni se exporta un XML de todas las llamadas REST).

Para crear las 4 consultas HTTP deseadas, sigue los siguientes pasos:

Paso 1: Abrir Insomnia Core (Fig. 4.1.10):

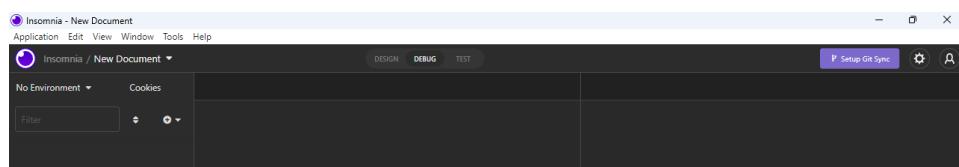


Fig. 4.1.10: Estado inicial del programa Insomnia Core, vacío, sin consultas inicialmente.

Paso 2: Clicar el botón de “+” y seleccionar “HTTP Request” (Fig. 4.1.11):

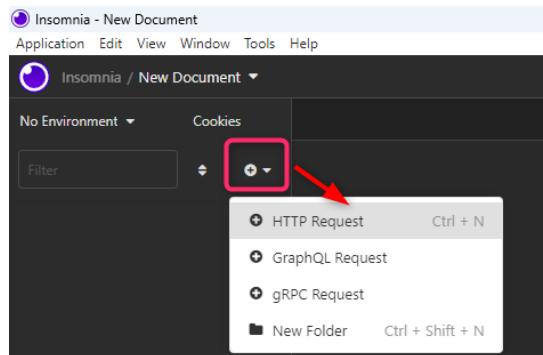


Fig. 4.1.11: Creación de consulta HTTP en Insomnia Core.

Paso 3: Se creó una nueva llamada HTTP, en el margen izquierdo, y tendrá los siguientes parámetros por defecto:

- Método HTTP: GET.
- Nombre por defecto: “New Request”.
- URL: inicialmente vacía.
- Todas las secciones de la llamada HTTP (Body, Auth, Query, Header y Docs): inicialmente vacías.
- Área de Response y el historial de responses: vacío.

Paso 4: Repetir proceso del paso 3, hasta crear las otras 3 llamadas CRUD.

Paso 5: Modificar el nombre de la llamada HTTP que fue creada en primer lugar:

- “New Request” pasará a llamarse “CrearUsuarios”.
 - Clicar sobre la llamada creada, y pulsar botón derecho.
 - En el submenú seleccionar “Settings” (ver Fig. 4.1.12).

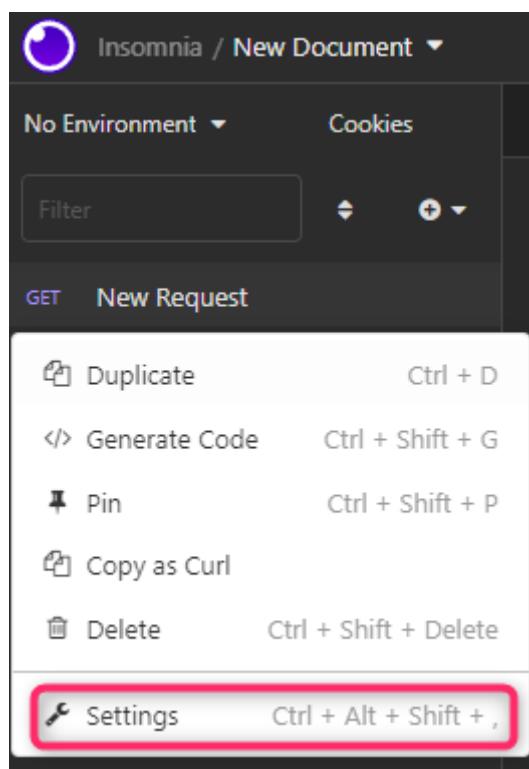


Fig. 4.1.12: Área de Settings, para cambiar nombre de la llamada.

- En la ventana emergente, modificar texto “New Request” por “CrearUsuarios” (ver Fig. 4.1.13 y Fig. 1.2.14).

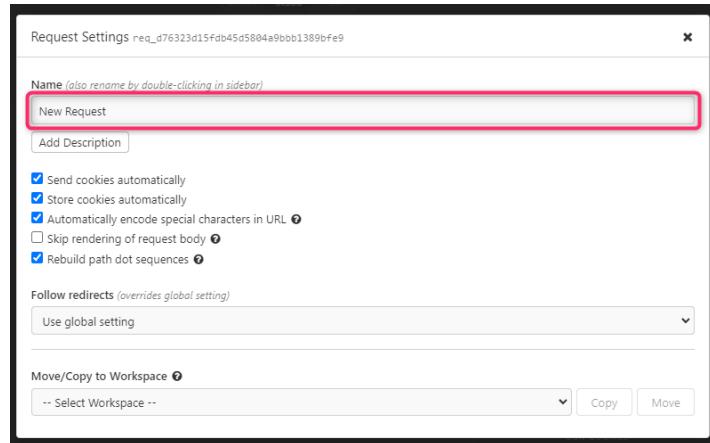


Fig. 4.1.13: Ventana emergente con nombre inicial de la llamada.

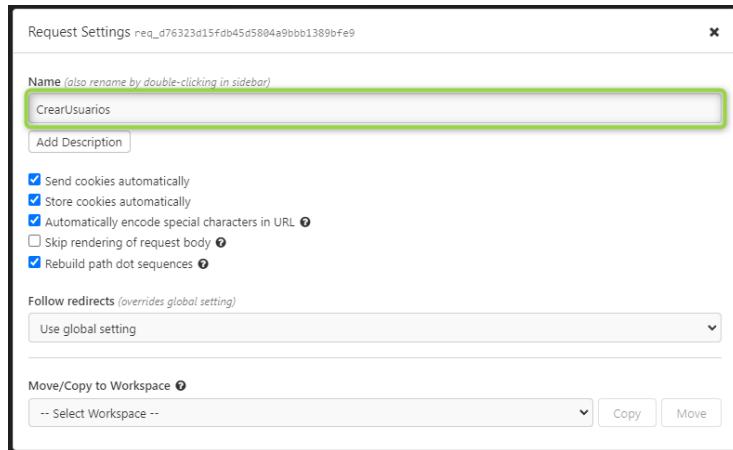


Fig. 4.1.14: Modificación del nombre de llamada.

- Cuando modificó el nombre de la llamada HTTP, cierra la ventana emergente y el nombre de llamada se habrá cambiado correctamente (ver Fig. 4.1.15).

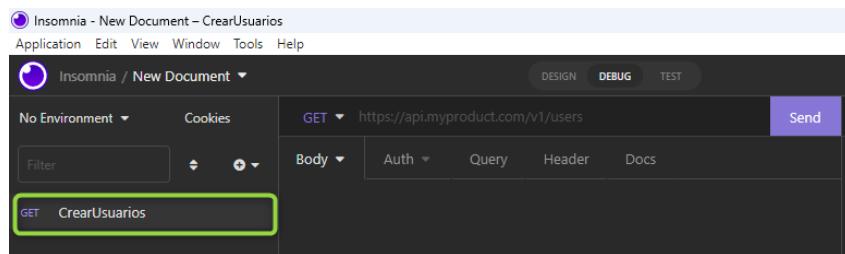


Fig. 4.1.15: Nombre de la llamada modificado satisfactoriamente.

Paso 6: Repetir proceso de modificación de nombre para cada una de las 3 llamada HTTP restantes. Ver el resto de nombres propuestos en la Fig. 4.1.16.

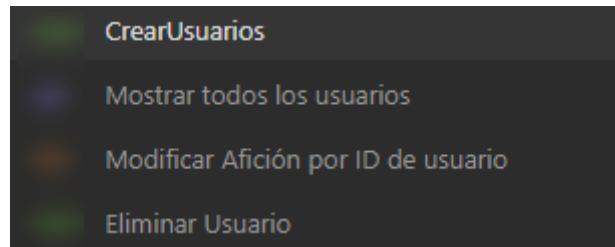


Fig. 4.1.16: Las 4 llamadas HTTP con sus nombres definitivos ya colocados.

Paso 7: Colocar el método HTTP correcto para cada caso concreto según la llamada HTTP que sea. Puede hacer uso de la siguiente lista:

- 1) Llamada “CrearUsuarios”:
 - POST (ver Fig. 4.1.17).
- 2) Llamada “Mostrar todos los usuarios”:
 - GET (ver Fig. 4.1.18).
- 3) Llamada “Modificar Afición por ID de usuario”:
 - PUT (ver Fig. 4.1.18).
- 4) Llamada “Eliminar Usuario”:
 - POST (ver Fig. 4.1.18).

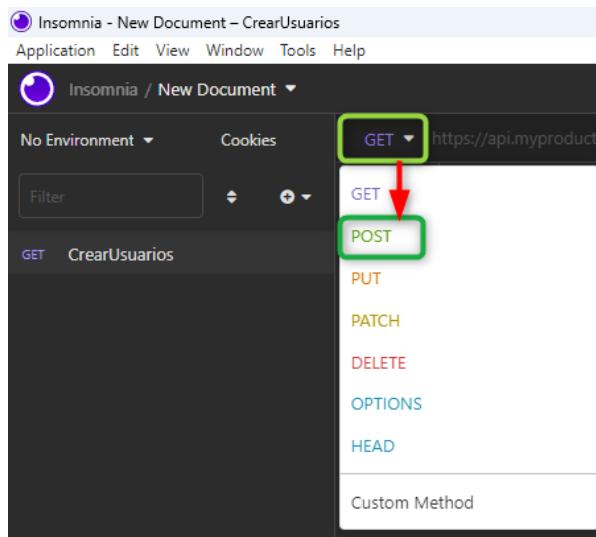


Fig. 4.1.17: Elección de método POST creando la 1^a llamada “CrearUsuarios”.

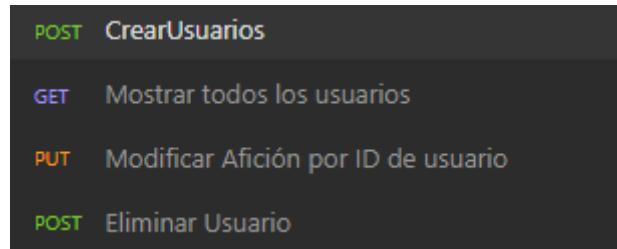


Fig. 4.1.18: Elección del método HTTP adecuado, en todas las llamadas creadas.

Paso 8: Preparar URL, JSON y Header adecuadamente, según proceda para cada llamada.

- Caso (1): Llamada “CrearUsuarios” (ver Fig. 4.1.19 y Fig. 4.1.20). La llamada contiene las siguientes cosas:
 - 1) URL fija “localhost:8080”.
 - 2) Subdirectorio de la URL llamado “crearusuario”.
 - 3) Con PathVariable en la URL, para añadir datos después del subdirectorio “crearusuario”.
 - 4) Con RequestBody: JSON enviado en el RequestBody (de la llamada HTTP con método POST) que contiene los atributos del nuevo usuario que se va a crear.
 - 5) Con Header, configurados correctamente (en Fig. 4.1.20).
 - 6) Sin queryString en la URL.

The screenshot shows the Insomnia REST client interface. A POST request is being prepared to the endpoint `localhost:8080/crearusuario/Futbol`. The JSON body is highlighted with a red box and contains the following user creation data:

```

1  {
2   "nombre": "Juan",
3   "apellidos": "Fernández",
4   "edad": 18,
5   "activado": true,
6   "telefono": "+34123456789",
7   "genero": 1,
8   "modalidad": 1
9 }

```

Fig. 4.1.19: URL de llamada POST que crea usuarios, juntos al RequestBody adecuado.

The screenshot shows the Insomnia REST client interface with the 'Header' tab active. A header named `Content-Type` is set to `application/json`, which is highlighted with a green box.

Fig. 4.1.20: Header necesario, activo cuando uso RequestBody de una llamada POST.

- Caso (2): Llamada “Mostrar todos los usuarios” (ver Fig. 4.1.21). La llamada contiene las siguientes cosas:
 - 1) URL fija “localhost:8080”.
 - 2) Sin subdirectorios.
 - 3) Sin PathVariable en la URL.
 - 4) Sin RequestBody (ni JSON). Imposible usar en llamadas tipo GET.
 - 5) Sin Header.
 - 6) Con QueryString en la URL.
 - Como esto: “?ordenadosPor=id”
 - Usado para modificar el orden de los elementos que se van a visualizar en el resultado de la consulta.

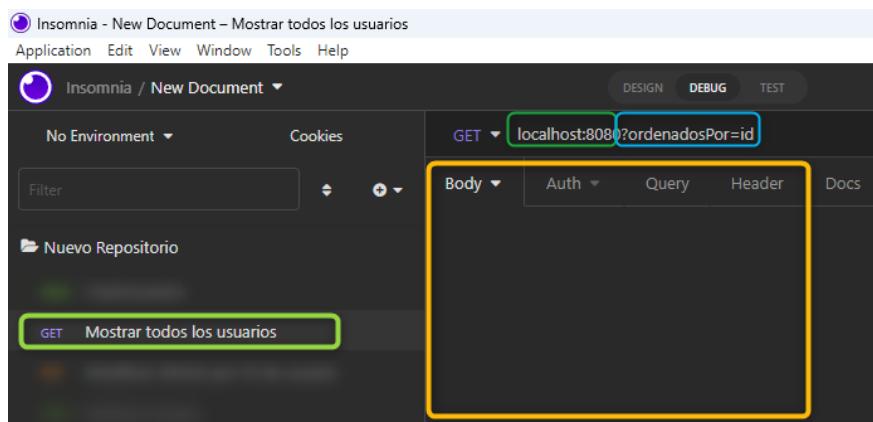


Fig. 4.1.21: URL de llamada GET que muestra todos los usuarios, ordenando los elementos por el atributo indicado en el queryString.

- Caso (3): Llamada “Modificar Afición por ID de usuario” (ver Fig. 4.1.22). La llamada contiene las siguientes cosas:
 - 1) URL fija “localhost:8080”.
 - 2) Subdirectorio de la URL llamado “modificarDetalleUsuario”.
 - 3) Con PathVariable en la URL, para añadir datos después del subdirectorio “modificarDetalleUsuario”.
 - 4) Sin RequestBody (ni JSON). No usado en la llamada PUT, aunque sería posible usarlo.
 - 5) Sin Header.
 - 6) Sin QueryString en la URL.

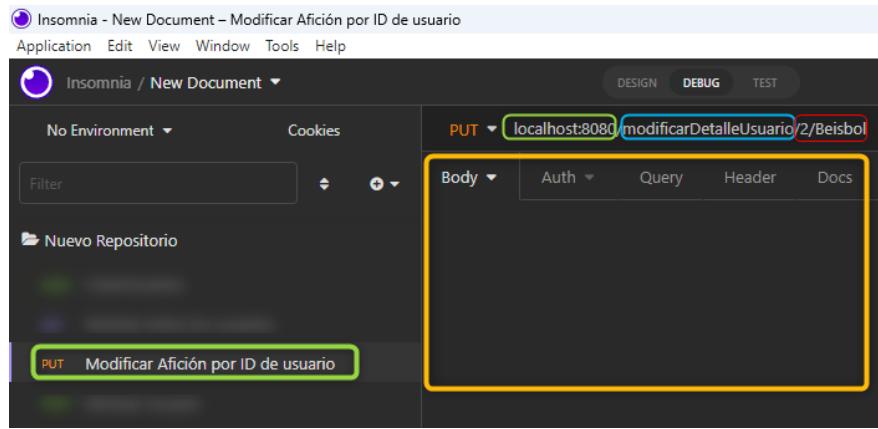


Fig. 4.1.22: URL de llamada PUT que modifica un atributo del usuario, sin RequestBody necesario.

- Caso (4): Llamada “Eliminar Usuario” (ver Fig. 4.1.23 y Fig. 4.1.24). La llamada contiene las siguientes cosas:
 - 1) URL fija “localhost:8080”.
 - 2) Subdirectorio de la URL llamado “eliminarUsuario”.
 - 3) Con PathVariable en la URL, para añadir datos después del subdirectorion “eliminarUsuario”.
 - 4) Sin RequestBody (ni JSON). No usado en la llamada POST, aunque sería posible usarlo.
 - 5) Con Header, configurados correctamente (en Fig. 4.1.24).
 - 6) Sin QueryString en la URL.

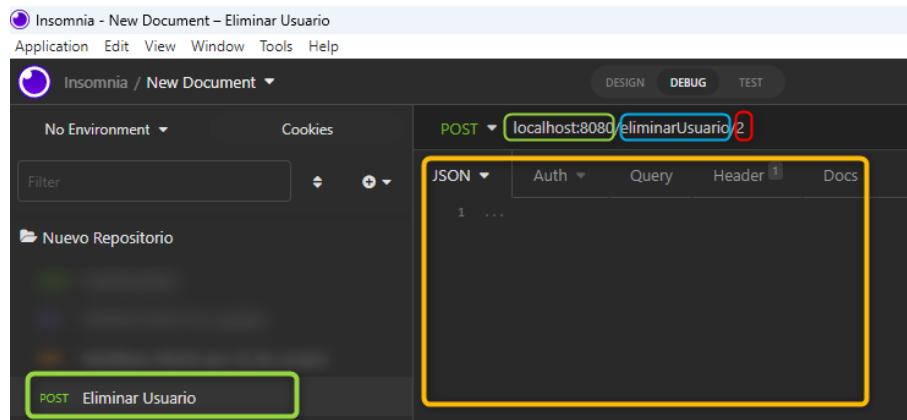


Fig. 4.1.23: URL de llamada POST que elimina usuarios, con RequestBody vacío.

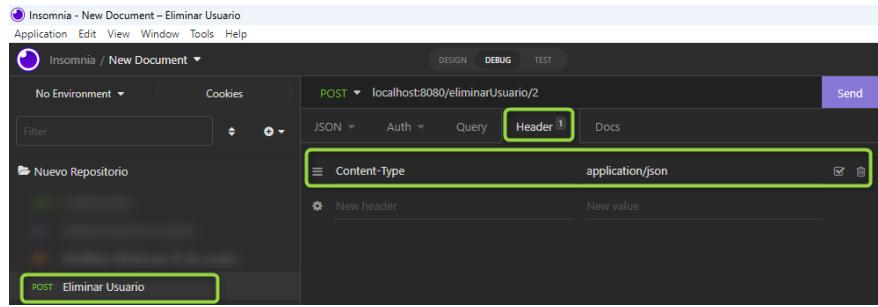


Fig. 4.1.24: Header necesario, activo cuando uso RequestBody de una llamada POST.

d) Creación de la arquitectura básica del proyecto Backend.

En este apartado se va a presentar la arquitectura básica del proyecto Back-end, creada desde Visual Studio Code.

Colocándonos en Visual Studio Code y viendo el workspace como se visualizaba en la Fig. 4.2.1, realizar los siguientes 13 pasos:

Paso 1: Entrar en el directorio “/demo/src/main/java/com/example/demo/”.

Paso 2: Crear dentro del directorio indicado las siguientes 7 carpetas (ver resultado final en Fig. 4.1.25):

1. controller
2. dto
3. entity
4. mapper
5. model
6. repository
7. service

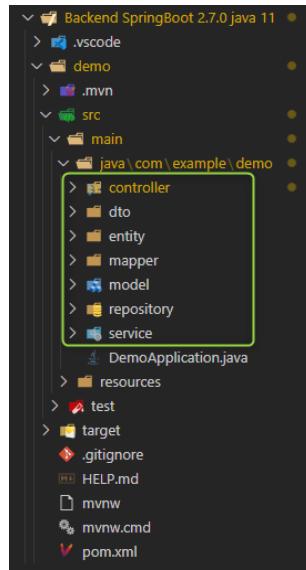


Fig. 4.1.25: Nuevos subdirectorios para la arquitectura del proyecto Backend.

Paso 3: Crear, en el interior de cada una de las carpetas recién creadas, los archivos indicados a continuación según proceda (ver resultado final en Fig. 4.1.26):

- 1. controller:
 - DemoController.java
- 2. dto:
 - UsuarioDTO.java
- 3. entity:
 - UsuarioEntity.java
- 4. mapper:
 - UsuarioEntityMapper.java
 - UsuarioMapper.java
- 5. model:
 - Usuario.java
- 6. repository:
 - UsuarioRepository.java
- 7. service:
 - DemoService.java

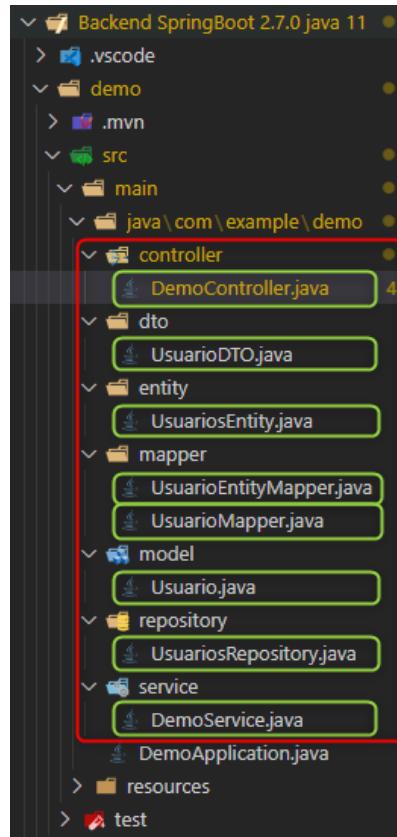
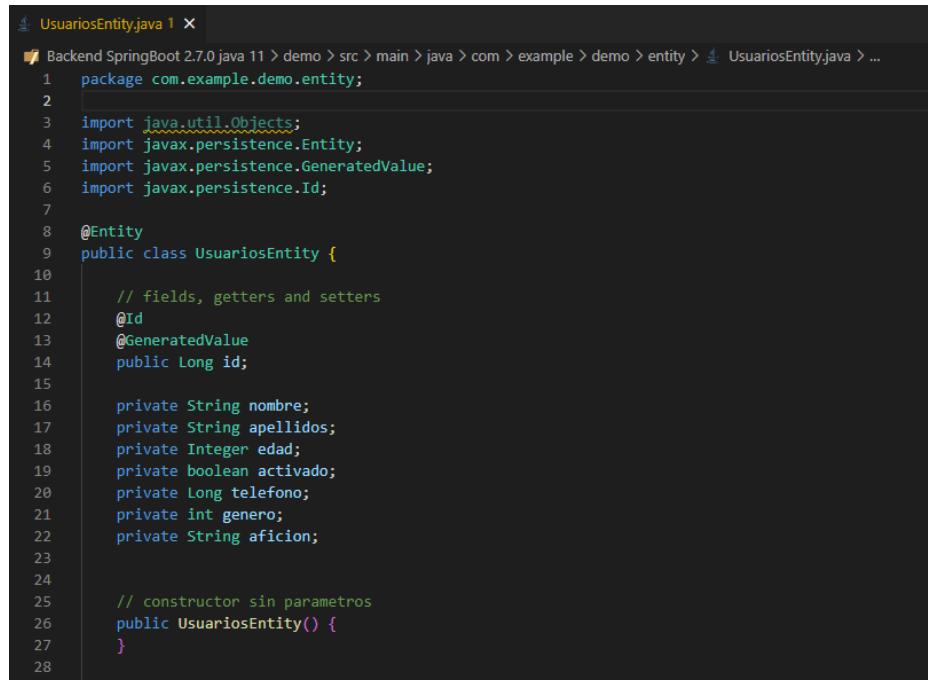


Fig. 4.1.26: Creación de las nuevas clases Java, dentro de cada uno de los nuevos subdirectorios.

Paso 4: Codificamos la clase “UsuarioEntity.java”, que será la que define la tabla “Usuarios” en la base de datos. Incluir uso de javadoc antes de cada método.

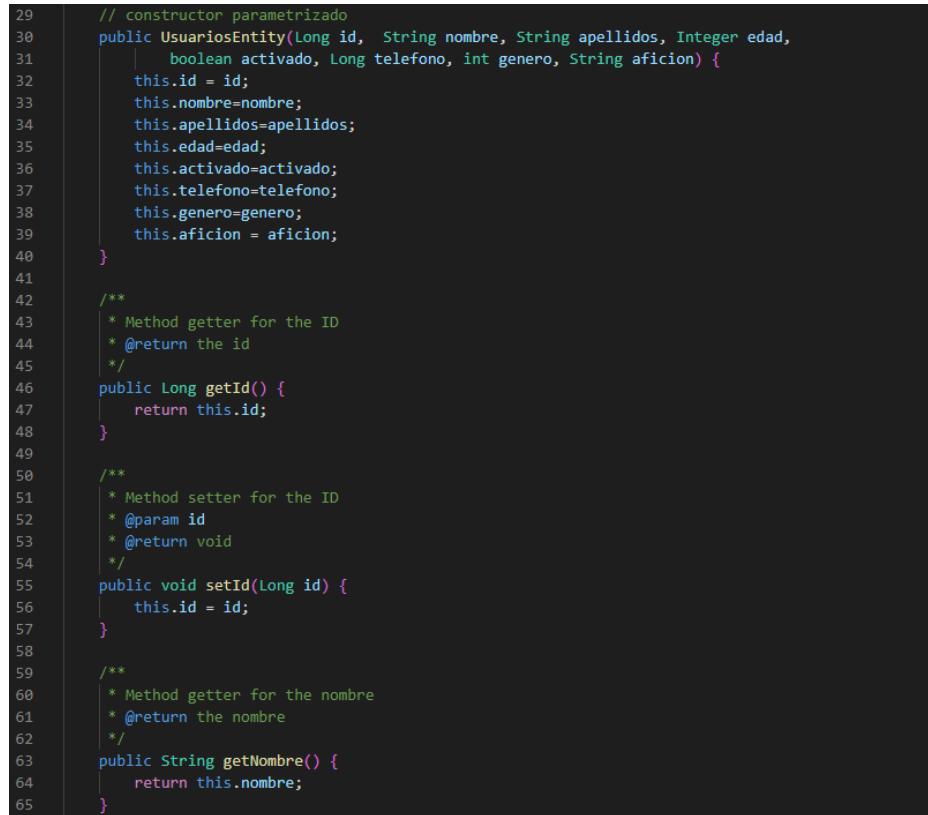
Puede verse la codificación de la misma en las siguientes figuras (desde Fig. 4.1.27 hasta Fig. 4.1.31).



The screenshot shows a Java code editor with the file 'UsuariosEntity.java' open. The code defines a class 'UsuariosEntity' with fields for id, nombre, apellidos, edad, activado, telefono, genero, and aficion. It includes a constructor and getters/setters for each field.

```
1 package com.example.demo.entity;
2
3 import java.util.Objects;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.Id;
7
8 @Entity
9 public class UsuariosEntity {
10
11     // fields, getters and setters
12     @Id
13     @GeneratedValue
14     public Long id;
15
16     private String nombre;
17     private String apellidos;
18     private Integer edad;
19     private boolean activado;
20     private Long telefono;
21     private int genero;
22     private String aficion;
23
24
25     // constructor sin parametros
26     public UsuariosEntity() {
27
28 }
```

Fig. 4.1.27: Fragmento 2, clase “UsuariosEntity.java”.



The screenshot shows the continuation of the 'UsuariosEntity.java' file. It includes a parameterized constructor, getters and setters for the fields, and annotations for the methods.

```
29     // constructor parametrizado
30     public UsuariosEntity(Long id, String nombre, String apellidos, Integer edad,
31             boolean activado, Long telefono, int genero, String aficion) {
32         this.id = id;
33         this.nombre=nombre;
34         this.apellidos=apellidos;
35         this.edad=edad;
36         this.activado=activado;
37         this.telefono=telefono;
38         this.genero=genero;
39         this.aficion = aficion;
40     }
41
42     /**
43      * Method getter for the ID
44      * @return the id
45      */
46     public Long getId() {
47         return this.id;
48     }
49
50     /**
51      * Method setter for the ID
52      * @param id
53      * @return void
54      */
55     public void setId(Long id) {
56         this.id = id;
57     }
58
59     /**
60      * Method getter for the nombre
61      * @return the nombre
62      */
63     public String getNombre() {
64         return this.nombre;
65     }
```

Fig. 4.1.28: Fragmento 2, clase “UsuariosEntity.java”.

```

67  /**
68   * Method setter for the nombre
69   * @param nombre
70   * @return void
71   */
72   public void setNombre(String nombre) {
73     this.nombre = nombre;
74   }
75
76 /**
77  * Method getter for the apellidos
78  * @return the apellidos
79  */
80   public String getApellidos() {
81     return this.apellidos;
82   }
83
84 /**
85  * Method setter for the apellidos
86  * @param apellidos
87  * @return void
88  */
89   public void setApellidos(String apellidos) {
90     this.apellidos = apellidos;
91   }
92
93 /**
94  * Method getter for the edad
95  * @return the edad
96  */
97   public Integer getEdad() {
98     return this.edad;
99   }
100

```

Fig. 4.1.29: Fragmento 3, clase “UsuariosEntity.java”.

```

101 /**
102  * Method setter for the edad
103  * @param edad
104  * @return void
105  */
106   public void setEdad(Integer edad) {
107     this.edad = edad;
108   }
109
110 /**
111  * Method getter for the activado
112  * @return the activado
113  */
114   public boolean getActivado() {
115     return this.activado;
116   }
117
118 /**
119  * Method setter for the activado
120  * @param activado
121  * @return void
122  */
123   public void setActivado(boolean activado) {
124     this.activado = activado;
125   }
126
127 /**
128  * Method getter for the telefono
129  * @return the telefono
130  */
131   public Long getTelefono() {
132     return this.telefono;
133   }

```

Fig. 4.1.30: Fragmento 4, clase “UsuariosEntity.java”.

```

134
135     /**
136      * Method setter for the telefono
137      * @param telefono
138      * @return void
139      */
140     public void setTelefono(Long telefono) {
141         this.telefono = telefono;
142     }
143
144     /**
145      * Method getter for the genero
146      * @return the genero
147      */
148     public int getGenero() {
149         return this.genero;
150     }
151
152     /**
153      * Method setter for the genero
154      * @param genero
155      * @return void
156      */
157     public void setGenero(int genero) {
158         this.genero = genero;
159     }
160
161     /**
162      * Method getter for the aficion
163      * @return a String value
164      */
165     public String getAficion() {
166         return this.aficion;
167     }
168
169     /**
170      * Method setter for the ilusion
171      * @param aficion
172      * @return void
173      */
174     public void setAficion(String aficion) {
175         this.aficion = aficion;
176     }
177 }

```

Fig. 4.1.31: Fragmento 5, clase “UsuariosEntity.java”.

Paso 5: Codificamos la clase “UsuarioDTO.java”, usada para el DTO del RequestBody de la llamada HTTP. Incluir uso de javadoc antes de cada método.

Puede verse la codificación del DTO (similar a la de la Entity) en las siguientes figuras (desde Fig. 4.1.32 hasta Fig. 4.1.35).

```
UsuarioDTO.java X
Backend SpringBoot 2.7.0.java 11 > demo > src > main > java > com > example > demo > dto > UsuarioDTO.java
1 package com.example.demo.dto;
2 import java.io.Serializable;
3
4 public class UsuarioDTO implements Serializable{
5
6     private Long id;
7     private String nombre;
8     private String apellidos;
9     private Integer edad;
10    private boolean activado;
11    private Long telefono;
12    private int genero;
13    private String aficion;
14
15    /**
16     * Getter del ID
17     * @return id
18     */
19    public Long getId(){
20        return this.id;
21    }
22
23    /**
24     * Setter del ID
25     * @param id
26     */
27    public void setId(Long id){
28        this.id= id;
29    }
30
31    /**
32     * Getter del NOMBRE
33     * @return nombre
34     */
35    public String getNombre(){
36        return this.nombre;
37    }
38
39    /**
40     * Setter del NOMBRE
41     * @param nombre
42     */
43    public void setNombre(String nombre){
44        this.nombre= nombre;
45    }
}
```

Fig. 4.1.32: Fragmento 1, clase “UsuarioDTO.java”.

```

45  /**
46   * Getter del APELLIDOS
47   * @return apellidos
48   */
49  public String getApellidos(){
50      return this.apellidos;
51  }
52
53 /**
54  * Setter del APELLIDOS
55  * @param apellidos
56  */
57 public void setApellidos(String apellidos){
58     this.apellidos= apellidos;
59 }
60
61 /**
62  * Getter del EDAD
63  * @return edad
64  */
65 public Integer getEdad(){
66     return this.edad;
67 }
68
69 /**
70  * Setter del EDAD
71  * @param edad
72  */
73 public void setEdad(Integer edad){
74     this.edad=edad;
75 }
76
77 /**
78  * Setter del ACTIVADO
79  * @param activado
80  */
81 public boolean getActivado(){
82     return this.activado;
83 }

```

Fig. 4.1.33: Fragmento 2, clase “UsuarioDTO.java”.

```

85 /**
86  * Setter del ACTIVADO
87  * @param activado
88  */
89 public void setActivado(boolean activado){
90     this.activado=activado;
91 }
92
93 /**
94  * Getter del TELEFONO
95  * @param telefono
96  */
97 public Long getTelefono(){
98     return this.telefono;
99 }
100
101 /**
102  * Setter del TELEFONO
103  * @param telefono
104  */
105 public void setTelefono(Long telefono){
106     this.telefono=telefono;
107 }
108
109 /**
110  * Getter del GENERO
111  * @param genero
112  */
113 public int getGenero(){
114     return this.genero;
115 }
116
117 /**
118  * Setter del GENERO
119  * @param genero
120  */
121 public void setGenero(int genero){
122     this.genero=genero;
123 }
124 }

```

Fig. 4.1.34: Fragmento 3, clase “UsuarioDTO.java”.

```

127 /**
128  * Method getter for the aficion
129  * @return a String value
130  */
131 public String getAficion() {
132     return this.aficion;
133 }
134
135 /**
136  * Method setter for the ilusion
137  * @param aficion
138  * @return void
139  */
140 public void setAficion(String aficion) {
141     this.aficion = aficion;
142 }
143

```

Fig. 4.1.35: Fragmento 4, clase “UsuarioDTO.java”.

Paso 6: Codificamos la clase “Usuario.java”, que será un modelo intermedio entre DTO y Entity. Incluir uso de javadoc antes de cada método.

Puede verse la codificación del modelo (similar a la de la Entity y el DTO) en las siguientes figuras (desde Fig. 4.1.36 hasta Fig. 4.1.39).

```

User.java X UsuarioDTO.java
Backend SpringBoot 2.7.0 Java 11 > demo > src > main > java > com > example > demo > model > Usuario.java >
1 package com.example.demo.model;
2
3 public class Usuario {
4
5     private Long id;
6     private String nombre;
7     private String apellidos;
8     private Integer edad;
9     private boolean activado;
10    private Long telefono;
11    private int genero;
12    private String aficion;
13
14    /**
15     * Getter del ID
16     * @return id
17     */
18    public Long getId(){
19        return this.id;
20    }
21
22    /**
23     * Setter del ID
24     * @param id
25     */
26    public void setId(Long id){
27        this.id= id;
28    }
29
30    /**
31     * Getter del NOMBRE
32     * @return nombre
33     */
34    public String getNombre(){
35        return this.nombre;
36    }
37
38    /**
39     * Setter del NOMBRE
40     * @param nombre
41     */
42    public void setNombre(String nombre){
43        this.nombre= nombre;
44    }

```

Fig. 4.1.36: Fragmento 1, clase “Usuario.java”.

```

46  /**
47  * Getter del APELLIDOS
48  * @return apellidos
49  */
50 public String getApellidos(){
51     return this.apellidos;
52 }
53
54 /**
55 * Setter del APELLIDOS
56 * @param apellidos
57 */
58 public void setApellidos(String apellidos){
59     this.apellidos= apellidos;
60 }
61
62 /**
63 * Getter del EDAD
64 * @return edad
65 */
66 public Integer getEdad(){
67     return this.edad;
68 }
69
70 /**
71 * Setter del EDAD
72 * @param edad
73 */
74 public void setEdad(Integer edad){
75     this.edad=edad;
76 }
77
78 /**
79 * Getter del ACTIVADO
80 * @return activado
81 */
82 public boolean getActivado(){
83     return this.activado;
84 }

```

Fig. 4.1.37: Fragmento 2, clase “Usuario.java”.

```

86 /**
87 * Setter del ACTIVADO
88 * @param activado
89 */
90 public void setActivado(boolean activado){
91     this.activado=activado;
92 }
93
94 /**
95 * Getter del TELEFONO
96 * @return telefono
97 */
98 public Long getTelefono(){
99     return this.telefono;
100 }
101
102 /**
103 * Setter del TELEFONO
104 * @param telefono
105 */
106 public void setTelefono(Long telefono){
107     this.telefono=telefono;
108 }
109
110 /**
111 * Getter del GENERO
112 * @return genero
113 */
114 public int getGenero(){
115     return this.genero;
116 }
117
118 /**
119 * Setter del GENERO
120 * @param genero
121 */
122 public void setGenero(int genero){
123     this.genero=genero;
124 }

```

Fig. 4.1.38: Fragmento 3, clase “Usuario.java”.

```

126  /**
127  * Getter del AFICION
128  * @return genero
129  */
130 public String getAficion(){
131     return this.aficion;
132 }
133
134 /**
135 * Setter del AFICION
136 * @param aficion
137 */
138 public void setAficion(String aficion){
139     this.aficion=aficion;
140 }
141 
```

Fig. 4.1.39: Fragmento 4, clase “Usuario.java”.

Paso 7: Codificamos la clase “UsuarioRepository.java”, la cual dará la funcionalidad de acceso a la entity y permitirá realizar el CRUD. Incluir uso de javadoc antes de cada método.

Puede verse la codificación del “Repository” en la siguiente figura (Fig. 4.1.40).

```

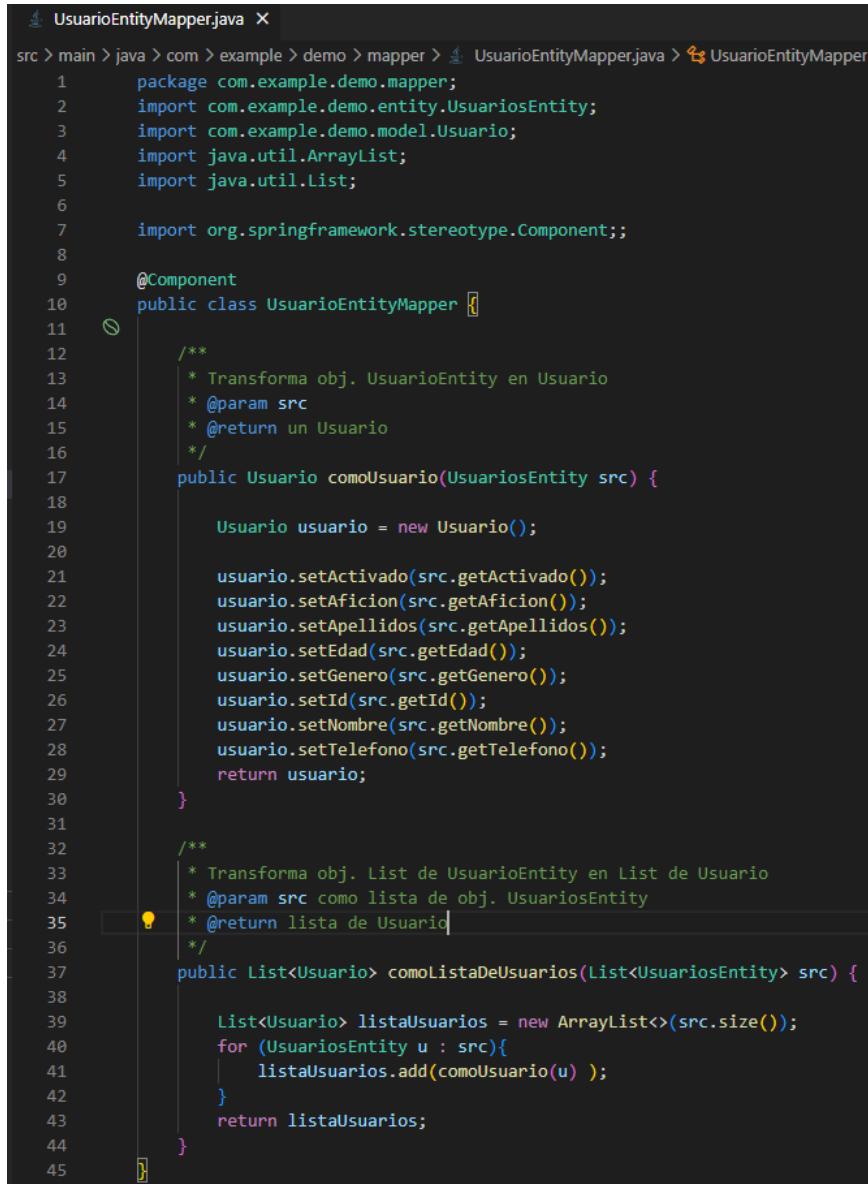
UsuariosRepository.java X
Backend SpringBoot 2.7.0 java 11 > demo > src > main > java > com > example > demo > repository > UsuariosRepository.java
1 package com.example.demo.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import com.example.demo.entity.UsuariosEntity;
6
7 // El primer parametro es el tipo de la clase de la cual quiero hacer el repositorio.
8 // El segundo parametro es el tipo del ID de esa clase.
9 // El repositorio es la interfaz a traves de la cual yo voy a escribir en BBDD esa entidad.
10 public interface UsuariosRepository extends JpaRepository<UsuariosEntity, Long> {
11
12 } 
```

Fig. 4.1.40: Fragmento único, clase “UsuariosRepository.java”.

Paso 8: Codificamos la clase “UsuarioEntityMapper.java”, con la que pasaremos de tipo de datos “UsuarioEntity” a “Usuario”, y vice versa. Incluir uso de javadoc antes de cada método.

Esta clase será utilizada en el Service, cuando una respuesta de BBDD necesite retornarse al controller como tipo Usuario.

Puede verse la codificación de “UsuarioEntity” en la siguiente figura (Fig. 4.1.41).



The screenshot shows a code editor with the file "UsuarioEntityMapper.java" open. The code defines a class "UsuarioEntityMapper" with two methods: "comoUsuario" and "comoListaDeUsuarios". Both methods include Javadoc comments describing their purpose and parameters. The code uses Java annotations like @Component and imports various Java packages.

```
src > main > java > com > example > demo > mapper > UsuarioEntityMapper.java > UsuarioEntityMapper
1 package com.example.demo.mapper;
2 import com.example.demo.entity.UsuariosEntity;
3 import com.example.demo.model.Usuario;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import org.springframework.stereotype.Component;
8
9 @Component
10 public class UsuarioEntityMapper {
11
12     /**
13      * Transforma obj. UsuarioEntity en Usuario
14      * @param src
15      * @return un Usuario
16      */
17     public Usuario comoUsuario(UsuariosEntity src) {
18
19         Usuario usuario = new Usuario();
20
21         usuario.setActivado(src.getActivado());
22         usuario.setAficion(src.getAficion());
23         usuario.setApellidos(src.getApellidos());
24         usuario.setEdad(src.getEdad());
25         usuario.setGenero(src.getGenero());
26         usuario.setId(src.getId());
27         usuario.setNombre(src.getNombre());
28         usuario.setTelefono(src.getTelefono());
29         return usuario;
30     }
31
32     /**
33      * Transforma obj. List de UsuarioEntity en List de Usuario
34      * @param src como lista de obj. UsuariosEntity
35      * @return lista de Usuario
36      */
37     public List<Usuario> comoListaDeUsuarios(List<UsuariosEntity> src) {
38
39         List<Usuario> listaUsuarios = new ArrayList<>(src.size());
40         for (UsuariosEntity u : src){
41             listaUsuarios.add(comoUsuario(u));
42         }
43     }
44 }

```

Fig. 4.1.41: Fragmento único, clase “UsuarioEntityMapper.java”.

Paso 9: Codificamos la clase “DemoService.java”, donde estarán todos los servicios que podremos llamar desde el Controller.

Puede verse la codificación de “DemoService” en las siguientes figuras (Fig. 4.1.42 – Fig. 4.1.45).

```

1 package com.example.demo.service;
2
3 import java.util.Comparator;
4 import java.util.List;
5 import java.util.Optional;
6 import com.example.demo.entity.UsuarioEntity;
7 import com.example.demo.mapper.UsuarioEntityMapper;
8 import com.example.demo.model.Usuario;
9 import com.example.demo.repository.UsuarioRepository;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.beans.factory.annotation.Value;
12 import org.springframework.stereotype.Service;
13
14
15 @Service
16 public class DemoService {
17
18     @Autowired
19     UsuarioRepository usuariosRepository;
20
21     @Autowired
22     private UsuarioEntityMapper usuarioEntityMapper;
23
24     @Value("${propiedad1}")
25     private String prefijo;
26     @Value("${propiedad2}")
27     private String sufijo;
28
29     public DemoService () {
30         this.prefijo = "relleno desde el constructor del servicio.";
31         this.sufijo = "relleno desde el constructor del servicio.";
32     }
33

```

Fig. 4.1.42: Fragmento 1, clase “DemoService.java”.

```

33 // servicio CREATE
34 public Long crearUsuario(Usuario src){
35
36     UsuariosEntity usuariosEntity = new UsuariosEntity();
37
38     // Entra un usuario con ID null porque aun no se asigne ID
39     // Por eso settear todo excepto el ID
40     usuariosEntity.setNombre(src.getNombre());
41     usuariosEntity.setApellido(src.getApellidos());
42     usuariosEntity.setedad(src.getEdad());
43     usuariosEntity.setActivado(src.getActivado());
44     usuariosEntity.setTelefono(src.getTelefono());
45     usuariosEntity.setGenero(src.getGenero());
46     usuariosEntity.setAficion(src.getAficion());
47
48     UsuariosEntity usuarioGuardadoEnDB = usuariosRepository.save(usuariosEntity);
49     Usuario usuario = usuarioEntityMapper.comoUsuario(usuarioGuardadoEnDB);
50     return usuario.getId();
51 }
52
53 // servicio READ
54 public List<Usuario> devolverTodosLosUsuarios(){
55
56     List<UsuariosEntity> usuariosEntity = usuariosRepository.findAll();
57     List<Usuario> usuarios = usuarioEntityMapper.comoListaDeUsuarios(usuariosEntity);
58     return usuarios;
59 }
60
61 // servicio UPDATE
62 public Usuario modificarUsuario(Long id, String valorModificada){
63
64     Optional<UsuarioEntity> usuarioOptional = usuariosRepository.findById(id);
65     if (usuarioOptional.isPresent()) {
66         // si me devuelve true ésta es este optional
67         // si me devuelve false no está en el optional
68         UsuariosEntity usuarioRecuperado = usuarioOptional.get();
69         usuarioRecuperado.setAficion(valorModificada);
70         UsuariosEntity usuarioEntityActualizado = usuariosRepository.save(usuarioRecuperado);
71         Usuario usuarioActualizado = usuarioEntityMapper.comoUsuario(usuarioEntityActualizado);
72         return usuarioActualizado;
73     } else{
74         return null;
75     }
76 }
77

```

Fig. 4.1.43: Fragmento 2, clase “DemoService.java”.

```

79     // servicio DELETE
80     public long eliminarUsuarioPorId(Long id){
81
82         Optional<UsuariosEntity> usuarioOpcional = usuariosRepository.findById(id);
83         if (usuarioOpcional.isPresent() ){
84             // si me devuelve true esta este optional
85             // si me devuelve false no esta en el optional
86             UsuariosEntity usuarioRecuperado = usuarioOpcional.get();
87             usuariosRepository.delete(usuarioRecuperado);
88             return id;
89         }
90         else{
91             return null;
92         }
93     }
94
95
96
97     /***** Ordenamiento de los usuarios segun valor de RequesParam *****/
98     **** OrdeanListaUsuarios (List<Usuario> usuarios, String ordenadosPor){
99
100        Boolean ordenado = false;
101        if(ordenadosPor.equals(anObject: "id")){
102            usuarios.sort(Comparador.comparing(Usuario::getId));
103            ordenado = true;
104        }
105        if(ordenadosPor.equals(anObject: "nombre")){
106            usuarios.sort(Comparador.comparing(Usuario::getNombre));
107            ordenado = true;
108        }
109        if(ordenadosPor.equals(anObject: "edad")){
110            usuarios.sort(Comparador.comparing(Usuario::getEdad));
111            ordenado = true;
112        }
113        if(!ordenado){
114            usuarios.sort(Comparador.comparing(Usuario::getId));
115        }
116    }
117    return usuarios;
118 }
```

Fig. 4.1.44: Fragmento 3, clase “DemoService.java”.

```

122     **** Consiguendo valores desde el constructor del service ****
123     **** vs ****
124     **** Consiguendo valores desde el fichero "application.property" ****
125     **** **** ****
126
127
128     public String consiguePrefijoSufijoDelConstructor(){
129         return ("prefijo: " + prefijo + "\nsufijo: " + sufijo + "\n");
130     }
131
132     public String consiguePrefijoSufijoDeApplicationProperties(){
133         return ("prefijo: " + prefijo + "\nsufijo: " + sufijo + "\n");
134     }
135
136 }
137 
```

Fig. 4.1.45: Fragmento 4, clase “DemoService.java”.

Paso 10: Codificamos la clase “UsuarioMapper.java”, con la que pasaremos de tipo de datos “UsuarioDTO” a “Usuario”, y vice versa.

Esta clase será utilizada en el Controller, cuando un dato de entrada al controlador venga en el RequestBody (como DTO), o cuando se quiera devolver algo por el controlador (como DTO) y del servicio nos venga modelado como Usuario.

Puede verse la codificación de “UsuarioMapper” en las siguientes figuras (Fig. 4.1.46 – Fig. 4.1.47).

```

Backend SpringBoot 2.7.0.java 11 > demo > src > main > java > com > example > demo > mapper > UsuarioMapper.java
1 package com.example.demo.mapper;
2
3 import org.springframework.stereotype.Component;
4
5 import com.example.demo.dto.UsuarioDTO;
6 import com.example.demo.model.Usuario;
7
8 @Component
9 public class UsuarioMapper{
10
11     /**
12      * Transforma un UsuarioDTO a Usuario
13      * @param src
14      * @param aficion
15      * @return obj. tipo Usuario
16      */
17     public Usuario comoUsuario(UsuarioDTO src, String aficion) {
18
19         if(src == null){
20             return null;
21         }
22         Usuario usuario = new Usuario();
23         usuario.setActivado(src.getActivado());
24         usuario.setAficion(aficion);
25         usuario.setApellidos(src.getApellidos());
26         usuario.setEdad(src.getEdad());
27         usuario.setGenero(src.getGenero());
28         usuario.setId(src.getId());
29         usuario.setNombre(src.getNombre());
30         usuario.setTelefono(src.getTelefono());
31         return usuario;
32     }
33 }

```

Fig. 4.1.46: Fragmento 1, clase “UsuarioMapper.java”.

```

34     /**
35      * Transforma un Usuario a UsuarioDTO
36      * @param src
37      * @return obj. tipo UsuarioDTO
38      */
39     public UsuarioDTO comoUsuarioDTO(Usuario src) {
40
41         if(src == null){
42             return null;
43         }
44         UsuarioDTO usuarioDTO = new UsuarioDTO();
45         usuarioDTO.setActivado(src.getActivado());
46         usuarioDTO.setAficion(src.getAficion());
47         usuarioDTO.setApellidos(src.getApellidos());
48         usuarioDTO.setEdad(src.getEdad());
49         usuarioDTO.setGenero(src.getGenero());
50         usuarioDTO.setId(src.getId());
51         usuarioDTO.setNombre(src.getNombre());
52         usuarioDTO.setTelefono(src.getTelefono());
53         return usuarioDTO;
54     }
55 }

```

Fig. 4.1.47: Fragmento 2, clase “UsuarioMapper.java”.

Paso 11: Codificamos la clase “DemoController.java”, donde crearemos todos los métodos controladores que nos permitirán realizar todas las llamadas HTTP descritas en el apartado anterior (apartado 4.2.c).

Puede verse la codificación de “DemoController” en las siguientes figuras (Fig. 4.1.48 – Fig. 4.1.50).

```

1 package com.example.demo.controller;
2
3 import java.util.Comparator;
4 import java.util.List;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.http.HttpStatus;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import org.springframework.web.bind.annotation.PutMapping;
13 import org.springframework.web.bind.annotation.RequestBody;
14 import org.springframework.web.bind.annotation.RequestMapping;
15 import org.springframework.web.bind.annotation.RequestParam;
16 import org.springframework.web.bind.annotation.RestController;
17
18 import com.example.demo.dto.UsuarioDTO;
19 import com.example.demo.entity.UsuarioEntity;
20 import com.example.demo.mapper.UsuarioMapper;
21 import com.example.demo.model.Usuario;
22 import com.example.demo.service.DemoService;
23
24 import io.micrometer.core.ipc.http.HttpSender.Response;
25
26 @RestController
27 @RequestMapping("/")
28 public class DemoController {
29
30     @Autowired
31     private DemoService miServicio; // Servicio sin constructor
32     private DemoService miServicioConConstructor = new DemoService(); // Servicio con constructor
33
34     @Autowired
35     private UsuarioMapper usuarioMapper; // Componente sin constructor
36
37     // Llamada CREATE: crea un usuario, con llamada HTTP tipo POST.
38     @PostMapping("crearusuario/{detalleUsuario}")
39     public Long crearUsuario (@PathVariable String detalleUsuario,
40                               @RequestBody UsuarioDTO usuarioDTO){
41
42         Usuario usuario = this.usuarioMapper.comoUsuario(usuarioDTO, detalleUsuario);
43         Long idNuevoUsuario = this.miServicio.crearUsuario(usuario);
44         return idNuevoUsuario;
45     }

```

Fig. 4.1.48: Fragmento 1, clase “DemoController.java”.

```

47
48
49     // Llamada READ: lee todos los usuarios, con llamada HTTP tipo GET.
50     @GetMapping
51     public ResponseEntity<List<Usuario>> mostrarTodosLosUsuarios (@RequestParam String ordenadosPor){
52
53         List<Usuario> usuarios = miServicio.devolverTodosLosUsuarios();
54         usuarios = miServicio.ordenarListaUsuarios(usuarios, ordenadosPor);
55         ResponseEntity response = new ResponseEntity(usuarios, HttpStatus.OK);
56         return response;
57     }
58
59     // Llamada UPDATE: modifica un usuario con llamada HTTP tipo PUT.
60     @PutMapping("/modificarDetalleUsuario/{idUsuario}/{nuevoDetalle}")
61     public ResponseEntity<Void> modificarDetalleDeUsuario (@PathVariable Long idUsuario, @PathVariable String nuevoDetalle){
62
63         Usuario usuario = miServicio.modificarUsuario(idUsuario, nuevoDetalle);
64         if (usuario != null){
65             return ResponseEntity.noContent().build();
66         }
67         else{
68             ResponseEntity responseEntity = new ResponseEntity("Imposible actualizar, el elemento no existe.", HttpStatus.BAD_REQUEST);
69             return responseEntity;
70         }
71     }
72
73     // Llamada DELETE: elimina un usuario con llamada HTTP tipo POST.
74     @PostMapping("eliminarUsuario/{id}")
75     public ResponseEntity<Void> eliminarUsuario (@PathVariable Long id){
76
77         Long idUserEliminado = miServicio.eliminarUsuarioPorId(id);
78         if (idUserEliminado != null){
79             return ResponseEntity.noContent().build();
80         }
81
82         return ResponseEntity.notFound().build();
83     }
84
85
86
87
88
89
90
91
92
93
94

```

Fig. 4.1.49: Fragmento 2, clase “DemoController.java”.

```

88  **** PRUEBAS EXTRA ****
89  ***** Consiguiendo valores desde el constructor del service ****
90  ***** ***** VS ****
91  ***** Consiguiendo valores desde el fichero "application.property" ****
92  ***** ****
93
94  @GetMapping("/getServiceConConstructor")
95  public String getServiceConConstructor (){
96      return this.miServicioConConstructor.consiguePrefijoSufijoDelConstructor();
97  }
98
99  @GetMapping("/getServiceConAutowired")
100 public String getServiceConAutowired (){
101     return this.miServicio.consiguePrefijoSufijoDeApplicationProperties();
102 }
103
104

```

Fig. 4.1.50: Fragmento 3, clase “DemoController.java”.

Paso 12: Añadir 2 propiedades en el fichero “application.properties”, las cuales se llamaban en el fichero del Service (paso 9).

Puede verse la codificación de las nuevas propiedades del “application.properties” en la siguiente figura (Fig. 4.1.51).

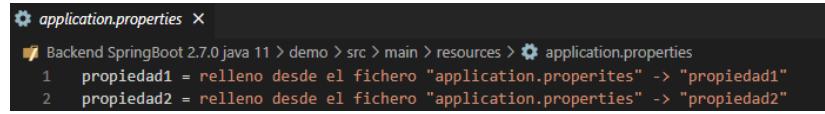


Fig. 4.1.51: Incluimos 2 propiedades, que se usarán en el constructor del Service.

Paso 13: Ejecutar el proyecto backend, aunque aún no esté conectada la base de datos. Comprobar que la ejecución del servidor local no lanza ningún error (Fig. 4.1.52).

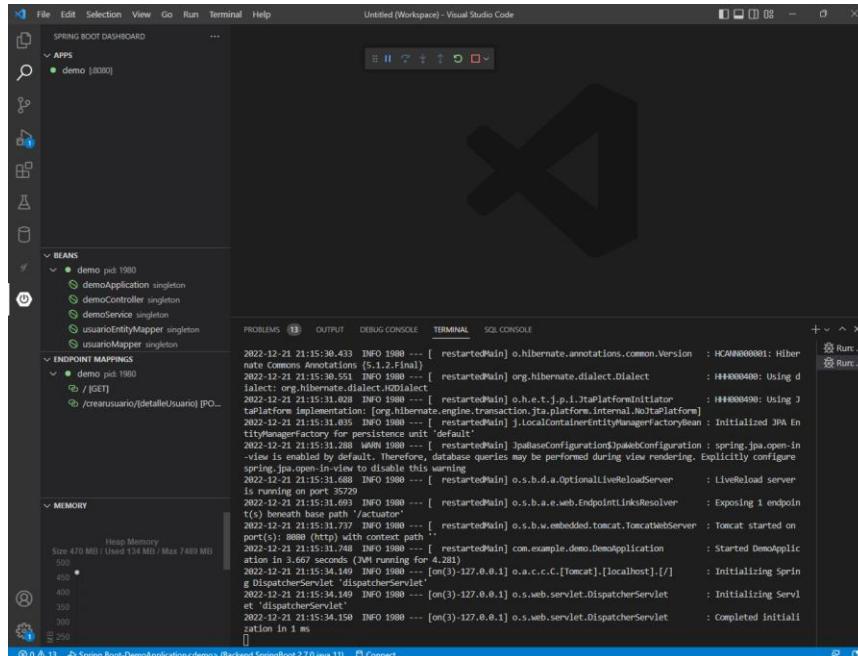


Fig. 4.1.52: Ejecución del servidor local sin errores.

e) Conexión del proyecto Backend a una base de datos “H2”.

La base de datos, con la que realizaremos las primeras pruebas de conectividad en el backend, será de tipo H2. Este tipo de base de datos se usa siempre en local y el estado de los datos se almacena de forma volátil (cuando apagamos el servidor local los datos se borran).

Como ya hemos podido apreciar, también se usa JPA (para mapear en forma de objetos Java las tablas y relaciones de la base de datos).

Como ya vimos en el apartado anterior, en la arquitectura del proyecto tenemos 5 secciones:

- controller
- dto
- entity
- repository
- service

Solo los directorios “entity” y “repository” estarán relacionadas con “bases de datos”.

En el directorio “entity” tendremos:

- Las clases Java (para crear las entity). Aquí se crearán las tablas de nuestra base de datos.
- Los atributos (de cada entity) y sus métodos getters/setters (de los atributos).

En el directorio “repository” tenemos:

- Las interfaces Java que crean los repositories: por cada “entity” creada habrá una interfaz “repository”.
- Cada interfaz, que en su cabecera de interfaz, extiende de “JpaRepository”.
- Cada interfaz “repository”, tendrá entre sus imports una clase “entity”, sobre la que ejecutará las operaciones CRUD necesarias.

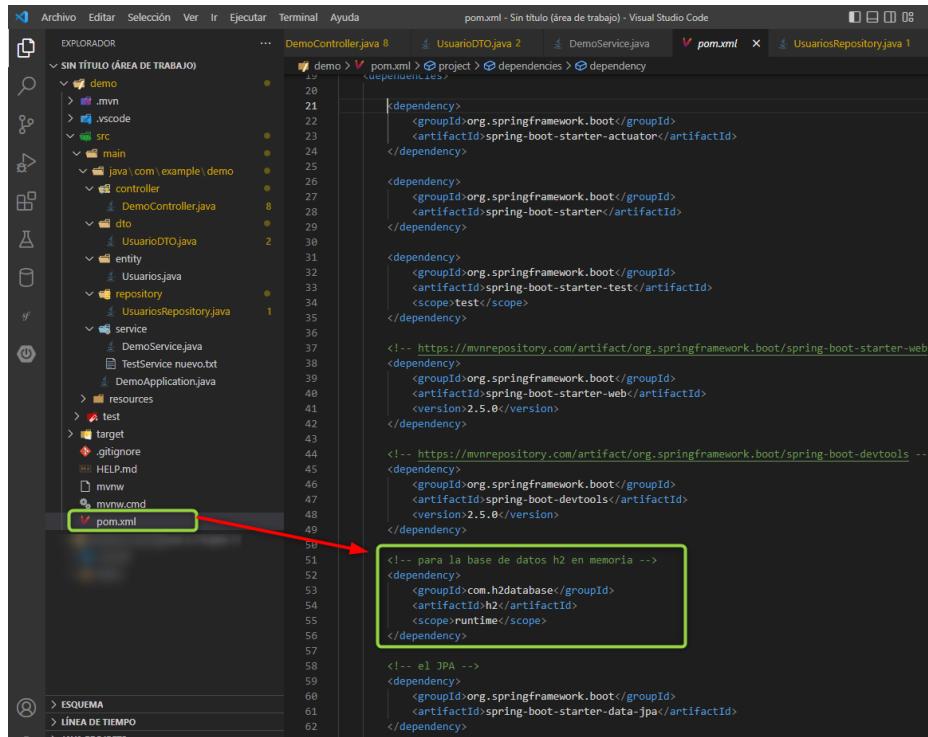
Por tanto, en el directorio “entity” tendremos cada una de las entidades o tablas de la base de datos. En el directorio “repository”, por otra parte, tendremos asociada la “entity” previa a este “repository”, y será este repositorio el que nos permite realizar las operaciones CRUD sobre la tabla que esté importando.

En última instancia, el “repository” se usará en el “service”, y a través de la llamada al método concreto del repository se consiguen todas las funcionalidades del CRUD para una tabla dada (“entity”).

Una vez tenemos claramente repasados los conceptos de “entity”, “jpa” y “repository”, abordemos la conectividad con la base de datos H2.

La dependencia que nos permite añadir una base de datos H2 al proyecto se consigue añadiendo un fragmento de código en el fichero “pom.xml”. Esta dependencia descargará un fichero .JAR desde “mavenrepository.com”, a nuestro directorio de artefactos local (normalmente un directorio llamado “.m2”).

Puede verse en la siguiente Fig. 4.1.53 el código copiado desde “mavenrepository.com”, el cual descarga la base de datos H2.



```

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- el JPA -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

```

Fig. 4.1.53: Configuración, en el fichero pom.xml, de la dependencia Maven para descargar el gestor de base de datos H2.

Después, configuraremos los parámetros de la Base de Datos H2 en nuestro fichero “application.properties”, parámetros tales como: username, password, etc. Véase a Fig. 4.1.54 para más información.

```

application.properties - Sin título (área de trabajo) - Visual Studio Code
application.properties
BackendSpringBoot 2.7.0.java 11 > demo > main > resources > application.properties
1 propiedad1 = relleno desde el fichero "application.properties" -> "propiedad1"
2 propiedad2 = relleno desde el fichero "application.properties" -> "propiedad2"
3
4 #parametros de configuración para la BBDD en memoria H2
5 spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1
6 spring.datasource.username=sa
7 spring.datasource.password=sa
8

```

Fig. 4.2.54: Configuración de las credenciales para la Base de Datos H2, desde el fichero “application.properties” del proyecto Spring Boot.

Cuando tenga añadido el código de la nueva dependencia en el fichero “pom.xml”, ejecute el proyecto y confirme que se ejecuta sin errores en consola. Este proceso descargará oficialmente la nueva dependencia en nuestro repositorio de artefactos local. Si todo ha ido bien pare el servidor local.

Una vez se ha incluido correctamente la base de datos H2, se debe probar el correcto funcionamiento de la misma, mediante una serie de consultas CRUD lanzadas desde “Insomnia Core”.

Por favor, vuelva a ejecutar el servidor local del backend en Visual Studio Code y, después, abra el Insomnia Core (que contenía las 4 consultas CRUD ya creadas en el apartado 4.2 c).

Finalmente, cubra todas las pruebas del CRUD siguiendo la siguiente lista de pruebas (9 llamadas HTTP desde cliente Insomnia Core):

- 1) Crear el primer usuario en BB.DD. (ver Fig. 4.1.55):
 - Se usa una “PathVariable”, para indicar el valor del atributo “afición” (su valor será “futbol”).
 - Se usa un “ResquestBody”, para asignar valores al resto de atributos del nuevo usuario que vamos a crear (en formato JSON).

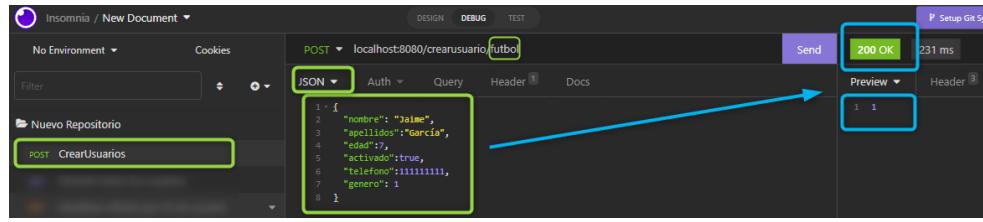


Fig. 4.1.55: Creación de 1er usuario en bb.dd. H2. Usada llamada POST con JSON en RequestBody para 6 atributos y PathVariable para 1 atributo. Respuesta del servidor si fue bien con 200 OK y el nº 1 que es el ID asignado al nuevo usuario creado (el primero).

2) Crear el segundo usuario en BB.DD. (ver Fig. 4.1.56):

- Misma “PathVariable” que usuario anterior, creado en el paso 1.
- Ahora solo se modifica del “RequestBody” los parámetros “nombre”, “apellidos” y “edad”.

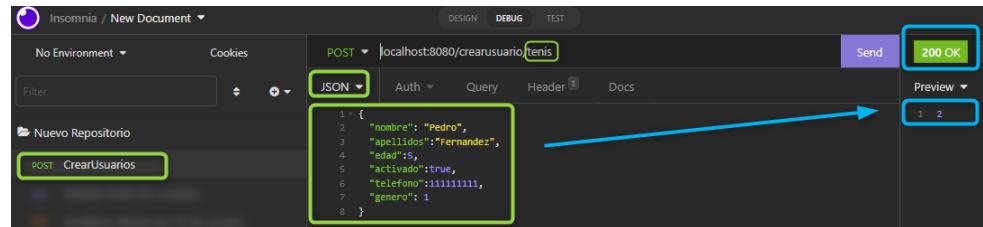


Fig. 4.1.56: Creación del 2º usuario en bb.dd. H2. Similar al anterior llamada en RequestBody y PathVariable. La llamada exitosa devuelve un 200 OK y el nº 2 que corresponde al ID del nuevo elemento creado en bb.dd. (el 2º usuario).

3) Mostrar usuarios existentes en BB.DD. ordenados por ID (ver Fig. 4.1.57):

- La bb.dd. solo contiene los usuarios con ID 1 y 2.
- Se muestran todos los usuarios, ordenados por ID.
- El atributo por el que se ordenan los elementos mostrados se especifica en un RequestParam de la URL (el QueryString en la URL especifica ordenamiento por id).
- Respuesta exitosa retorna cod. 200 OK y la lista con todos los elementos (ordenada por id, de menor a mayor).

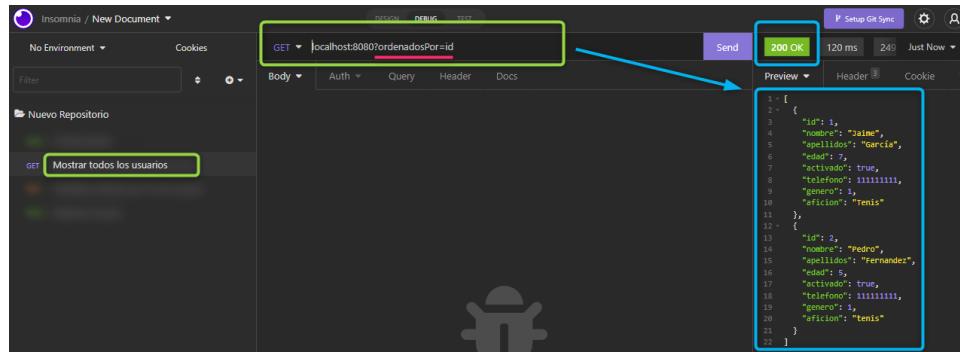


Fig. 4.1.57: Consulta de todos los usuarios almacenados en la bb.dd. Si la respuesta es exitosa, entonces devuelve 200 OK y una lista JSON de los 2 primeros usuarios ordenados por el ID de menor a mayor.

- 4) Mostrar usuarios existentes en BB.DD. ordenados por EDAD (ver Fig. 4.1.58):
- La bb.dd. solo contiene los usuarios con ID 1 y 2.
 - Se muestran todos los usuarios, ordenados por EDAD.
 - El atributo por el que se ordenan los elementos mostrados se especifica en un RequestParam de la URL (el QueryString en la URL especifica ordenamiento por edad).
 - Respuesta exitosa retorna cod. 200 OK y la lista con todos los elementos (ordenada por EDAD, de menor a mayor).

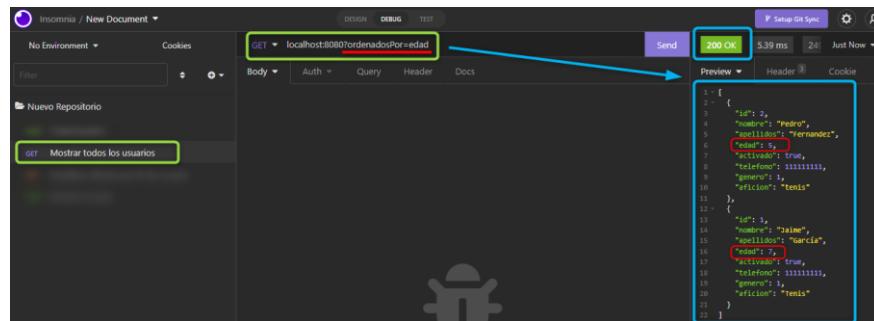


Fig. 4.1.58: Consulta de todos los usuarios almacenados en la bb.dd. Si la respuesta es exitosa, entonces devuelve 200 OK y una lista JSON de los 2 primeros usuarios ordenados por el EDAD de menor a mayor.

- 5) Modificar el atributo “afición” del usuario 3, que no existe, (ver Fig. 4.1.59):
- Se usa PathVariable, para enviar el valor modificado del atributo que deseó modificar.
 - Devuelve código de error 400 (ya que el elemento no existe).
 - Generará una respuesta con mensaje de error personalizado.

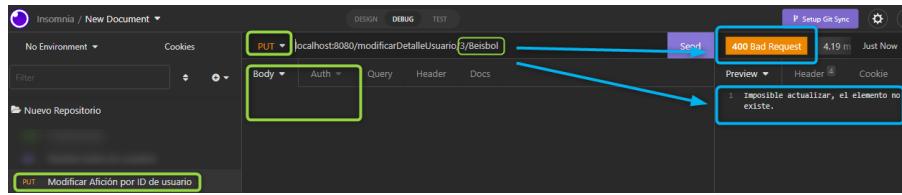


Fig. 4.1.59: Intenta modificar un atributo de un usuario que NO existe en bb.dd. Se envía como PathVariable en la URL el ID del usuario seleccionado y el valor modificado del atributo “afición”. La respuesta nunca será exitosa, y devolverá cod. Error 400 y un mensaje personalizado.

- 6) Modificar el campo “afición” del usuario 2, que sí existe (ver Fig. 4.1.60):
 - Se usa PathVariable para enviar el valor modificado del atributo que deseo modificar.
 - Atributo “afición” del usuario con Id=2, cambiará su valor de “Tenis” por el de “Beisbol”.
 - Devuelve un código exitoso 204 NO CONTENT (porque este usuario sí existe).
 - No genera ninguna respuesta tras la modificación exitosa (sin contenido).

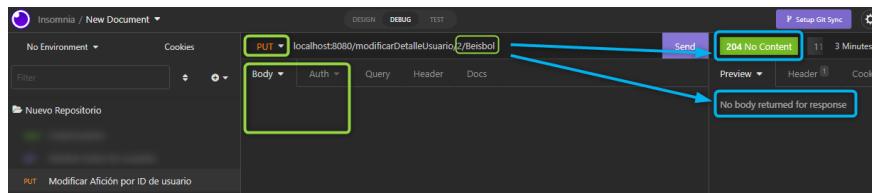


Fig. 4.1.60: Intenta modificar un atributo de un usuario que SÍ existe en bb.dd. Se envía como PathVariable en la URL el ID del usuario seleccionado y el valor modificado del atributo “afición”. La respuesta es exitosa, y devolverá cod. 204 NO CONTENT.

- 7) Tras modificar atributo “afición” del usuario con ID=2, consulto valores de todos los usuarios (ver Fig. 4.1.61):
 - Antes todos los usuarios (los 2 existentes) tenían atributo “afición” = “tenis”.
 - Ahora en el usuario con Id=2, el atributo “afición” tiene modificado su valor por “Beisbol”.
 - Se especifica en el QueryString de la URL que se mostrarán todos los usuarios ordenados por id, de menos a mayor valor de id.

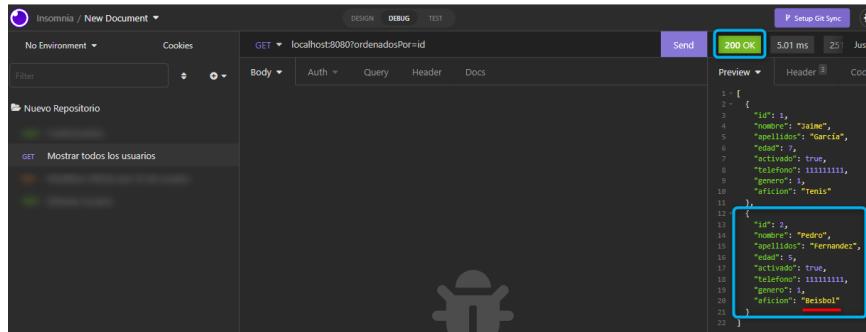


Fig. 4.1.61: Respuesta 200 OK con los valores de todos los usuarios, resaltando la modificación del atributo “afición” del usuario con Id=2.

8) Eliminar al usuario 2 (ver Fig. 4.1.62):

- Se usa PathVariable para enviar el valor del ID del usuario que deseamos eliminar de la base de datos (el que tiene id=2).
- Devuelve un código exitoso 204 NO CONTENT.
- No genera ninguna respuesta tras la modificación exitosa (sin contenido).

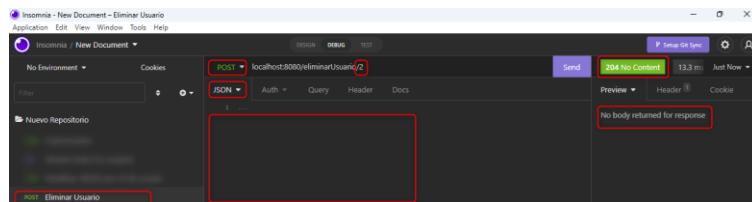


Fig. 4.1.62: Llamada HTTP de eliminación del usuario con Id 2.

9) Por último, mostrar todos los usuarios existentes en la base de datos, y comprobar que solo tenemos al usuario 1 y no aparece el usuario 2 (ver Fig. 4.1.63):

- La bb.dd. solo debe contener al usuario con ID 1.
- Se muestran todos los usuarios, ordenados por ID.
- El atributo por el que se ordenan los elementos mostrados se especifica en un RequestParam de la URL (el QueryString en la URL especifica ordenamiento por ID).
- Respuesta exitosa retorna cod. 200 OK y la lista con todos los elementos (ordenada por ID, de menor a mayor).

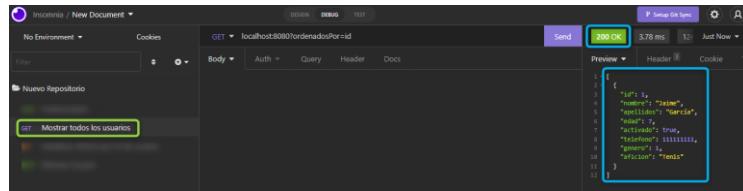


Fig. 4.1.63: Muestro todos los usuarios de la base de datos y solo parece el usuario con id 1, tras borrar el usuario 2.

f) Conexión del proyecto Backend a una base de datos Cloud (“Firebase”).

Hasta el momento teníamos funcionando correctamente el proyecto backend, con:

- Una base de datos (en adelante “BBDD”) de tipo H2.
- Usábamos JPA.

Sin embargo, para un uso profesional, no es posible usar una BBDD que sea volátil en cuanto a persistencia de los datos.

Por todo ello, en este apartado 4.2.f) se va a configurar el proyecto Backend (con Spring Boot) para que use la BBDD Firebase.

Al acabar este apartado no volveremos a utilizar una BBDD local con JPA, pero para simplificar la configuración se dejarán coexistiendo las dependencias de Firebase y H2; tampoco se quitarán las clases ya creadas relacionadas con JPA. Es decir, los cambios tendrán una naturaleza meramente de “evolutiva” y no se quitan los desarrollos previos.

Concluyendo, puede decirse que al acabar este apartado (y durante todo el apartado 4.3) tendremos un producto mínimo viable de BBDD y consultas REST, con tecnología “Firebase Cloud Firestore”.

Por favor, seguir los siguientes 28 pasos:

Paso 1: Entra en tu navegador favorito (aquí usaremos Firefox).

Luego iniciar sesión de Gmail con tu usuario (para tener activa la sesión personal de Google en todo el navegador).

Paso 2: Entra en la URL de Firebase <https://firebase.google.com>, donde se verá una pantalla similar a la de la imagen siguiente (Fig. 4.1.64).

En esta primera pantalla de Firebase, ir a la parte superior derecha y clicar en el botón “Ir a consola”.

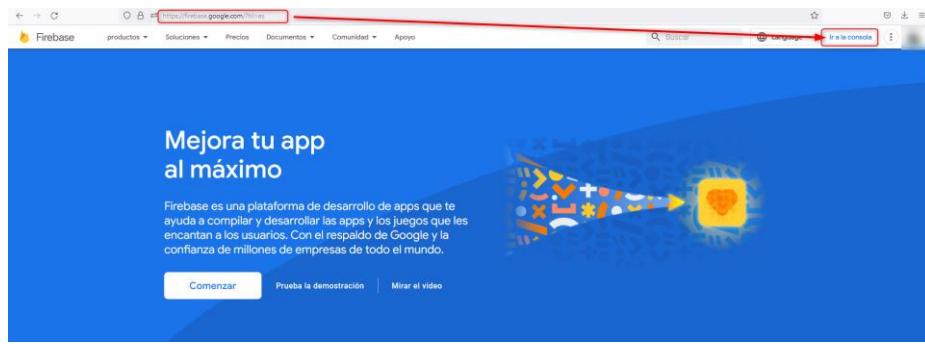


Fig. 4.1.64: Pantalla inicial Firebase, antes de clicar botón “ir a la consola”.

Paso 3: Ahora, si aun no creaste ningún proyecto con Firebase, puedes encontrarte una pantalla similar a la de la siguiente imagen (Fig. 4.1.65).

Desde esta nueva pantalla (haya creado algún proyecto o no), clicar en el botón “Crear un proyecto”.

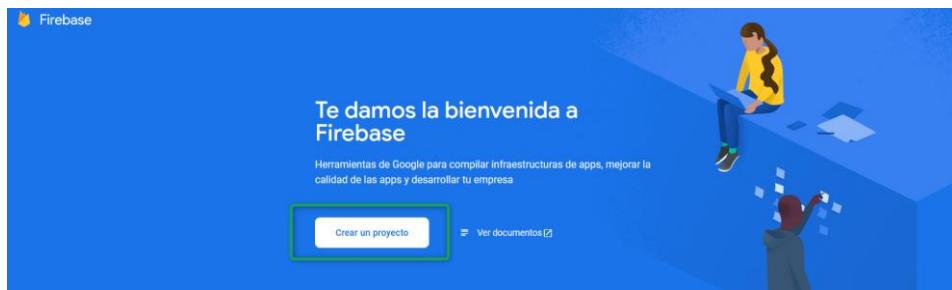


Fig. 4.1.65: Pantalla de “Consola” con Firebase. Para continuar, clicar en el botón “Crear un proyecto”.

Paso 4: Introducir el nombre del nuevo proyecto que se vas a crear (ver Fig. 4.1.66), y clicar en “Continuar”. Aceptar incluir “Google Analytics”, durante el proceso de creación del nuevo proyecto Firebase (ver Fig. 4.1.67).

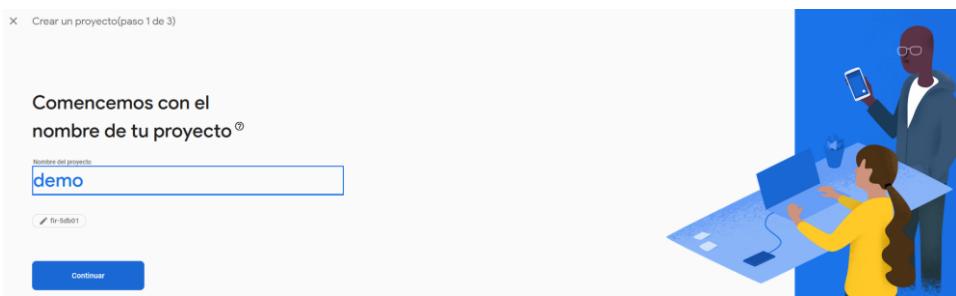


Fig. 4.1.66: Introducción del nombre del nuevo proyecto en Firebase.

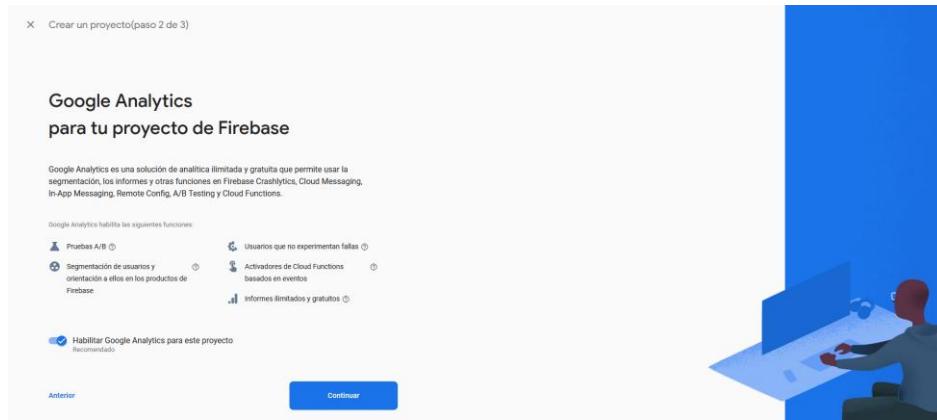


Fig. 4.1.67: Selección de Google Analytics, durante la creación del nuevo proyecto Firebase.

Paso 5: Configurar Google Analytics (ver Fig. 4.1.68), y llegar al estado final de creación del proyecto Firebase (ver Fig. 4.1.69).

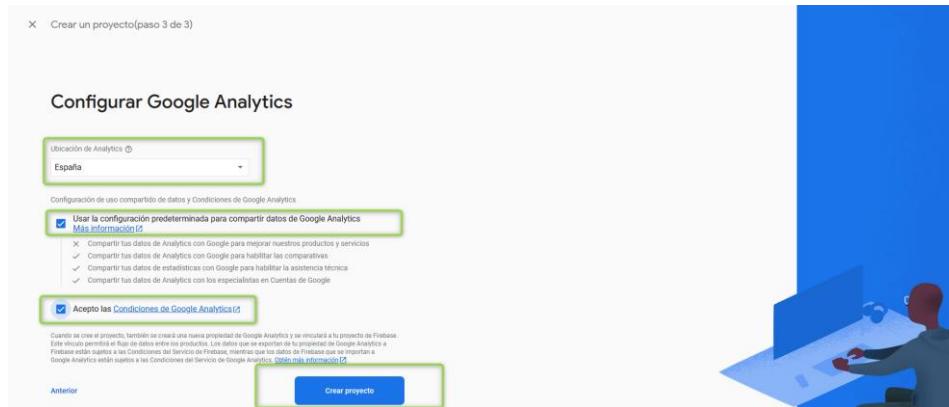


Fig. 4.1.68: Selección de opciones de Google Analytics, durante la creación del nuevo proyecto Firebase.

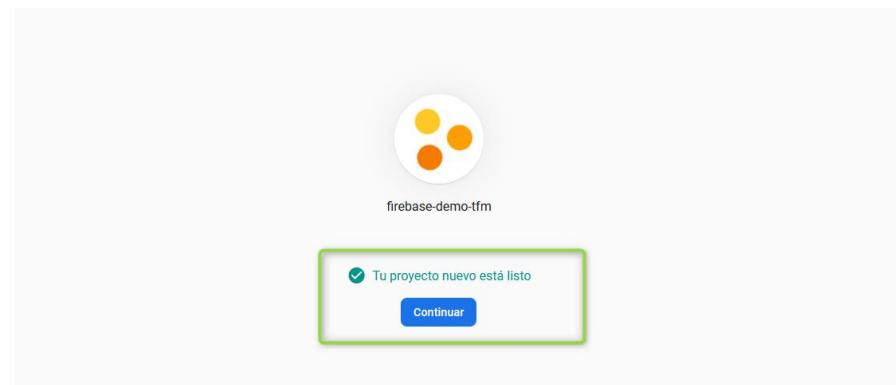


Fig. 4.1.69: Creación terminada del nuevo proyecto Firebase.

Paso 6: Una vez creado el nuevo proyecto, dentro del mismo y teniendo seleccionado el proyecto actual (“demo”), elegir el producto Cloud Store (ver Fig. 4.1.70):

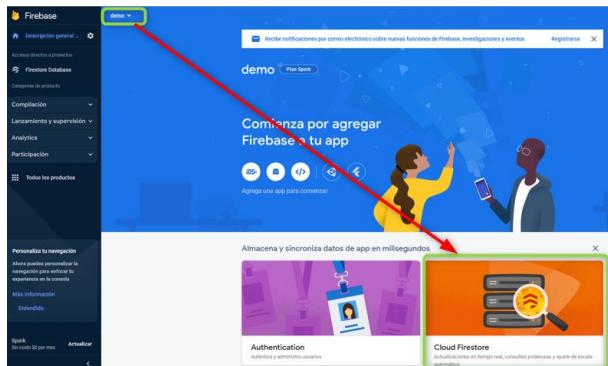


Fig. 4.1.70: Panel de configuración del nuevo proyecto, donde debemos seleccionar sección “Cloud Firestore”.

Paso 7: Crear una nueva BBDD, clicando en el botón central “Crear base de datos”(ver Fig. 4.1.71):



Fig. 4.1.71: Panel de Cloud Firestore, clicar en botón “crear base de datos”.

Paso 8: En cuanto a los permisos de acceso y lectura de la BBDD que nos ofrece, usaremos la 2^a opción “Comenzar en modo de prueba” (ver Fig. 4.1.72):

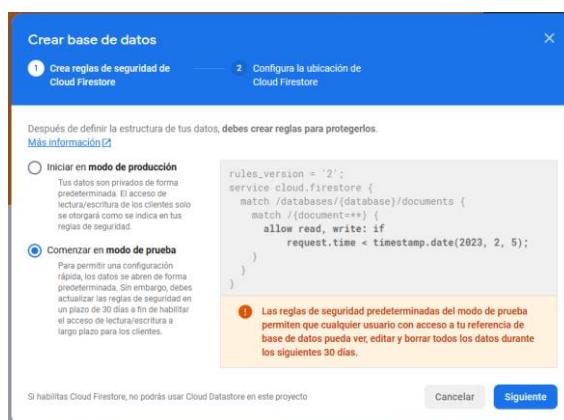


Fig. 4.1.72: Selección del modo prueba al comenzar la nueva BBDD.

Paso 9: Seleccionar la localización y creación de la nueva BBDD.

Seleccione la opción más cercana a usted para reducir la latencia con la BBDD (ver Fig. 4.1.73).

Al clicar en “Habilitar” la nueva BBDD habrá sido creada (Fig. 4.1.74).

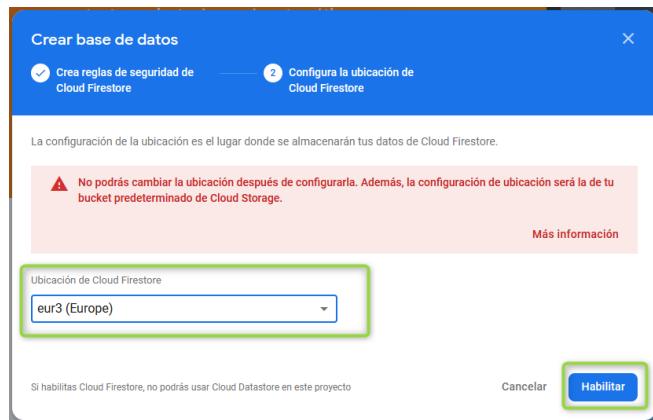


Fig. 4.1.73: Selección de localización para la nueva BBDD, durante el proceso de creación.



Fig. 4.1.74: Creación de la nueva BBDD exitosamente.

La nueva BBDD ha sido creada, y su referencia a ella será “fir-5db01” (como indica la imagen anterior).

Paso 10: Ahora conectar el backend Java con la nueva BBDD cloud. Para ello, clicar en “Ir a la documentación”, como se ve señalado en la esquina superior derecha de Fig. 4.1.74.

Nota: Firebase es una BBDD No-SQL, por eso en la interfaz se pueden crear documentos (Firebase trabaja con documentos, no con tablas).

Paso 11: En la nueva ventana de documentación vamos al campo de búsqueda, y ahí escribir: “spring boot con cloud firestore”. Puede ver las posibles pantallas que se encontrará en las imágenes Fig. 4.1.75 y Fig. 4.1.76.



Fig. 4.1.75: Campo de búsqueda de documentación por palabras clave.

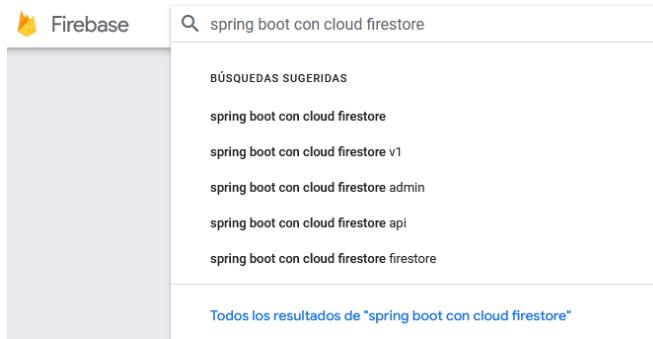


Fig. 4.1.76: Autocompletados propuestos durante búsqueda de documentación.

Paso 12: Para conectar nuestro backend Java con Firebase, usar la documentación de la primera coincidencia encontrada (llamada “Add the Firebase Admin SDK to your server”). En la Fig. 4.1.77 tiene la lista de coincidencias encontradas.

La URL de la documentación es la siguiente:

<https://firebase.google.com/docs/admin/setup#java>

Add the Firebase Admin SDK to your server
Firebase › Documentation › Admin SDK
Dec 5, 2022 ... Generate and verify Firebase auth tokens. Access Google Cloud resources like Cloud Storage buckets and Cloud Firestore databases associated with ...

Java samples for Cloud Firestore | Firebase
Firebase › Documentation › Firestore
To clone the source: Run git clone https://github.com/googleapis/java-firebase . Browse to the samples/snippets directory to ...

Paginate data with query cursors | Firestore | Firebase
Firebase › Documentation › Firestore
Dec 5, 2022 ... Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser. With query cursors in Cloud Firestore, ...

Use the Cloud Firestore REST API | Firebase
Firebase › Documentation › Firestore
https://www.googleapis.com/auth.firebaseio . If you authenticate your requests with a service account and a Google Identity OAuth 2.0 token, Cloud Firestore ...

Authorize send requests | Firebase Cloud Messaging
Firebase › Documentation › FCM
To authorize access to FCM, request the scope https://www.googleapis.com/auth/firebase.messaging . To add the access token to an HTTP request header: Add the ...

Connect your app to the Cloud Firestore Emulator | Firebase Local ...
Firebase › Documentation › Local Emulator Suite
Dec 5, 2022 ... Local Emulator Suite supports emulation of real Firebase projects and demo projects. Project type, Features, Use with emulators. Real. A real ...

Fig. 4.1.77: Lista de documentaciones encontradas. Selección de primera documentación para lectura completa.

Paso 13: Dentro de la documentación ir a la sección “Agregar el SDK” y seleccionar el lenguaje Java.

Luego copiar en el portapapeles el código que hay bajo la sección de Maven (ver Fig. 4.1.78). Este código será pegado en el firechero pom.xml del proyecto backend, como nueva dependencia.

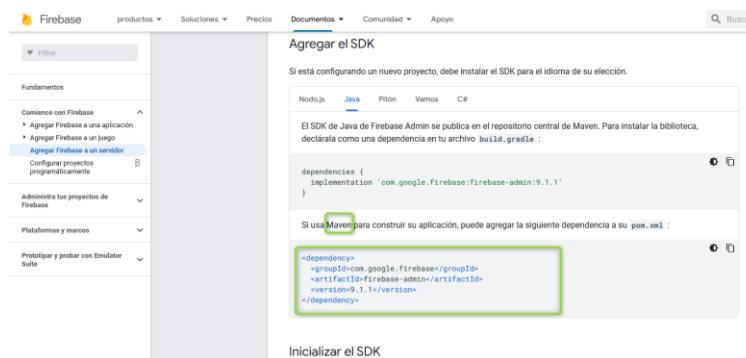


Fig. 4.1.78: Código que nos interesa copiar de la documentación Firebase en el “pom.xml” del proyecto backend.

Paso 14: En el tutorial, más adelante nos encontramos la configuración para la variable de entorno en Windows con PowelShell. Véase Fig. 4.2.79.

Lo único que necesitamos de esta nueva sección del tutorial es clicar en “Cuentas de servicio”, que es un texto que nos aparece en azul, aproximadamente a la mitad de la Fig. 4.1.79.

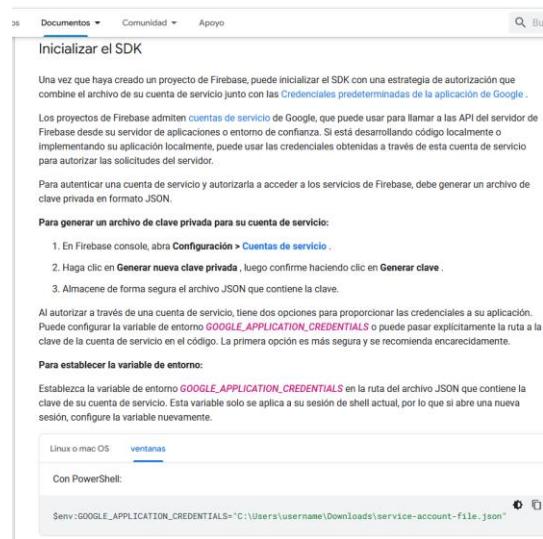


Fig. 4.1.79: Documentación Firebase, sección “Inizializar el SDK”, para las “Cuentas de servicio”.

Paso 15: Cuando estemos en “Cuentas de servicio” de nuestro proyecto actual, tendremos una pantalla como la de la Fig. 4.1.80.

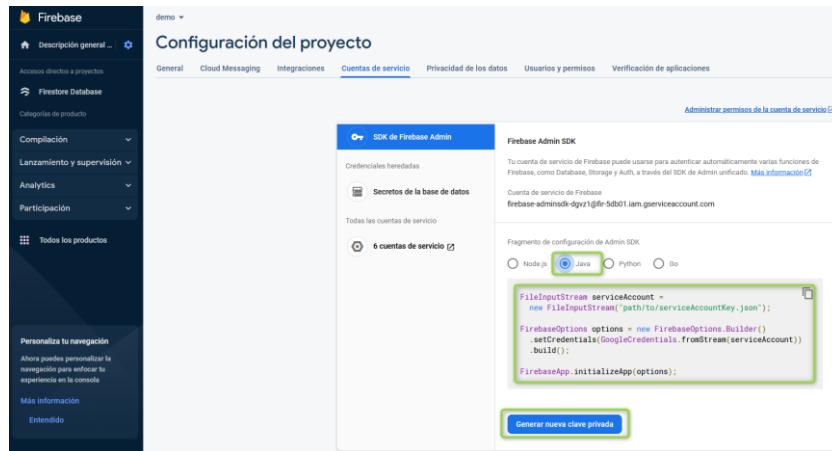


Fig. 4.1.80: Pantalla de “Cuentas de Servicio”, con código para vincular los servicios Java con Firebase.

Paso 16: Copiar el código señalado de la imagen anterior, y guardarlo temporalmente en un fichero de texto (para pegarlo posteriormente en el proyecto Backend).

Gracias a este fragmento de código nos podremos conectar a la BBDD Firebase desde nuestros servicios Java, creados en el backend.

Paso 17: Generar la clave privada del SDK de Firebase, necesaria para conectar Firebase con Java (desde los servicios Java backend).

Para ello, clica el botón “Generar nueva clase privada” (ver Fig. 4.1.80). Esto descargará un JSON, el cual será el archivo que funcione como clave privada.

Véanse las imágenes pre-descarga en Fig. 4.1.81 y post-descarga en Fig. 4.1.82.



Fig. 4.1.81: Advertencia previa a la descarga de la clave privada.

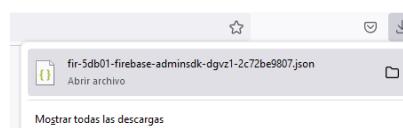


Fig. 4.1.82: Descarga ejecutada del JSON, clave privada de firebase.

Paso 18: Mover el fichero json descargado al interior de nuestro proyecto backend.

El fichero que es la clave privada se va a colocar en el directorio de “resources”. Véase Fig. 4.1.83.

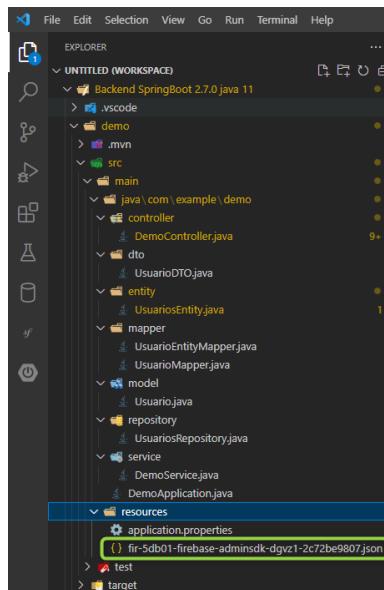


Fig. 4.1.83: Moviendo el fichero json descargado al interior del proyecto Backend.

Paso 19: Cambia el nombre del fichero json que acabas de poner en el directorio “resources”, por otro más amigable.

Propuesta de nombre alternativo: “private-key-firebase.json” (véase Fig. 4.1.84).

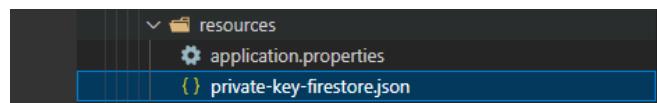


Fig. 4.1.84: Cambio de nombre del fichero json.

Paso 20: Crear un nuevo directorio en nuestro proyecto backend, llamado “firebase”.

En su interior crear la clase Java llamada “FirebaseInitializer.java”, y esta clase se encargará de conectarse a la base de datos de Firebase, bajo el producto “Cloud Firestore” (se conectará por nosotros).

Véase Fig. 4.1.85 para ver un ejemplo del nuevo directorio y la nueva clase.

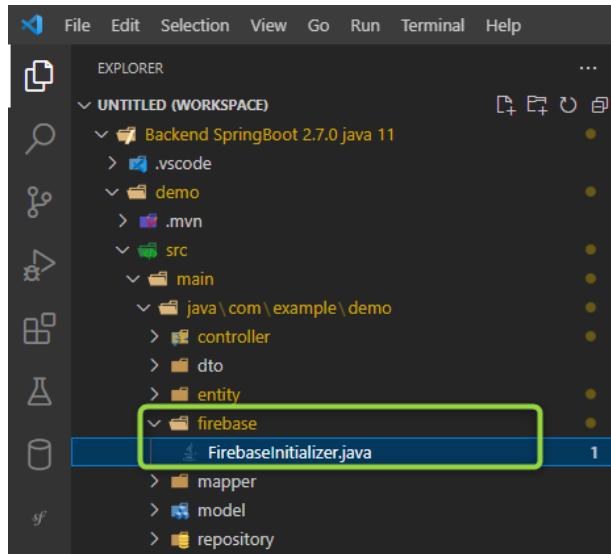


Fig. 4.1.85: Nuevo directorio y nueva clase Java creados en el proyecto backend.

Paso 21: Entrar en la nueva clase “FirebaseInitializer.java” y definirla con la etiqueta @Service.

La nueva clase contendrá 2 métodos:

- El primero será privado para inicializar Firebase.
- El segundo será público para obtener las instancias Firebase de la conexión realizada con el método anterior.

El código de esta nueva clase lo puede ver completo en la Fig. 4.1.86.

Nota 1: Entre las líneas 20 y 34 de la clase, el primer método que es privado, nos inicializa la base de datos Firebase. Contamos con la ayuda de la etiqueta @PostConstruct. El código de este primer método será el que, durante el paso 16, habíamos copiado (aquí con algunas modificaciones). Para destacar, 2 cosas:

- a) En línea 27 colocamos la URL de la base de datos siguiendo la estructura: [https://\[nombreDB\].firebaseio.com/](https://[nombreDB].firebaseio.com/), donde “[nombreDB]” será el nombre de la base de datos que puede encontrarse en la imagen de la base de datos recién creada.
- b) Inicialmente la línea 31 fue copia sin bucle IF. El bucle IF se añade para controlar que “si no se conecta correctamente a la BBDD, tampoco intente inicializar la conexión a la misma; en caso de conexión OK entonces if==true y se inicializa la conexión”.

Nota 2: Entre las líneas 36 y 38 creamos el método “getFirestore()”, el cual será público y devolverá una instancia de Firestore (el cliente de la conexión recién establecida).

```

1  FirebaseInitializer.java 1 ✘
2  Backend SpringBoot 2.7.0 Java 11 > demo > src > main > java > com > example > demo > firebase > FirebaseInitializer.java > FirebaseInitializer > initFirestore()
3
4  package com.example.demo.firebaseio;
5
6  import org.springframework.stereotype.Service;
7
8  import java.io.IOException;
9  import java.io.InputStream;
10 import javax.annotation.PostConstruct;
11
12 import com.google.auth.oauth2.GoogleCredentials;
13 import com.google.cloud.firestore.Firestore;
14 import com.google.firebase.FirebaseApp;
15 import com.google.firebase.FirebaseOptions;
16 import com.google.firebase.cloud.FirestoreClient;
17
18 @Service
19 public class FirebaseInitializer {
20
21     @PostConstruct
22     private void initFirestore() throws IOException{
23
24         InputStream serviceAccount = getClass().getClassLoader().getResourceAsStream("private-key-firebase.json");
25
26         FirebaseOptions options = new FirebaseOptions.Builder()
27             .setCredentials(GoogleCredentials.fromStream(serviceAccount))
28             .setDatabaseUrl("https://fir-sdb01.firebaseio.com/")
29             .build();
30
31         if(FirebaseApp.getApps().isEmpty()){
32             FirebaseApp.initializeApp(options);
33         }
34     }
35
36     public Firestore getFirestore(){
37         return FirestoreClient.getFirestore();
38     }
39 }

```

Fig. 4.1.86: Código completo de la nueva clase “FirebaseInitializer.java”.

Paso 22: Revisar el código del fichero “pom.xml” tras añadir la dependencia que conecta con la BBDD “Firebase”. Véase Fig. .4.1.87.

Notar que se añadió únicamente la dependencia de la BBDD “Firebase”, y se mantiene la dependencia de BBDD tipo H2 sin inconvenientes.

```

1  pom.xml 1 ✘
2  Backend SpringBoot 2.7.0 Java 11 > demo > pom.xml > project
3
4      <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
5      <!-- Usada para poder trabajar con ORMs como JPA -->
6      <dependency>
7          <groupId>org.springframework.boot</groupId>
8          <artifactId>spring-boot-starter-data-jpa</artifactId>
9          <version>2.7.0</version>
10         </dependency>
11
12         <!-- librería para corregir el error de "Failed to Refresh live data from process" -->
13         <dependency>
14             <groupId>org.springframework.boot</groupId>
15             <artifactId>spring-boot-starter-actuator</artifactId>
16         </dependency>
17
18         <!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
19         <!-- Usada para cargar el gestor de Base de Datos volátil H2 -->
20         <dependency>
21             <groupId>com.h2database</groupId>
22             <artifactId>h2</artifactId>
23             <scope>runtime</scope>
24         </dependency>
25
26         <!-- librería para conectar con 8080 google Firebase - producto Cloud Firestore -->
27         <dependency>
28             <groupId>com.google.firebaseio</groupId>
29             <artifactId>firebase-admin</artifactId>
30             <version>9.1.1</version>
31         </dependency>
32
33     </dependencies>
34
35     <build>
36         <plugins>
37             <plugin>
38                 <groupId>org.springframework.boot</groupId>
39                 <artifactId>spring-boot-maven-plugin</artifactId>
40             </plugin>
41         </plugins>
42     </build>
43 </project>

```

Fig. 4.1.87: Añadida dependencia de base de datos Firebase de Google.

Paso 23: Ahora implementar los nuevos servicios que conecten con Firebase. Para ello, crear otra clase y su interfaz en el directorio ya existente, llamado “service”.

La nueva interface se llamará “FirebaseDemoService.java”, y la nueva clase que la implementa se llamará “FirebaseDemoServiceImpl.java”.

Puede verse más detalles de la arquitectura en la imagen “Fig. 4.1.88”.

¿Qué contendrá esta nueva clase impl y su interface?

- Esencialmente tendrá los mismos servicios que la antigua clase “DemoService.java”, pero en lugar de conectarse a una “H2” se conectará a “Firebase”.

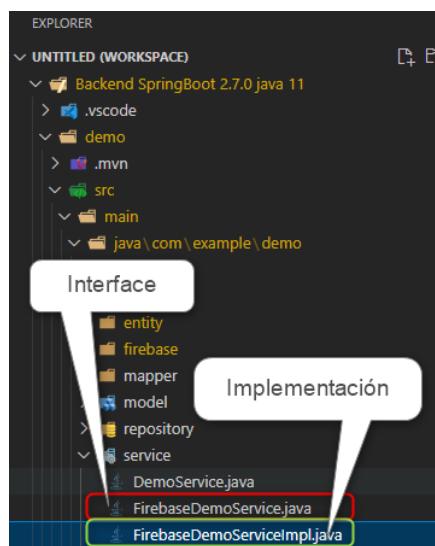


Fig. 4.1.88: Nueva interface y su clase impl, que desarrollan los nuevos servicios que conectan con Firebase.

Paso 24: Crear la nueva tabla de datos Cloud para los “usuarios”. Para ello, en jerga de Google necesitaremos crear una nueva colección (donde almacenaremos la lista de los usuarios existentes).

La nueva colección se llamará “usuarios-coleccion”.

Puede ver cómo crear una nueva colección viendo desde Fig.4.1.89 hasta Fig. 4.1.93.

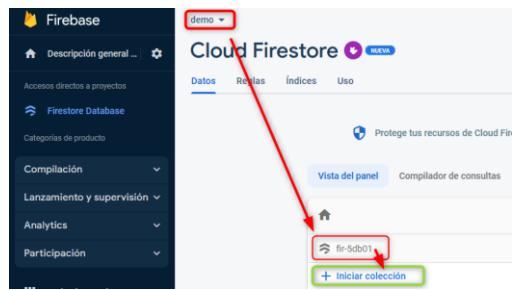


Fig. 4.1.89: Panel Cloud Firestore desde donde crear una nueva colección de datos.

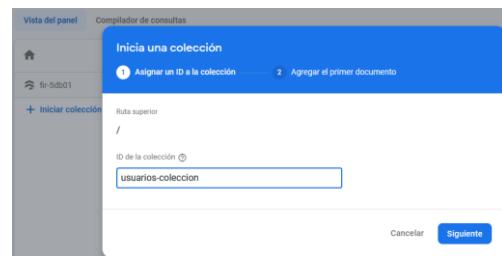


Fig. 4.1.90: Nombrando la nueva colección de datos que está creandose.

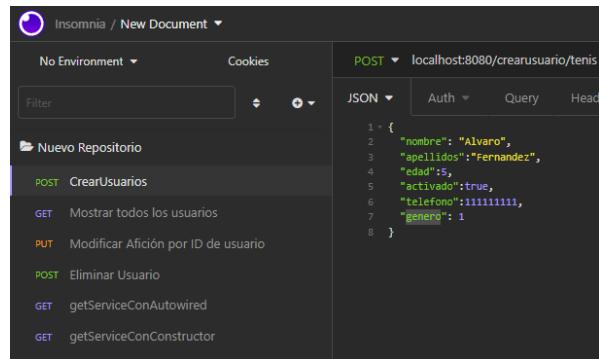


Fig. 4.1.91: Revisar la consulta HTTP “CrearUsuarios” y copiar su JSON.

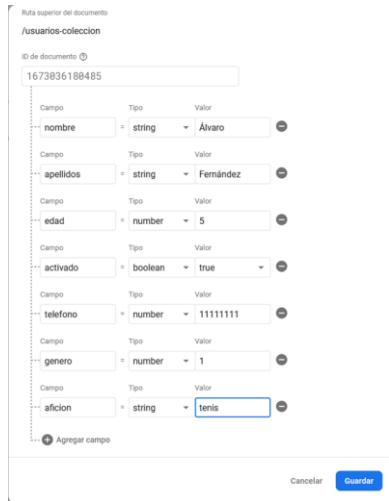


Fig. 4.1.92: Creación del primer elemento de la colección, siguiente estructura del JSON de usuarios. El id de este elemento será manualmente introducido, tipo Long/numeric y corresponde al DateTime del instante en que se crea el usuario.

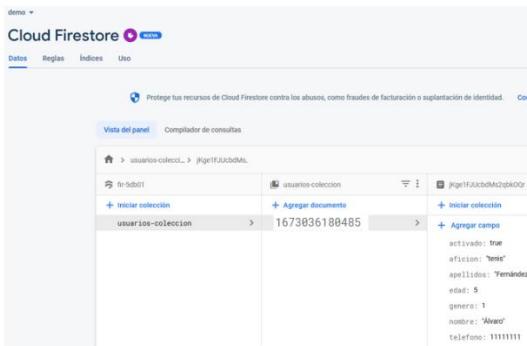


Fig. 4.1.93: Primer elemento creado en nuestra nueva colección de Firebase.

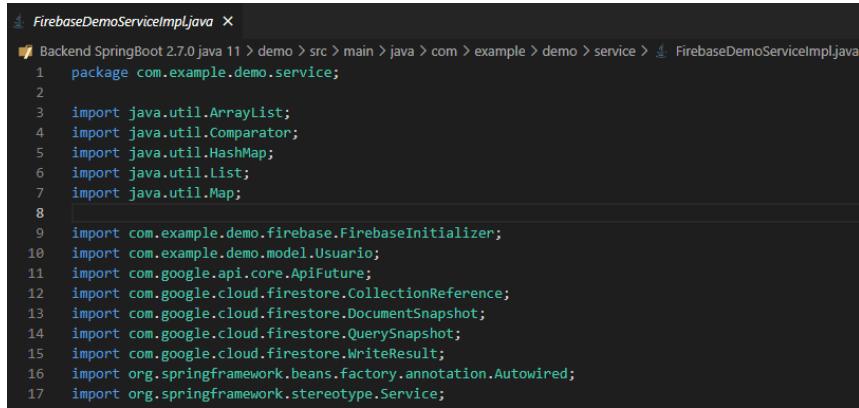
Paso 25: Crear las cabeceras de los nuevos servicios en la nueva interface “FirebaseDemoService.java”. Véase el código completo en la “Fig. 4.1.94”.

```
 FirebaseDemoService.java X
Backend SpringBoot 2.7.0 Java 11 > demo > src > main > java > com > example > demo > service > FirebaseDemoService.java
1 package com.example.demo.service;
2
3 import java.util.List;
4
5 import com.example.demo.model.Usuario;
6
7 public interface FirebaseDemoService {
8
9     List<Usuario> listarUsuariosFirebase();
10
11     Long crearUsuarioFirebase(Usuario usuario);
12
13     Boolean modificarUsuarioFirebase(String id, Usuario usuario);
14
15     Boolean eliminarUsuarioPorIdFirebase(String id);
16
17     List<Usuario> ordenarListaUsuarios (List<Usuario> usuarios, String ordenadosPor);
18
19 }
```

Fig. 4.1.94: Código completo de la nueva interfaz, para los nuevos servicios.

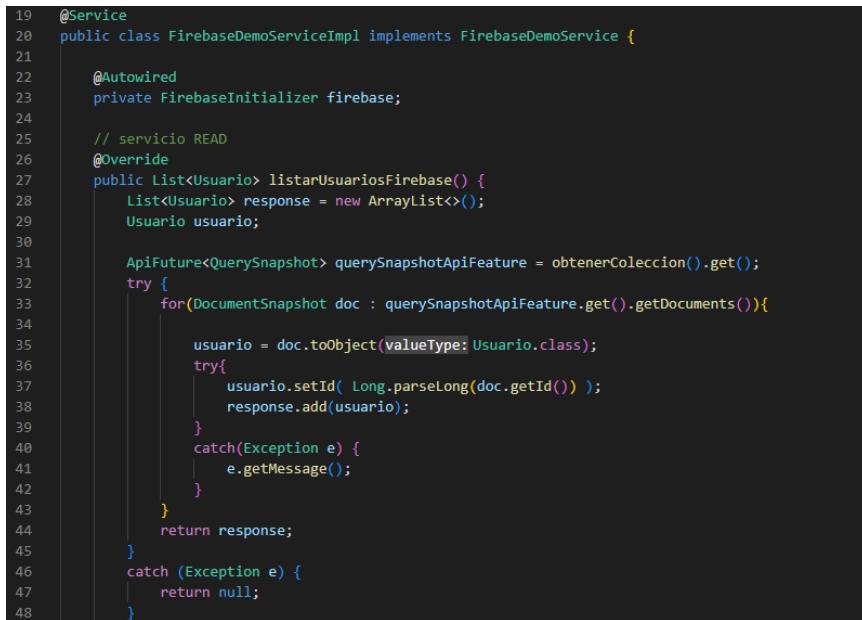
Paso 26: Crear la implementación completa de los nuevos servicios, en la clase “FirebaseDemoServiceImpl.java”.

Véase figuras desde la 4.1.95 hasta la 4.1.99.



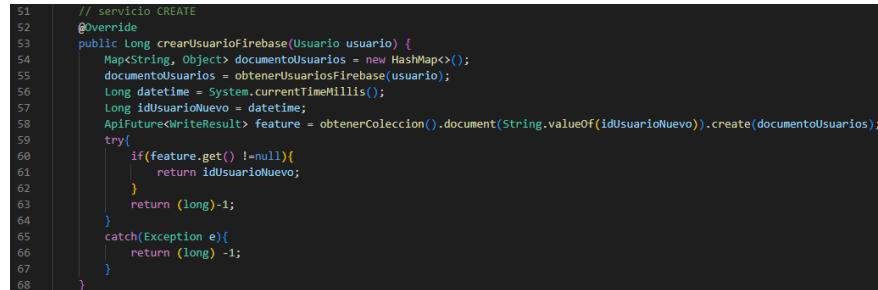
```
Backend SpringBoot 2.7.0 java 11 > demo > src > main > java > com > example > demo > service > FirebaseDemoServiceImpl.java
1 package com.example.demo.service;
2
3 import java.util.ArrayList;
4 import java.util.Comparator;
5 import java.util.HashMap;
6 import java.util.List;
7 import java.util.Map;
8
9 import com.example.demo.firebaseio.FirebaseInitializer;
10 import com.example.demo.model.Usuario;
11 import com.google.api.core.ApiFuture;
12 import com.google.cloud.firestore.CollectionReference;
13 import com.google.cloud.firestore.DocumentSnapshot;
14 import com.google.cloud.firestore.QuerySnapshot;
15 import com.google.cloud.firestore.WriteResult;
16 import org.springframework.beans.factory.annotation.Autowired;
17 import org.springframework.stereotype.Service;
```

Fig. 4.1.95: Imports de la implementación de los nuevos servicios.



```
19 @Service
20 public class FirebaseDemoServiceImpl implements FirebaseDemoService {
21
22     @Autowired
23     private FirebaseInitializer firebase;
24
25     // servicio READ
26     @Override
27     public List<Usuario> listarUsuariosFirebase() {
28         List<Usuario> response = new ArrayList<>();
29         Usuario usuario;
30
31         ApiFuture<QuerySnapshot> querySnapshotApiFeature = obtenerColeccion().get();
32         try {
33             for(DocumentSnapshot doc : querySnapshotApiFeature.get().getDocuments()){
34
35                 usuario = doc.toObject(valueType: Usuario.class);
36                 try{
37                     usuario.setId( Long.parseLong(doc.getId()) );
38                     response.add(usuario);
39                 }
39                 catch(Exception e) {
40                     e.getMessage();
41                 }
42             }
43             return response;
44         }
45         catch (Exception e) {
46             return null;
47         }
48     }
```

Fig. 4.1.96: Cabecera de clase y método READ de la implementación de los nuevos servicios.



```
51     // servicio CREATE
52     @Override
53     public Long crearUsuarioFirebase(Usuario usuario) {
54         Map<String, Object> documentoUsuarios = new HashMap<>();
55         documentoUsuarios = obtenerUsuariosFirebase(usuario);
56         Long datetime = System.currentTimeMillis();
57         Long idUsuarioNuevo = datetime;
58         ApiFuture<WriteResult> feature = obtenerColección().document(String.valueOf(idUsuarioNuevo)).create(documentoUsuarios);
59         try{
60             if(feature.get() != null){
61                 return idUsuarioNuevo;
62             }
63             return (long)-1;
64         }
65         catch(Exception e){
66             return (long) -1;
67         }
68     }
```

Fig. 4.1.97: CREATE de la implementación de los nuevos servicios.

```

70     // servicio UPDATE
71     @Override
72     public Boolean modificarUsuarioFirebase(String id, Usuario usuario) {
73         Map<String, Object> documentoUsuarios = new HashMap<>();
74         documentoUsuarios = obtenerUsuariosFirebase(usuario);
75         ApiFuture<WriteResult> feature = obtenerColeccion().document(id).set(documentoUsuarios);
76         try{
77             if(feature.get() !=null){
78                 return Boolean.TRUE;
79             }
80             return Boolean.FALSE;
81         }
82         catch(Exception e){
83             return Boolean.FALSE;
84         }
85     }
86
87     // servicio DELETE
88     @Override
89     public Boolean eliminarUsuarioPorIdFirebase(String id) {
90         ApiFuture<WriteResult> feature = obtenerColeccion().document(id).delete();
91         try{
92             if(feature.get() !=null){
93                 return Boolean.TRUE;
94             }
95             return null;
96         }
97         catch(Exception e){
98             return null;
99         }
100    }

```

Fig. 4.1.98: UPDATE y DELETE de la implementación de los nuevos servicios.

```

102    // metodo que ordena los Usuarios mostrados en operaciones GET HTTP, segun id, nombre o edad
103    @Override
104    public List<Usuario> ordenarListaUsuarios (List<Usuario> usuarios, String ordenadosPor){
105        Boolean ordenado = false;
106        if(ordenadosPor.equals(anObject: "id")){
107            usuarios.sort(Comparator.comparing(Usuario::getId));
108            ordenado = true;
109        }
110        if(ordenadosPor.equals(anObject: "nombre")){
111            usuarios.sort(Comparator.comparing(Usuario::getNombre));
112            ordenado = true;
113        }
114        if(ordenadosPor.equals(anObject: "edad")){
115            usuarios.sort(Comparator.comparing(Usuario::getEdad));
116            ordenado = true;
117        }
118        if(!ordenado){
119            usuarios.sort(Comparator.comparing(Usuario::getId));
120        }
121        return usuarios;
122    }
123
124    // metodo interno 1: localiza referencia de mi collection Firebase
125    private CollectionReference obtenerColeccion (){
126        return firebase.getFirestore().collection(path: "usuarios-coleccion");
127    }
128
129    // metodo interno 2: rellena el hashMap a partir del Usuario entrante y devuelve el hashMap completado
130    private Map<String, Object> obtenerUsuariosFirebase(Usuario usuario){
131        Map<String, Object> documentoUsuariosMap = new HashMap<>();
132        documentoUsuariosMap.put(key: "nombre", usuario.getNombre());
133        documentoUsuariosMap.put(key: "apellidos", usuario.getApellidos());
134        documentoUsuariosMap.put(key: "edad", usuario.getEdad());
135        documentoUsuariosMap.put(key: "activado", usuario.getActivado());
136        documentoUsuariosMap.put(key: "telefono", usuario.getTelefono());
137        documentoUsuariosMap.put(key: "genero", usuario.getGenero());
138        documentoUsuariosMap.put(key: "aficion", usuario.getAficion());
139    }
140
141 }

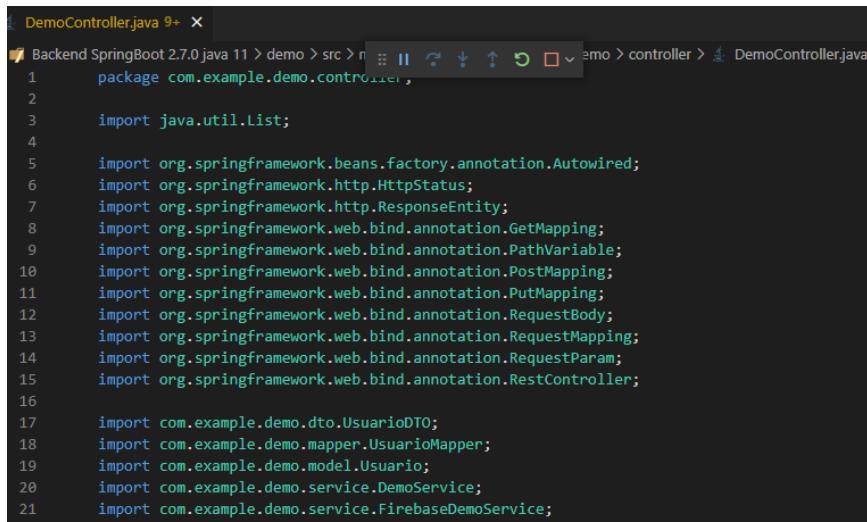
```

Fig. 4.1.99: Método de ordenación de lista de Usuarios y método para obtener usuarios de Firebase, de la implementación de los nuevos servicios. Fin de la implementación.

Paso 27: Modificación del fichero del Controller. Conectaremos así los EndPoint's definidos en el Controller, a los nuevos servicios (que usan Firebase).

Véanse las siguientes imágenes para conocer los cambios realizados en el controller (desde Fig. 4.1.100 hasta Fig. 4.1.103).

Esencialmente se añaden las líneas 31 y 32, y dentro de cada método se modifica la llamada al servicio (usando el objeto de Firebase) y dejamos comentada la llamada a los servicios anteriores (usados para H2).

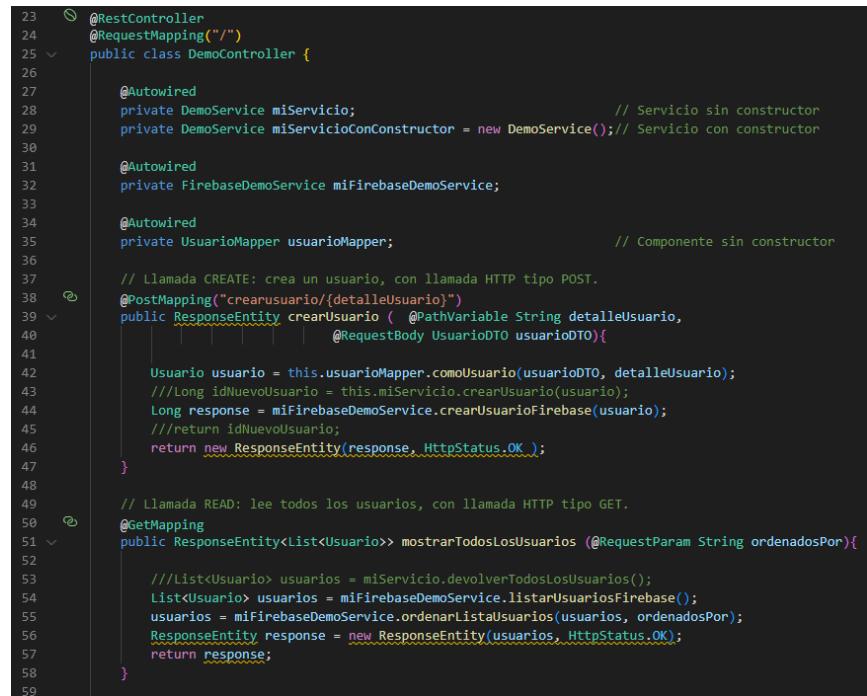


```

1 package com.example.demo.controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.PathVariable;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.PutMapping;
12 import org.springframework.web.bind.annotation.RequestBody;
13 import org.springframework.web.bind.annotation.RequestMapping;
14 import org.springframework.web.bind.annotation.RequestParam;
15 import org.springframework.web.bind.annotation.RestController;
16
17 import com.example.demo.dto.UsuarioDTO;
18 import com.example.demo.mapper.UsuarioMapper;
19 import com.example.demo.model.Usuario;
20 import com.example.demo.service.DemoService;
21 import com.example.demo.service.FirebaseDemoService;

```

Fig. 4.1.100: Imports, del controller.



```

23
24     @RestController
25     @RequestMapping("/")
26     public class DemoController {
27
28         @Autowired
29         private DemoService miService; // Servicio sin constructor
30         private DemoService miServicioConConstructor = new DemoService(); // Servicio con constructor
31
32         @Autowired
33         private FirebaseDemoService miFirebaseDemoService;
34
35         @Autowired
36         private UsuarioMapper usuarioMapper; // Componente sin constructor
37
38         // Llamada CREATE: crea un usuario, con llamada HTTP tipo POST.
39         @PostMapping("crearusuario/{detalleUsuario}")
40         public ResponseEntity<Usuario> crearUsuario (@PathVariable String detalleUsuario,
41                                         @RequestBody UsuarioDTO usuarioDTO){
42
43             Usuario usuario = this.usuarioMapper.comoUsuario(usuarioDTO, detalleUsuario);
44             //Long idNuevoUsuario = this.miService.crearUsuario(usuario);
45             Long response = miFirebaseDemoService.crearUsuarioFirebase(usuario);
46             //return idNuevoUsuario;
47             return new ResponseEntity(response, HttpStatus.OK);
48         }
49
50         // Llamada READ: lee todos los usuarios, con llamada HTTP tipo GET.
51         @GetMapping
52         public ResponseEntity<List<Usuario>> mostrarTodosLosUsuarios (@RequestParam String ordenadosPor){
53
54             //List<Usuario> usuarios = miService.devolverTodosLosUsuarios();
55             List<Usuario> usuarios = miFirebaseDemoService.listarUsuariosFirebase();
56             usuarios = miFirebaseDemoService.ordenarListaUsuarios(usuarios, ordenadosPor);
57             ResponseEntity<List<Usuario>> response = new ResponseEntity(usuarios, HttpStatus.OK);
58             return response;
59         }

```

Fig. 4.1.101: Cabecera de clase, variables de clase y métodos CREATE y READ, del controller.

```

50
51     // Llamada UPDATE: modifica un usuario con llamada HTTP tipo PUT.
52     @PutMapping("/modificarDetalleUsuario/{idUsuario}/{nuevoDetalle}")
53     public ResponseEntity<Void> modificarDetalleUsuario (@PathVariable String idUsuario,
54                                         @PathVariable String nuevoDetalle,
55                                         @RequestBody UsuarioDTO usuarioDTO){
56         //Usuario usuario = miServicio.modificarUsuario(idUsuario, nuevoDetalle);
57         Usuario usuario = this.usuarioMapper.comoUsuario(usuarioDTO, nuevoDetalle);
58         Boolean b = miFirebaseDemoService.modificarUsuarioFirebase(idUsuario, usuario);
59         if( b == true){
60             return ResponseEntity.noContent().build();
61         }
62         else{
63             ResponseEntity responseEntity = new ResponseEntity("Imposible actualizar, el elemento no existe.", HttpStatus.BAD_REQUEST);
64             return responseEntity;
65         }
66     }
67
68     // Llamada DELETE: elimina un usuario con llamada HTTP tipo POST.
69     @PostMapping("/eliminarUsuario/{id}")
70     public ResponseEntity<Void> eliminarUsuario (@PathVariable String id){
71
72         //Long idUserEliminado = miServicio.eliminarUsuarioPorId(id);
73         Boolean b = miFirebaseDemoService.eliminarUsuarioPorIdFirebase(id);
74         if (b != null){
75             return ResponseEntity.noContent().build();
76         }
77
78         return ResponseEntity.notFound().build();
79     }
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108

```

Fig. 4.1.102: Métodos UPDATE y DELETE, del controller.

```

90
91
92
93
94
95
96
97
98     @GetMapping("/getServiceConConstructor")
99     public String getServiceConConstructor (){
100         return this.miServicioConConstructor.consiguePrefijoSufijoDelConstructor();
101     }
102
103     @GetMapping("/getServiceConAutowired")
104     public String getServiceConAutowired (){
105         return this.miServicio.consiguePrefijoSufijoDeApplicationProperties();
106     }
107
108

```

Fig. 4.1.103: Código final (irrelevante para los cambios de este apartado), del controller.

Paso 28: Modificación en las consultas REST (HTTP) con InsomniaCore:

- Se mantiene sin cambios las consultas REST realizadas desde InsomniaCore, salvo en el caso de consultas tipo “Update” o actualizar.
- En caso de la consulta de tipo Update, crear una 2^a consulta PUT que contenga JSON en el Body (y dejar de usar la consulta “actualizar” anterior).
- La lista final de consultas totales, tras la modificación, quedaría como se puede ver en la Fig. 4.1.104:

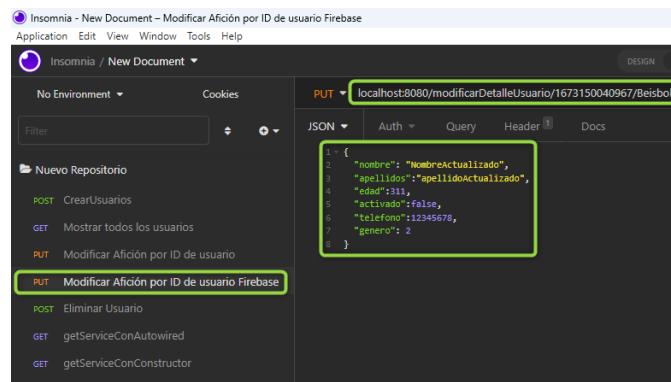


Fig. 4.1.104: Lista de consultas REST final, tras añadir la nueva consulta de tipo PUT- update.

4.2 Tutorial Front-end (Angular)

Angular trabaja completamente con “TypeScript”, además de usar HTML, CSS y XML.

Por ello, para explicar correctamente “cómo programar en Angular”, primero van a explicarse conceptos básicos de la programación con TypeScript. Posteriormente irán varios tutoriales a cerca de funcionalidades en Angular.

a) Tutorial Básico de TypeScript (con ejercicios):

TypeScript es un lenguaje de programación, que podría traducirse como “JavaScript con Tipos”: todo el código que se desee desarrollar en Javascript será escrito en typescript, con sintaxis de tipos, y después “transpilado” a lenguaje JavaScript automáticamente.

Pero, ¿qué es transpilar? Es un concepto similar al de “compilar” software:

- Compilar código: proceso de traducir código de programación de un lenguaje A hacia otro lenguaje B, siempre que <A> y sean lenguajes de diferente nivel de abstracción. Ejemplo: compilar código de lenguaje Java a lenguaje Máquina.
- Transpilar código: proceso de traducir código de programación de un lenguaje A hacia otro lenguaje B, siempre que <A> y sean lenguajes de mismo nivel de abstracción. Ejemplo: compilar código de lenguaje TypeScript a lenguaje JavaScript.
-

Ejercicio 1:

Usando la página web oficial de <https://www.typescriptlang.org/> seccione la pestaña “PlayGround” para que veamos cómo transforma un código TypeScript a su equivalencia de código JavaScript.

Dato extra: Este ejercicio no requiere instalar Node Js porque esta ejecutando código online, pero no olvide instalar “Node js” y “TypeScript” en su equipo local para el resto de ejercicios (revisar los pasos 2, 3, 4, 5, 10, 11 y 12 del apartado 3.6 de este tutorial).

Paso 1: Ir a la página oficial de TypeScript, sección PlayGround, y se crea el siguiente código ejemplo en lenguaje TypeScript, el cual genera el código equivalente en JavaScript de su derecha (ver Fig. 4.2.1).

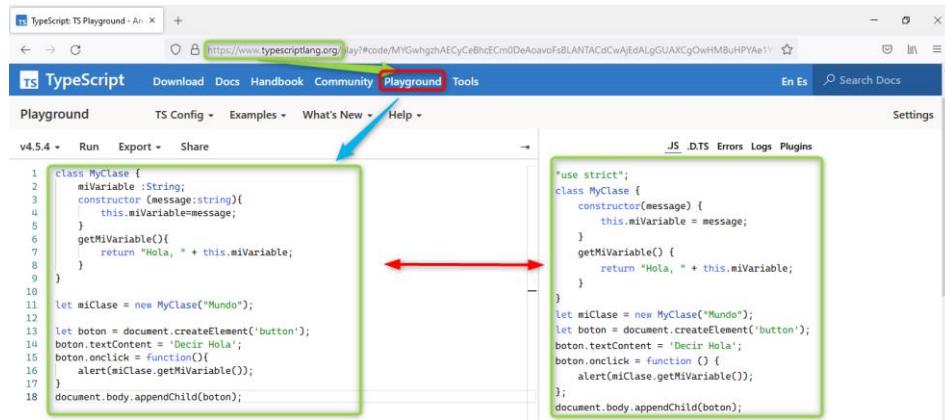


Fig. 4.2.1: Pantalla del playground oficial para practicar Typescript.

Ejercicio 2:

Crear nuestro primer “console.log(‘Hola mundo’)” web, a través de un programa TypeScript.

Paso 1: Abrir nuestro IDE “Visual Studio Code”, y traer a nuestra Área de Trabajo una carpeta que se ha llamado “TypeScript”.

Paso 2: En Visual Studio Code, entrar en la carpeta “TypeScript” y crear 2 archivos nuevos, desde el botón derecho del mouse sobre la carpeta indicada:

- app1.ts (ver Fig. 4.2.2)
- index.html (ver Fig. 4.2.2)

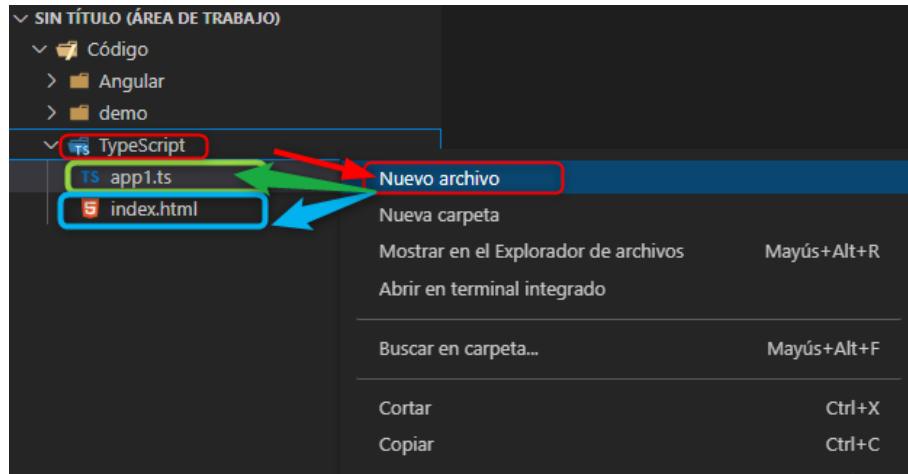


Fig. 4.2.2: Creación de ficheros extensión TS y HTML manualmente.

Paso 3: Una vez se han creado los 2 archivos indicados en la imagen anterior, de forma manual, se llena primeramente el archivo “index.html” y se completa la estructura básica automáticamente.

Para completar el archivo “index.html”, empezar a escribir “html:” y seleccionar la opción de “html:5” como se indica en la Fig. 4.2.3.

Véase el resultado final del código completado en la Fig. 4.2.4.

- Auto-completado HTML 5:

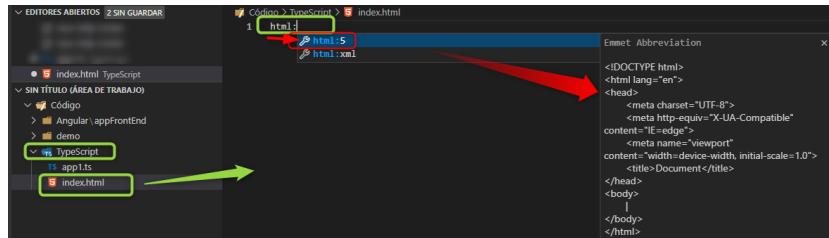


Fig. 4.2.3: Fichero “index.html”, en proceso de autocompletado de la plantilla HTML.

- Resultado del Auto-completado HTML 5:

```

Código > TypeScript > index.html > html > head > meta
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10 
11 </body>
12 </html>

```

Fig. 4.2.4: Fichero “index.html”, con el resultado autocompletado plantilla HTML.

Paso 4: En el nuevo HTML, hacer la llamada al fichero .js que usaré para cargar el código Javascript a mi HTML.

El fichero JavaScript que se llamará desde el HTML tal vez aún no exista, pero debe llamarse de la misma forma que el fichero Typescript creado anteriormente (si existe “app1.ts”, nosotros llamaremos desde el HTML al “app1.js”).

- Añadimos un script vinculado a este nuevo HTML (Fig. 4.2.5):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="app1.js"></script>
</body>
</html>
```

Fig. 4.2.5: Incrustación llamada a fichero javascript.

Paso 5: creo el código del fichero “app1.ts” para que genere un mensaje por consola (Fig. 4.2.6):

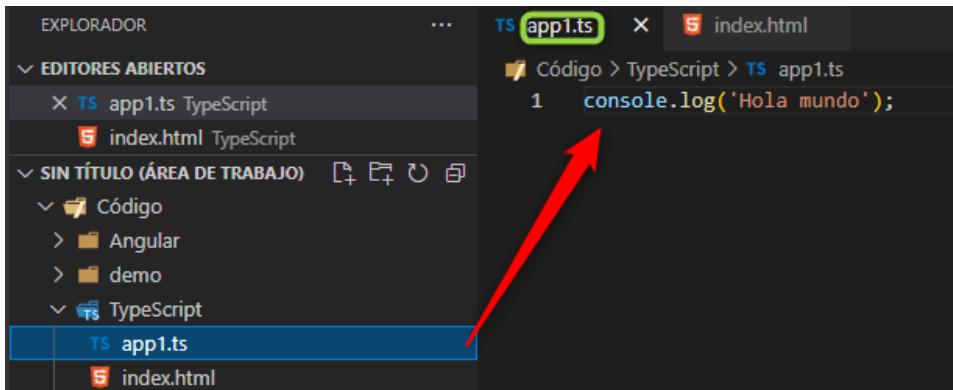


Fig. 4.2.6: Ejemplo Hello World en Typescript.

Paso 6: ejecuto el archivo “app1.ts” desde el terminal, colocándome en el interior del directorio del “TypeScript”.

El resultado de la ejecución será la creación automática de un archivo JavaScript, homólogo al de extensión TS que acabamos de ejecutar (ver Fig. 4.2.7).

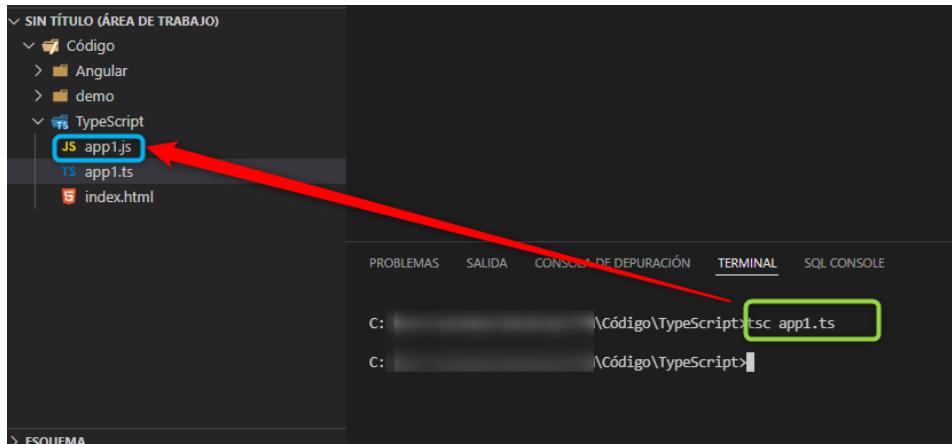


Fig. 4.2.7: La ejecución del fichero Typescript genera un archivo Javascript.

Paso 7: Si abrimos el nuevo archivo JS generado de la ejecución del “app1.ts”, en este caso podremos comprobar que será exactamente igual a nuestro código del archivo “app1.ts” (véase Fig. 4.2.8).

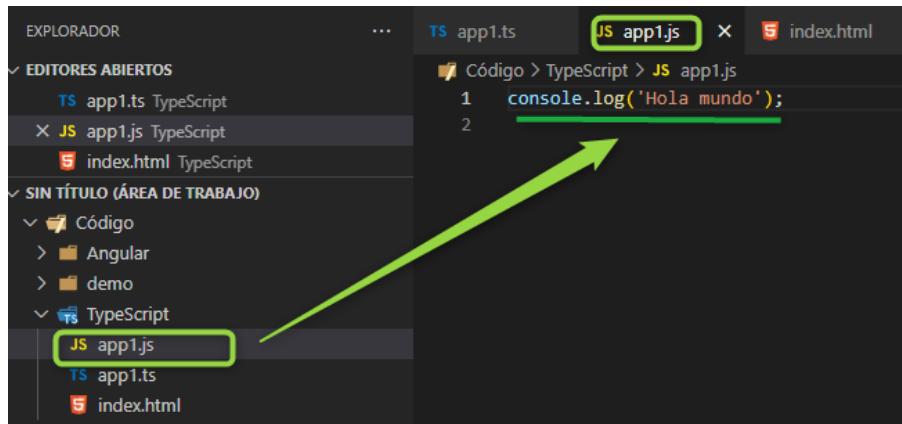


Fig. 4.2.8: Visualización contenido del javascript auto-generado.

Paso 8: Si cada vez que guardes dicho archivo con extensión TS deseas que se realice la transpilación de forma automática, desde un archivo TS a otro JS, ejecuta el siguiente comando en la terminal (véase Fig. 4.2.9).

Se creará un proceso “observador” que refrescará el archivo JS de forma automática cada vez que se guarde el TS.

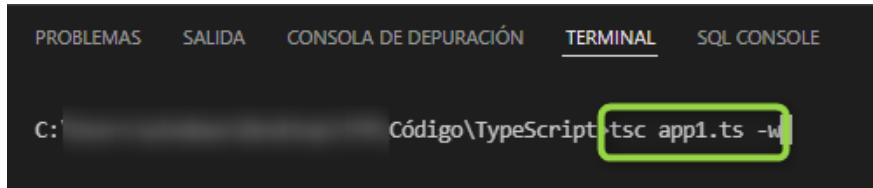
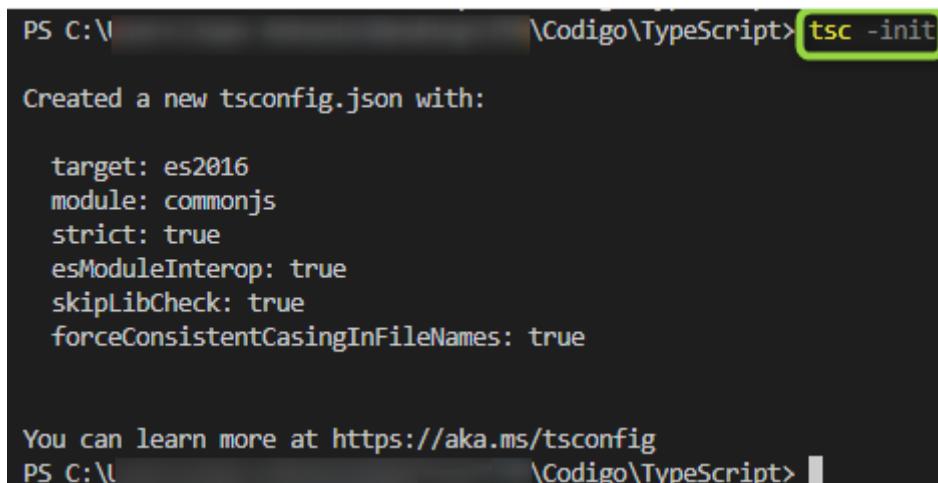


Fig. 4.2.9: Ejecución del comando que mantiene la escucha activa sobre un archivo typescript.

Ejercicio 3:

Crear un proyecto TypeScript completo (en la misma carpeta que hemos usando para los ejercicios anteriores). Después estudie la estructura básica del nuevo proyecto software.

Paso 1: Si deseas crear un proyecto en Typescript completo, ejecuta el siguiente comando desde un terminal de comandos Windows y posicionado en el directorio que tienes cargado en Área de Trabajo de Visual Studio Code: “tsc - init” (ver Fig. 4.2.10).



```
PS C:\          \Codigo\TypeScript> tsc -init
Created a new tsconfig.json with:

target: es2016
module: commonjs
strict: true
esModuleInterop: true
skipLibCheck: true
forceConsistentCasingInFileNames: true

You can learn more at https://aka.ms/tsconfig
PS C:\          \Codigo\TypeScript>
```

Fig. 4.2.10: Creación de nuevo proyecto Typescript por consola de comandos.

Paso 2: Observar la estructura básica del nuevo proyecto creado de Typescript.

- Como pudimos ver en la fig. 4.2.10 el nuevo proyecto Typescript se crea dentro del directorio “.../Codigo/Typescript”.
- Anteriormente ya teníamos creados los archivos “app1.ts” y “app1.js”.
- Con la creación del nuevo proyecto Typescript se crearon 2 nuevos ficheros (ver Fig. 4.2.11):
 - “index.html”
 - “tsconfig.json”

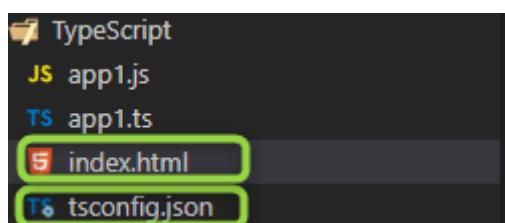


Fig. 4.2.11: Estructura del proyecto Typescript (nuevos archivos en verde).

Ejercicio 4:

Estudiar el contenido del archivo de configuración del nuevo proyecto TypeScript (“tsconfig.json”).

Paso 1: Visualizar el contenido del nuevo JSON (tsconfig.json), que contiene las configuraciones del proyecto (véase Fig. 4.2.12):

```
tsconfig.json
{
  "compilerOptions": {
    /* Projects */
    // "incremental": true, /* Enable incremental compilation */
    // "composite": true, /* Enable constraints that allow a TypeScript project to reference other projects */
    // "tsBuildInfoFile": "./", /* Specify the folder for .tsbuildinfo incremental */
    // "disableSourceFileProjectReferenceRedirect": true, /* Disable preferring source files instead of declarations for a project */
    // "disableSolutionSearching": true, /* Opt a project out of multi-project reference checks */
    // "disableReferencedProjectLoad": true, /* Reduce the number of projects loaded automatically */

    /* Language and Environment */
    // "target": "es2016", /* Set the JavaScript language version for emitted JS */
    // "lib": [], /* Specify a set of bundled library declaration files */
    // "jsx": "preserve", /* Specify what JSX code is generated */
    // "experimentalDecorators": true, /* Enable experimental support for TC39 stage 2 decorators */
    // "emitDecoratorMetadata": true, /* Emit design-type metadata for decorated declarations */
    // "jsxFactory": "", /* Specify the JSX factory function used when targeting ES5 */
    // "jsxFragmentFactory": "", /* Specify the JSX fragment reference used for fragments */
    // "jsxImportSource": "", /* Specify module specifier used to import the JSX package */
    // "reactNamespace": "", /* Specify the object invoked for 'createElement' */
    // "noEmit": true, /* Disable emitting any library files, including type definitions */
    // "useDefineForClassFields": true, /* Emit ECMAScript-standard-compliant class fields */

    /* Modules */
    "module": "commonjs", /* Specify what module code is generated */
    // "rootDir": "./", /* Specify the root folder within your source files */
    // "baseUrl": "./", /* Specify how TypeScript looks up a file from a relative path */
    // "paths": {}, /* Specify a set of entries that re-map imports to different locations */
    // "rootDirs": [], /* Allow multiple folders to be treated as one when resolving imports */
    // "typeRoots": [], /* Specify multiple folders that act like ./node_modules/@types */
    // "types": [], /* Specify type package names to be included with --types */
    // "allowNonGlobalAccess": true, /* Allow accessing UMD globals from modules */
    // "resolveJsonModule": true, /* Enable importing .json files */
    // "noResolve": true, /* Disallow import's, require's or <reference> */

    /* JavaScript Support */
    // "allowJs": true, /* Allow JavaScript files to be a part of your project */
    // "checkJs": true, /* Enable error reporting in type-checked JavaScript files */
    // "maxNodeModulesDepth": 1, /* Specify the maximum folder depth used for checking */

    /* Emissions */
    // "declaration": true, /* Generate .d.ts files from TypeScript and JavaScript code */
    // "declarationMap": true, /* Create sourcemaps for d.ts files */
    // "emitDeclarationOnly": true, /* Only output d.ts files and not JavaScript files */
    // "sourceMap": true, /* Create source map files for emitted JavaScript code */
    // "outFile": "./", /* Specify a file that bundles all outputs into a single file */
    // "outDir": "./", /* Specify an output folder for all emitted files */
    // "removeComments": true, /* Disable emitting comments */
    // "noEmit": true, /* Disable emitting files from a compilation */
    // "importHelpers": true, /* Allow importing helper functions from tslib or DefinitelyTyped */
    // "importNotUsedAsValues": "remove", /* Specify emit/checked behavior for imports that are not used */
    // "downlevelIteration": true, /* Emit more compliant, but verbose and less performant JavaScript */
    // "sourceRoot": "", /* Specify the root path for debuggers to find the source code */
    // "mapRoot": "", /* Specify the location where debugger should locate the map files */
    // "inlineSourceMap": true, /* Include sourcemap files inside the emitted JavaScript code */
    // "inlineSources": true, /* Include source code in the sourcemaps inside the emitted JavaScript code */
    // "emitBOM": true, /* Emit a UTF-8 Byte Order Mark (BOM) in the beginning of each file */
    // "newline": "\r\n", /* Set the newline character for emitting files */
    // "stripInternal": true, /* Disable emitting declarations that have '_internal' */
    // "noEmitHelpers": true, /* Disable generating custom helper functions */
    // "noEmitOnError": true, /* Disable emitting files if any type checking errors occur */
    // "preserveConstEnums": true, /* Disable erasing 'const enum' declarations */
    // "declarationDir": "./", /* Specify the output directory for generated declarations */
    // "preserveValueImports": true, /* Preserve unused imported values in the JavaScript code */

    /* Interop Constraints */
    // "isolatedModules": true, /* Ensure that each file can be safely transpiled */
    // "allowSyntheticDefaultImports": true, /* Allow 'import x' from 'y' when a module doesn't have an export default */
    // "esModuleInterop": true, /* Emit additional JavaScript to ease support for ES6+ features */
    // "preserveSymlinks": true, /* Disable resolving symlinks to their realpath */
    // "forceConsistentCasingInFileNames": true, /* Ensure that casing is correct in imports */

    /* Type Checking */
    "strict": true, /* Enable all strict type-checking options */
    // "noImplicitAny": true, /* Enable error reporting for expressions and declarations with an implied any type */
    // "strictNullChecks": true, /* When type checking, take into account 'null' and 'undefined' */
    // "strictFunctionTypes": true, /* When assigning functions, check to ensure parameters and return types are compatible */
    // "strictBindCallApply": true, /* Check that the arguments for 'bind', 'call', and 'apply' methods match their types */
    // "strictPropertyInitialization": true, /* Check for class properties that are declared but have no initial value */
    // "noImplicitThis": true, /* Enable error reporting when 'this' is given the wrong type */
    // "useUnknownInCatchVariables": true, /* Type catch clause variables as 'unknown' instead of 'any' */
    // "alwaysStrict": true, /* Ensure 'use strict' is always emitted */
    // "noUnusedLocals": true, /* Enable error reporting when a local variable is not used */
    // "noUnusedParameters": true, /* Raise an error when a function parameter isn't used */
    // "exactOptionalPropertyTypes": true, /* Interpret optional property types as written, rather than as nullable types */
    // "noImplicitReturns": true, /* Enable error reporting for code paths that do not contain a return statement */
    // "noFallthroughCasesInSwitch": true, /* Enable error reporting for fall-through cases in switch statements */
    // "noUncheckedIndexedAccess": true, /* Include 'undefined' in index signature results */
    // "noImplicitOverride": true, /* Ensure overriding members in derived classes are checked for compatibility */
    // "noPropertyAccessFromIndexSignature": true, /* Enforces using indexed accessors for keys declared with the index signature */
    // "allowUnusedLabels": true, /* Disable error reporting for unused labels */
    // "allowUnreachableCode": true, /* Disable error reporting for unreachable code */

    /* Completeness */
    // "skipDefaultLibCheck": true, /* Skip type checking .d.ts files that are included in a project */
    // "skipLibCheck": true, /* Skip type checking all .d.ts files */
  },
  "files": [
    "app.ts"
  ]
}
```

Fig. 4.2. 12: Contenido del fichero tsconfig.json.

Ejercicio 5:

Crear algunas de las posibles variables de tipados básicos existentes en lenguaje TypeScript (veremos un ejemplo en la Fig. 4.1.13). Para ello usaremos el archivo “app1.ts” usado en el ejercicio 2.

Crear también, en otro archivo distinto, un ejemplo de P.O.O. (Programación Orientada a Ojetos) básica en TypeScript. Esta nuevo archivo de clase en Typescript lo llamaremos “FicheroClase.ts”.

Paso 1: Definir algunas variables en el fichero “app1.ts”. Note que se pueden crear variables de tipado fuerte o tipado dinámico.

En JavaScript podíamos usar siempre tipado dinámico, pero eso podría dar origen a confusión.

TypeScript evita confusión con el Tipado Fuerte y fijo, aunque TypeScript tiene un tipo llamado ANY (equivaldría a “cualquier tipado” o “tipado dinámico”).

Véase la Fig. 4.2.13 para ver el caso ejemplo.

The screenshot shows the Visual Studio Code interface with the file 'app1.ts' open. The code defines several variables with different types:

```
1 console.log('Hola mundo');
2
3 // Tipos de datos de TypeScript
4
5 // Tipo String
6 let texto:string      = "esto es un texto tipo string";
7 // Tipo Int
8 let numero:number     = 1234;
9 // Tipo booleano
10 let trueFalse:boolean = false;
11 // Tipo ANY
12 let cualquiera:any   = texto;
13 cualquiera = numero;

14
15 // Tipo arrayDeString
16 let arrayString:string[] = ["texto1","texto2"];
17 // Tipo arrayDeNumeros
18 let arrayNumeros:number[] = [1,2,3];
19 // Tipo arrayDeBoolean
20 let arrayBoolean:boolean[] = [true,false, true];
21 // Tipo arrayDeAny
22 let arrayAny:any[]        = ["texto1",1,true];
```

Annotations highlight specific parts of the code:

- A green callout labeled "Tipos Compuestos (o array)" points to the array declarations (lines 16-22).
- A red callout labeled "Tipos Simples" points to the declaration of 'numero' (line 8).

Fig. 4.2.13: Ejemplos de variables tipadas en Typescript.

Nótese en la figura 4.2.13, que en las líneas 12 y 13 se define una variable de tipado dinámico (con el tipo “any”). Lo mismo sucede en la línea 22, ya que definimos un array que puede contener en sus posiciones cualquier tipo de dato.

Paso 2: Ahora crea un archivo nuevo con extensión TS, dentro de nuestra carpeta “TypeScript”. El nuevo archivo lo vamos a llamar “FicheroClase.ts” (ver Fig. 4.2.14). Dentro de este nuevo archivo vamos a crear una o más clases de código Typescript, y desarrollaremos un programa “Orientado a Objetos”.

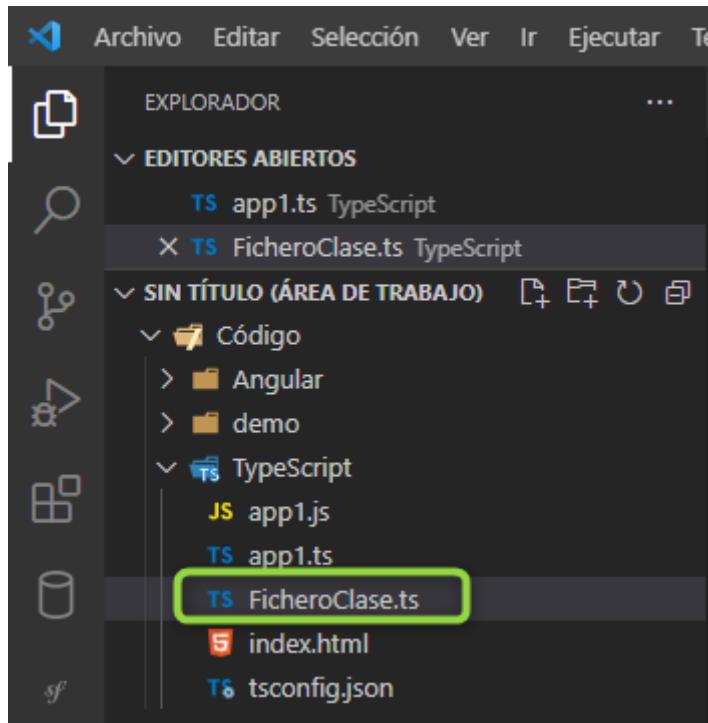


Fig. 4.2.14: Creación del nuevo archivo “FicheroClase.ts”.

Paso 3: Abrimos ahora el nuevo archivo y creamos la estructura básica de una clase en TypeScript (ver Fig. 4.1.15).

Definimos la estructura básica de una clase en Typescript haciendo 4 cosas:

- 1) Crearemos la cabecera de la nueva clase (con la palabra reservada “class”) y a llamaremos a esta clase “Calculadora”. Ver línea 1 de Fig. 4.2.15.
- 2) Crearemos una variable de instancia (o de clase) que se llamará “numero”. Ver línea 3 de Fig. 4.2.15.
- 3) Crearemos un constructor parametrizado de la clase: el constructor inicializa la variable de instancia y tendrá un parámetro de entrada de tipo “number”. Ver líneas 5-8 de Fig. 4.2.15.
- 4) Crear un objeto de tipo “Calculadora” fuera de dicha clase. Al nuevo objeto lo llamaremos “claseA” y tendrá un valor de entrada numérico igual a 4. Ver línea 10 de Fig. 4.2.15.

```
TS FicheroClase.ts ●
Código > TypeScript > TS FicheroClase.ts > ...
1  class Calculadora {
2
3      numero:number;
4
5      constructor(valor:number){
6          console.log('Iniciando calculadora...')
7          this.numero = valor
8      }
9  }
10 let claseA = new Calculadora(4);
11
```

Fig 4.2.15: Estructura básica de una clase en TypeScript.

Paso 4: Crearemos también algún método de tipo “getter” y/o “setter”.

En nuestro caso hemos definido un método de tipo getter, llamado “sumar()”, el cual devuelve el doble del valor que tenga la variable de instancia “numero”.

Verá en la Figura 4.2.16 el código creado de este paso 4: la creación del método “sumar()” de tipo getter se encuentra en las líneas 9-11, mientras que la llamada al método creado puede ver en la línea 14.

```
TS FicheroClase.ts ●
Código > TypeScript > TS FicheroClase.ts > ...
1  class Calculadora {
2
3      numero:number;
4
5      constructor(valor:number){
6          console.log('Iniciando calculadora...')
7          this.numero = valor
8      }
9      sumar(){
10         return this.numero + this.numero;
11     }
12 }
13 let claseA = new Calculadora(4);
14 console.log(claseA.sumar());
15
```

Fig. 4.2.16: Creación de métodos getter y setter de una clase en Typescript.

b) ¿Qué hemos explicado ya de Angular?

De la parte de Angular ya se explicó:

- Qué es Angular (apartado 2.6 completo, en pág. 18).
- Instalación de “Angular CLI” mediante el gestor de paquetes “npm” (+info. en apartado 3.6, paso 6 en pág. 78). Ver Fig. 4.2.17:

```
C:\Users\...\Frontend Angular>npm install -g @angular/cli
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

changed 219 packages, and audited 220 packages in 14s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 8.11.0 -> 8.13.1
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.13.1
npm notice Run npm install -g npm@8.13.1 to update!
npm notice
```

Fig. 4.2.17: Instalación cliente @angular/cli.

- Creación del nuevo proyecto de Angular, mediante el comando “ng” (+info. en apartado 3.6 paso 8 en, pág. 79). Ver Fig. 4.2.18:

```
C:\...\Frontend Angular>ng new appFrontend
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE appFrontend/angular.json (2948 bytes)
CREATE appFrontend/package.json (1043 bytes)
CREATE appFrontend/README.md (1065 bytes)
CREATE appFrontend/tsconfig.json (863 bytes)
CREATE appFrontend/.editorconfig (274 bytes)
CREATE appFrontend/.gitignore (548 bytes)
CREATE appFrontend/.browserslistrc (600 bytes)
CREATE appFrontend/karma.conf.js (1429 bytes)
CREATE appFrontend/tsconfig.app.json (287 bytes)
CREATE appFrontend/tsconfig.spec.json (333 bytes)
CREATE appFrontend/.vscode/extensions.json (130 bytes)
CREATE appFrontend/.vscode/launch.json (474 bytes)
```

Fig. 4.2.18: Creación nuevo proyecto Angular.

- Instalador del lenguaje y el transpilador de “TypeScript” (apartado 3.6, paso 11 en pág. 82).
- Instalación de complementos y pluggins para el IDE “Visual Studio Code”, recomendado para el desarrollo del proyecto (apartado 3.2 pasos 9 y 10 en pág. 80).
- Tutorial completo de TypeScript: tipos de datos, clases, constructores y P.O.O en TypeScript (apartado 4.2.a completo).

c) Conceptos Clave y Arquitectura software en Angular.

Los conceptos clave en Angular son 8, según documentación oficial de la web de Angular (ver Fig. 4.2.17):

- **Módulo:** es uno de los elementos principales de Angular. Los mismos proporcionan un contexto o dominio de aplicación para los componentes y servicios que contiene.
- **Componente:** Un componente representa una porción de la aplicación o página y esta contenido dentro de un módulo y esta dividido en la parte lógica (código) y la vista (HTML/CSS).
- **Servicio:** Un servicio es un componente lógico de código reutilizable que cumple con una función específica. Su principal objetivo es proveer funcionalidad extra a un componente.
- **Directiva:** Una directiva permite añadir, manipular o eliminar elementos del DOM del HTML. La misma es representada como un atributo más de una etiqueta HTML.
- **Data Bindings:** Es un mecanismo para vincular la vista (HTML) con su componente. Pueden ser del tipo código-a-vista, vista-a-código, bidireccional.
- **Inyección de dependencias (DI):** Angular cuenta con su propio framework de DI. Las dependencias generalmente son los servicios que una clase necesita para ejecutar una determinada función.

Fig. 4.2.17: Conceptos clave de Angular. Fuente: web oficial www.angular.io y canal de youtube “Android Desde Cero”.

Arquitectura de un programa desarrollado en Angular, según documentación oficial de la web de Angular (ver Fig. 4.2.18):

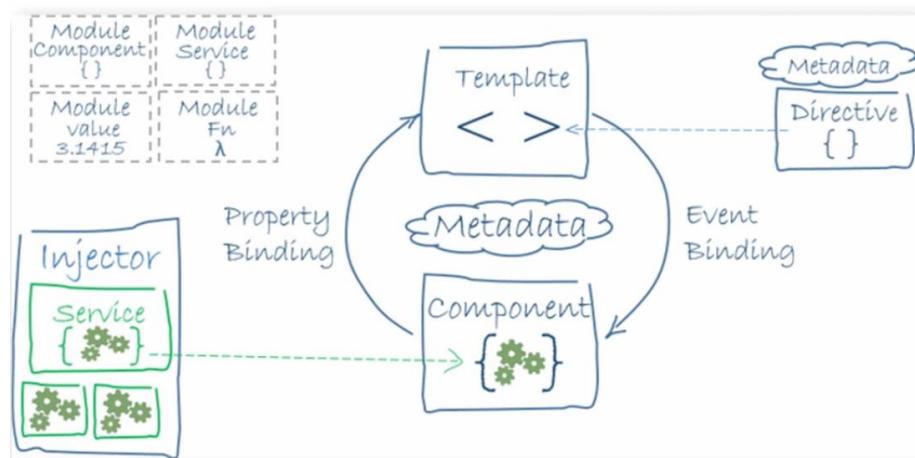


Fig. 4.2.18: Arquitectura en Angular. Fuente: web oficial www.angular.io y canal de youtube “Android Desde Cero”.

- **Una aplicación en Angular** se compone de 1 o más módulos, y cada módulo se compone de varios componentes/servicios.
- **Un componente** sería 1 HTML + 1 clase en Typescript-
- **Un servicio** sería la “lógica de negocio”.
- **Los módulos interactuarán** entre sí para renderizar las vistas en el navegador.

d) Probando nuestro nuevo proyecto.

Paso 1: Cargar en el área de trabajo de Visual Studio Code el nuevo proyecto Angular que hemos creado, llamado “appFrontend”.

- El nuevo proyecto de Angular ya lo teníamos creado (ver Fig. 1.4.18), y llamamos al nuevo proyecto “appFrontend”.
- El proyecto fue creado en el siguiente directorio:

“...\\Código\\”Frontend Angular”

- Incluiremos entonces el nuevo proyecto “appFrontend”, en nuestra área de trabajo de Visual Studio Code. Puede verse un par de ejemplos en las figuras “Fig. 3.94” y “Fig. 3.99”.
- Una vez cargado en nuestra área de trabajo de Visual Studio Code, el proyecto inicial de Angular se ve como detalla la Fig. 4.2.19:

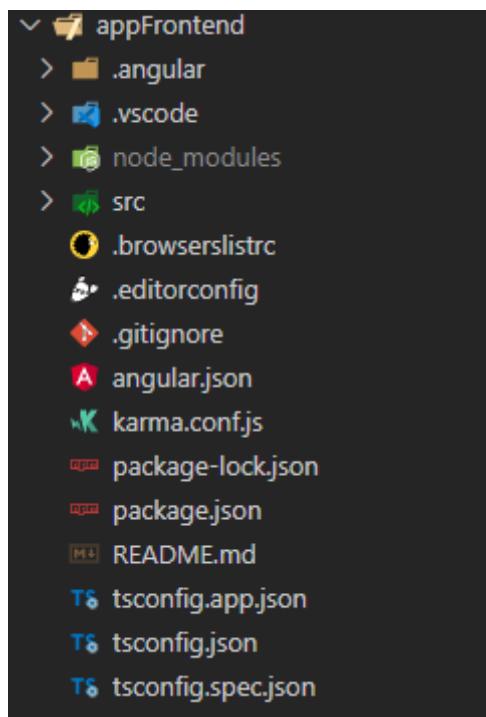


Fig. 4.22.19: Proyecto Angular recién creado, cargado en nuestra Área de Trabajo de Visual Studio Code.

Paso 2: Levantar el proyecto front-end en un servidor local.

- Primero, desde el terminal inferior de Visual Studio Code, ejecutar el comando que mantenga activo el servidor local de frontend. Tenga siempre en cuenta realizar la ejecución dentro del directorio del proyecto.

Ese comando será:

“ng serve”

- Confirmar que la respuesta del comando devuelva en consola diferentes archivos ejecutados, la Build se realiza e indica la fecha y hora de realización y que el servidor está escuchando por una IP local y puerto concretos (los cuales son informados por el servidor).
- Puede encontrarse una situación similar a la que se presenta en la figura Fig. 4.2.20:

```
PS C:\Código\Frontend Angular\appFrontEnd> ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files      | Names           | Raw Size
vendor.js                | vendor          | 1.73 MB
polyfills.js              | polyfills       | 313.43 kB
styles.css, styles.js     | styles          | 207.35 kB
main.js                   | main            | 48.33 kB
runtime.js                | runtime         | 6.52 kB

| Initial Total | 2.29 MB

Build at: 2022-06-15T11:45:45.322Z - Hash: ded2177a306b7cccd - Time: 2442ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200 **

✓ Compiled successfully.
```

Fig. 4.2.20: Ejecución del comando “ng serve” para activar el servidor frontend.

Paso 3: Con el servidor levantado y esperando peticiones del navegador web, probar a visualizar la web que viene por defecto.

Probar la correcta visualización (desde el navegador) y la no generación de errores por el terminal de consola (donde mantenemos el servidor levantado).

- Abrimos el navegador y cargamos la URL del deployment que nos indicaba el servidor local:

“localhost:4200”

- Debería verse una web similar a la que se muestra a continuación, en la figura “Fig. 4.2.21”.

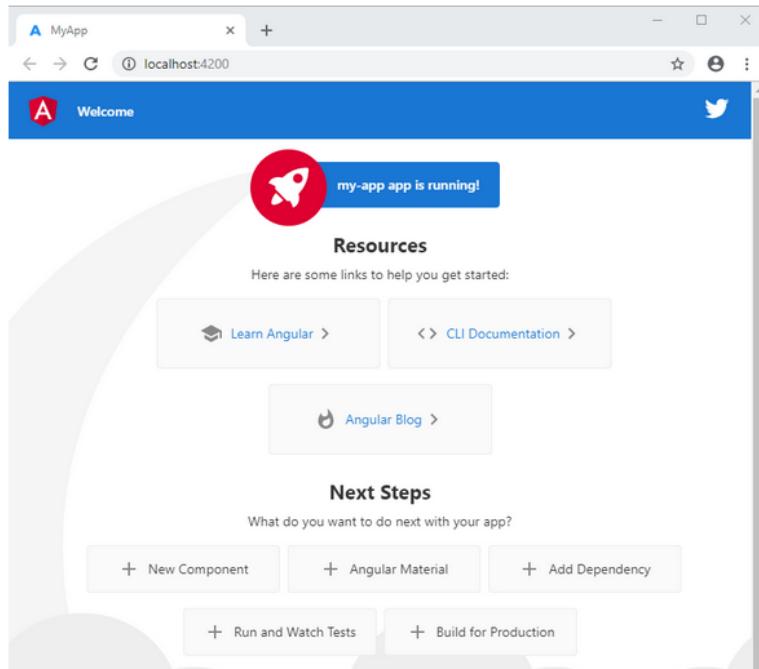


Fig. 4.2.21: Web de presentación del proyecto Angular.

e) Viendo el código que incluye nuestro nuevo proyecto.

Al abrir el proyecto Angular recién creado, de todo el código que incluye, nos interesan principalmente 2 cosas:

- 1) El **archivo “package.json”** (ver Fig. 4.2.22).
- Este archivo es similar a un “manifest” del proyecto, donde se informa de las dependencias que usa el proyecto (y alguna información extra importante para el proyecto).
- 2) El **directorio “src”** (ver Fig. 4.2.22).
- Este directorio englobará los desarrollos funcionales del proyecto, y agrupará esos desarrollos en los subdirectorios:
 - “app”.
 - “assets”.
 - “environments”.
 - <<subdirectoriosAdicionalesPorNuevosComponentes>>.
- Archivo “main.ts”.

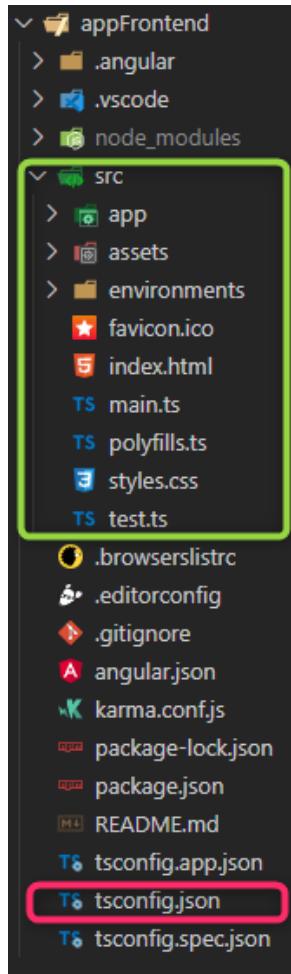


Fig. 4.2.22: Arquitectura con archivos y directorios claves.

Analizando un poco el contenido del archivo principal de configuración y el directorio principal, se tiene:

1) Análisis del contenido del archivo “**package.json**”:

- Nombre, versión y comandos script para levantar el servidor del frontend en Angular, hacer la build del proyecto, test, etc (ver “Fig. 4.2.23”).

```
appFrontend > package.json > {} dependencies > @angular/core
1  {
2    "name": "app-frontend",
3    "version": "0.0.0",
4    "scripts": {
5      "ng": "ng",
6      "start": "ng serve",
7      "build": "ng build",
8      "watch": "ng build --watch --configuration development",
9      "test": "ng test"
10 },
Depuración de
```

Fig. 4.2.23: “**package.json**”, comandos para levantar los servicios ofrecidos.

- (b) Módulos que va a necesitar la aplicación para poder funcionar (librerías o dependencias), declarados de forma privada (ver Fig. 4.2.24).

```

11  "private": true,
12  "dependencies": [
13    "@angular/animations": "^14.0.0",
14    "@angular/common": "^14.0.0",
15    "@angular/compiler": "^14.0.0",
16    "@angular/core": "^14.0.0",
17    "@angular/forms": "^14.0.0",
18    "@angular/platform-browser": "^14.0.0",
19    "@angular/platform-browser-dynamic": "^14.0.0",
20    "@angular/router": "^14.0.0",
21    "rxjs": "~7.5.0",
22    "tslib": "^2.3.0",
23    "zone.js": "~0.11.4"
24  ],

```

Fig. 4.2.24: “package.json”, dependencias para servicios que trae el proyecto.

- (c) Dependencias necesarias para desarrollar código en el proyecto (ver Fig. 4.2.25).

```

25  "devDependencies": {
26    "@angular-devkit/build-angular": "^14.0.4",
27    "@angular/cli": "~14.0.4",
28    "@angular/compiler-cli": "^14.0.0",
29    "@types/jasmine": "~4.0.0",
30    "jasmine-core": "~4.1.0",
31    "karma": "~6.3.0",
32    "karma-chrome-launcher": "~3.1.0",
33    "karma-coverage": "~2.2.0",
34    "karma-jasmine": "~5.0.0",
35    "karma-jasmine-html-reporter": "~1.7.0",
36    "typescript": "~4.7.2"
37  }
38 }
39

```

Fig. 4.2.25: “package.json”, dependencias para desarrollar código que trae el proyecto.

2) Análisis del contenido del directorio “src”:

- (a) El subdirectorio “src/assets” se suele usar para poner diferentes hojas de estilos.
- (b) El subdirectorio “src/app” es donde viene definido nuestro primer componente, y se usará como componente inicial.
- (c) Archivo “src/main.ts”: dentro del directorio “src”. Es el archivo que indica a la aplicación “cómo debe arrancarse” (ver Fig. 4.2.26).

```
appFrontend > src > ts main.ts > ...
1 import { enableProdMode } from '@angular/core';
2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4 import { AppModule } from './app/app.module';
5 import { environment } from './environments/environment';
6
7 if (environment.production) {
8 | enableProdMode();
9 }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12 .catch(err => console.error(err));
```

Fig. 4.2.26: Fichero “main.ts”, el cual arranca la aplicación.

- El “main.ts” llama al fichero “app.module.ts” (en Fig. 4.2.27). Ya en el “app.module.ts” se llamará al fichero “app.component.ts” (que está en dentro del directorio “src/app”).

```
appFrontend > src > app > ts app.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5
6 @NgModule({
7   declarations: [
8     AppComponent
9   ],
10  imports: [
11    BrowserModule
12  ],
13  providers: [],
14  bootstrap: [AppComponent]
15 })
16 export class AppModule { }
```

Fig. 4.2.27: Contenido del archivo “app.module.ts” → llamado desde el “main.ts”.

- Por último, una vez llegados al interior del fichero “appComponent.ts” (ver Fig. 4.2.28), llamaremos a los HTML y al CSS:
 - templateUrl: './app.component.html' (Fig. 4.2.29).
 - styleUrls: ['./app.component.css'] (Fig. 4.2.29).

```

appFrontend > src > app > TS app.component.ts > ...
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9     title = 'appFrontend';
10 }

```

Fig. 4.2.28: Contenido del archivo “app.component.ts” → llama a los HTML y CSS que use.

![Angular logo](data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHN2pzXdc3ig9Ijg=.)

```

<!-- Toolbar -->
<div class="toolbar" role="banner">

<span>Welcome</span>
<div class="spacer"></div>
<a aria-label="Angular on twitter" target="_blank" rel="noopener" href="https://twitter.com/angular"><img id="twitter-logo" height="24" data-name="Logo" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 400 400" aria-label="Angular on Twitter" target="_blank" rel="noopener" href="https://twitter.com/angular"/></a>
<a aria-label="Angular on YouTube" target="_blank" rel="noopener" href="https://youtube.com/angular"><img id="youtube-logo" height="24" width="24" data-name="Logo" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 400 400" aria-label="Angular on YouTube" target="_blank" rel="noopener" href="https://youtube.com/angular"/></a>

```

Fig. 4.2.29: Señalados en rojo y azul los archivos CSS y HTML usados, y mostrado el contenido del archivo “app.component.html”.

f) Módulos y Componentes en Angular.

¿Qué eran los módulos en Angular? Donde se almacenaban nuestros componentes y servicios de la aplicación final.

¿Qué encontramos en un módulo? (ver Fig. 4.2.30):

- Está escrita en lenguaje Typescript, y será una clase en Typescript.
- Nos encontramos las importaciones necesarias, que como mínimo serán el “BrowserModule”, “NgModule” y el “AppComponent”.
- Etiquetas para configurar nuestro módulo, como es el caso de @NgModule. Bajo esta etiqueta se configurará todo lo necesario para el módulo.

The screenshot shows the file structure of an Angular application named 'appFrontend'. Inside the 'src' folder, there is an 'app' folder containing 'app.module.ts', which is highlighted with a red rectangle. A green arrow points from the left margin of this file towards the code editor. The code in 'app.module.ts' is as follows:

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 
4 import { AppComponent } from './app.component';
5 
6 @NgModule({
7   declarations: [
8     AppComponent
9   ],
10  imports: [
11    BrowserModule
12  ],
13  providers: [],
14  bootstrap: [AppComponent]
15 })
16 export class AppModule { }
```

Fig. 4.2.30: Componente principal de nuestra aplicación Angular.

¿Qué son los componentes?

- Son los bloques básicos con los que construimos las webs en Angular. Por ello, cada componente tiene una parte visual (HTML+CSS), la parte del controlador back (clase TypeScript) y metadatos.

¿Qué encontramos en un componente?

- Una clase TypeScript (que aporta la lógica al componente) y varios archivos visuales (HTML y CSS).
- La clase TypeScript empaqueta el resto de ficheros del componente, y así poder después exportar este componente.
- La clase TypeScript viene a realizar la función de controlador software, y vincula la template, los estilos y la creación de la nueva etiqueta HTML que crea nuestro componente (llamada “selector”). Ver Fig. 4.2.31:
 - **El selector del componente**, para usarlo en ficheros HTML externos como “index.html”.

- **La template**, que vincula el HTML del componente.
- **Los estilos**, que vinculan los CSS al HTML anterior.
- Después **podremos incluir el código HTML y los CSS de nuestro componente dentro de “index.html”**. Para ello, poner la etiqueta HTML que se llame como el selector del componente, y esto cargará todo el código del componente (ver Fig. 4.2.32).

```
appFrontend > src > app > app.component.ts > ...
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'appFrontend';
10 }
11
```

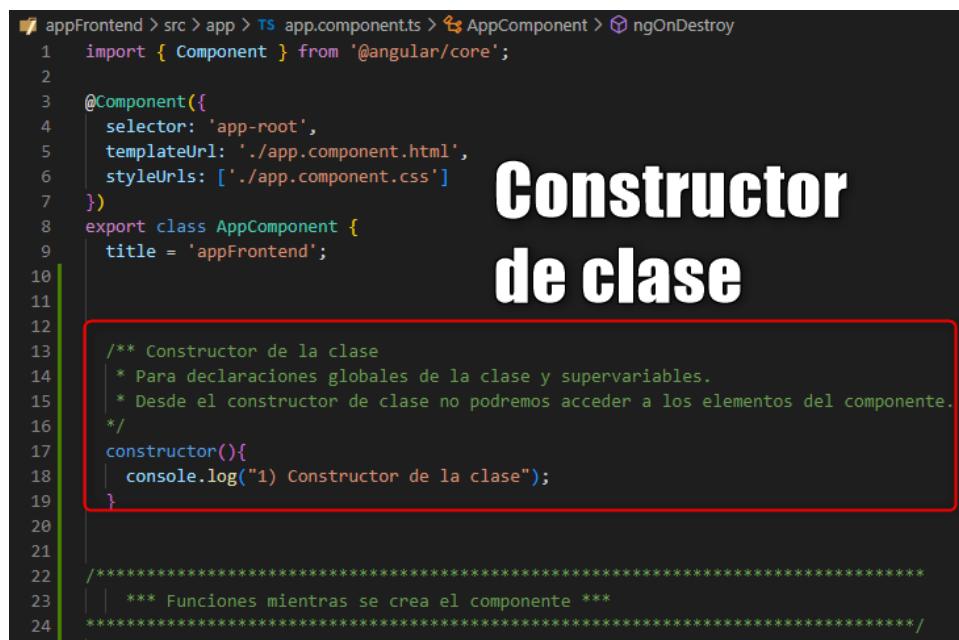
Fig. 4.2.31: Cargando ficheros del componente, en líneas 4, 5 y 6.

```
appFrontend > src > index.html > ...
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <title>AppFrontend</title>
6     <base href="/">
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8     <link rel="icon" type="image/x-icon" href="favicon.ico">
9   </head>
10  <body>
11    <app-root></app-root>
12  </body>
13 </html>
14
```

Fig. 4.2.32: Uso de la nueva etiqueta HTML en index.html, creada previamente en nuestro componente “app”.

¿Cuál es el ciclo de vida de un componente en Angular y cómo funciona?

- En la figura 4.1.31 veíamos el controlador TypeScript que trae nuestro componente inicial del proyecto.
- Sobre esta misma clase TypeScript vamos a crear las diferentes funciones que forman parte del ciclo de vida de cualquier componente:
 - **Una función** “constructor” de la clase TypeScript (construye la clase, no el componente).
 - **Tres funciones** previas a la carga del componente (se ejecutan antes de que el componente termine de crearse o cargarse).
 - **Cuatro funciones** posteriores a la carga el componente (se ejecutan después de terminar de crear el componente).
 - **Una función** Destroy, para destrucción del componente (después de terminar de cargarse el componente, la función se ejecuta solo cuando sea invocada).
- Veremos en las siguientes figuras (Fig. 4.2.33, Fig. 4.2.34 y Fig. 4.2.35) las 9 funciones relacionadas con el ciclo de vida de un componente en Angular. Estas 9 funciones se crean en la clase TypeScript del componente que viene por defecto en el proyecto.
- También veremos en la figura Fig. 4.2.36 el orden real de ejecución de las funciones antes mencionadas, que visualizamos en la consola del navegador con un “console.log()”.



```
appFrontend > src > app > ts app.component.ts > App Component > ngOnDestroy
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'appFrontend';
10
11
12
13 /**
14  * Constructor de la clase
15  * Para declaraciones globales de la clase y supervariables.
16  * Desde el constructor de clase no podremos acceder a los elementos del componente.
17  */
18 constructor(){
19   console.log("1) Constructor de la clase");
20 }
21
22 /**
23  * Funciones mientras se crea el componente ***
24 ****/

```

Constructor de clase

Fig. 4.2.33: creación de un constructor de la clase no parametrizado (ejecutado el primero y automáticamente para crear la clase).

```

17  constructor(){
18    |  console.log("1) Constructor de la clase");
19  }
20
21
22 /**
23  * *** Funciones mientras se crea el componente ***
24 ****
25
26 /** El "ngOnChanges" es la primera funcion que se ejecuta de nuestro componente.
27 * Se ejecuta cada vez que un valor de entrada se modifica o aparezca.
28 * Esta funcion se dispara cada vez que le llegue un valor al componente desde el exterior del componente.
29 */
30 ngOnChanges(): void {
31   |  console.log("2)ngOnChanges");
32 }
33
34 /** Se ejecuta después del "ngOnChanges".
35 * Es el constructor interno del componente.
36 * Se ejecuta una sola vez y nunca mas.
37 */
38 ngOnInit(): void {
39   |  console.log("3) ngOnInit - Constructor del componente");
40 }
41
42 /** Se ejecuta despues del ngOnInit, y vuelve a ejecutarse cada vez que se realicen cambios en el componente.
43 * Esta funcion sirve para comprobar constantemente si algo de nuestro componente ha cambiado.
44 */
45 ngDoCheck(): void {
46   |  console.log("4) ngDoCheck");
47 }
48
49

```

Funciones ejecutadas antes de terminar la creación del componente.

Fig. 4.2.34: creación de las 3 funciones que se ejecutan antes de terminar de crear el componente (automáticamente).

```

50
51 /**
52  * *** Funciones después de que se cree el componente ***
53  * -> Estas funciones forman parte del componente, porque afecta al template del componente.
54  * -> No sirven en un servicio ni con directivas.
55 ****
56
57 /** Se carga cuando todo el contenido del HTML este cargado. */
58 ngAfterContentInit(): void {
59   |  console.log("5) ngAfterContentInit");
60 }
61
62 /** se ejecuta cuando haya algo modificado despues de terminar de cargar el HTML */
63 ngAfterContentChecked(): void {
64   |  console.log("6) ngAfterContentChecked");
65 }
66
67 /** Para comprobar que los Views y ViewChild estan cargados.
68 * La funcion que mas se utiliza del grupo de funciones "ngAfter...".
69 */
70 ngAfterViewInit(): void {
71   |  console.log("7) ngAfterViewInit");
72 }
73
74 /** Se ejecuta cada vez que un View o ViewChild se modifica.
75 *
76 */
77 ngAfterViewChecked(): void {
78   |  console.log("8) ngAfterViewChecked");
79 }
80
81
82 /**
83  * *** Funcion para destruir el componente ***
84 ****
85
86 /** ngOnDestroy: si tenemos muchos componentes activos y dejamos de usar alguno, es aconsejable destruirlo.
87 * En los componentes se pueden hacer muchas operaciones: listener, observables, llamadas, etc.
88 * Con ngOnDestroy, se cierran los listener, observables, llamadas, etc.
89 * Luego se destruye el componente.
90 */
91 ngOnDestroy(): void {
92   |  console.log("9) ngOnDestroy");
93 }
94

```

Funciones cargadas después de crear el componente.

Función Destroy del componente.

Fig. 4.2.35: creación de las 4 funciones que se ejecutan posteriormente a la creación del componente (automáticamente), y la función Destroy usada para destruir el componente y todo lo relacionado con él (su ejecución será manual).

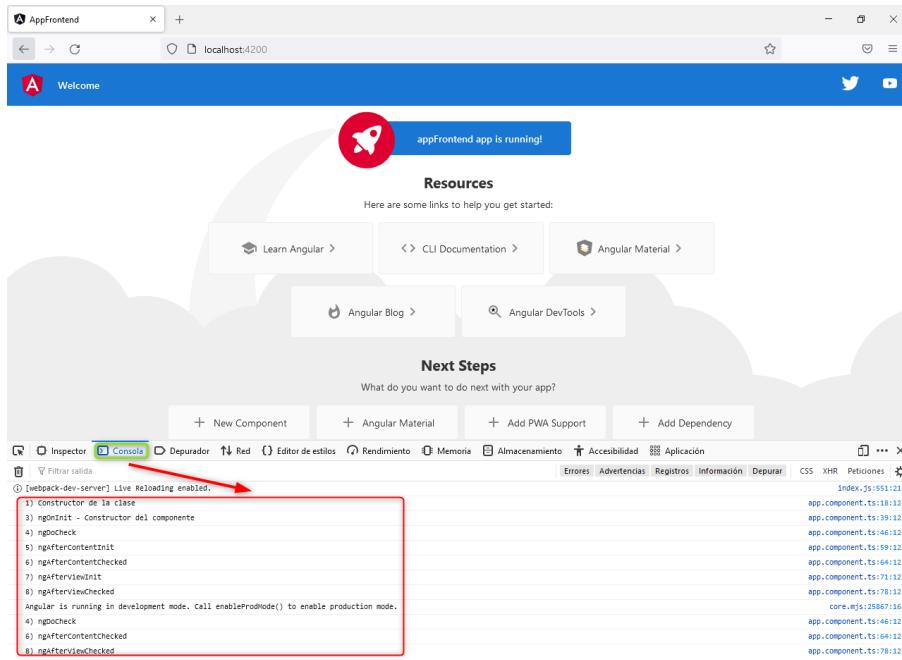


Fig. 4.2.36: Pruebas en la consola del navegador, para visualizar el orden de ejecución de las funciones del ciclo de vida del componente principal que trae nuestro proyecto.

Hasta el momento hemos estado viendo solamente el componente inicial que viene con la creación del proyecto, llamado “app”.

También se ha comprobado que el diseño de la página inicial se carga desde el “index.html”, pero que no está su código en el “index.html”, sino que lo importa.

El funcionamiento de la carga del componente en el “index.html” es el siguiente:

- En el “index.html”, dentro de su sección “body” se llama al nuevo elemento html correspondiente al selector “app-root” (selecciona el componente “app”).
- De este modo, el navegador cargará el “index.html”, pero el html conseguirá el código llamando al componente “app” desde el “index.html”.
- Podemos ver los ejemplos de lo explicado en las figuras 4.2.31 y 4.2.32:
 - En la “Fig. 4.2.31” vemos el selector “app-root” creado desde el controlador del componente “app”.
 - La llamada del contenido de la web, realizado en “index.html” llamando al componente “app” usando su selector puede verse en la “Fig. 4.2.32”.

En Angular una página padre se puede formar a partir de HTML’s hijos (el html de un componente hijo del componente actual). Por eso el fichero “index.html” podría considerarse un contenedor del contenido de la página principal, pero será el HTML de componente “app” el que aportará el contenido de la página cuando se llame este componente usando su selector.

g) Primer componente en Angular: creación, configuración y uso.

El nuevo componente se llamará “primer-componente-formulario-login”, y será un “componente hijo” del componente “app”.

Para crear un nuevo componente que sea hijo del componente “app”, ir a la consola de Visual Studio Code y posicionarse en la ruta raíz del proyecto front-end.

Usaremos un comando de “angular cli” para crear el componente y las clases que necesita. El comando se llama “ng generate component [nombre-componente-nuevo]”. Ver Fig. 4.2.37 para ver el detalle de la creación del componente.

Nota: Es muy importante usar el guion a la hora de poner el nombre del nuevo componente, si este es un nombre compuesto.

```
PS C:\          \appFrontend> ng generate component primer-componente-formulario-login
CREATE src/app/primer-componente-formulario-login/primer-componente-formulario-login.component.html (49 bytes)
CREATE src/app/primer-componente-formulario-login/primer-componente-formulario-login.component.spec.ts (777 bytes)
CREATE src/app/primer-componente-formulario-login/primer-componente-formulario-login.component.ts (384 bytes)
CREATE src/app/primer-componente-formulario-login/primer-componente-formulario-login.component.css (0 bytes)
UPDATE src/app/app.module.ts (502 bytes)
PS C:\          \appFrontend>
```

Fig. 4.2.37: Creación del nuevo componente por línea de comandos, usando “@angular/CLI”.

Al crear el nuevo componente “primer-componente-formulario-login”, gracias a @angular/cli se crearon los 4 ficheros necesarios para conformar el nuevo componente: extensiones “.ts”, “.html”, “.css” y “.spec.ts”. El nuevo componente aparece dentro del componente “app”, y los 4 ficheros del nuevo componente aparecen dentro del nuevo componente. Todo esto puede verse en la Fig. 4.2.38.

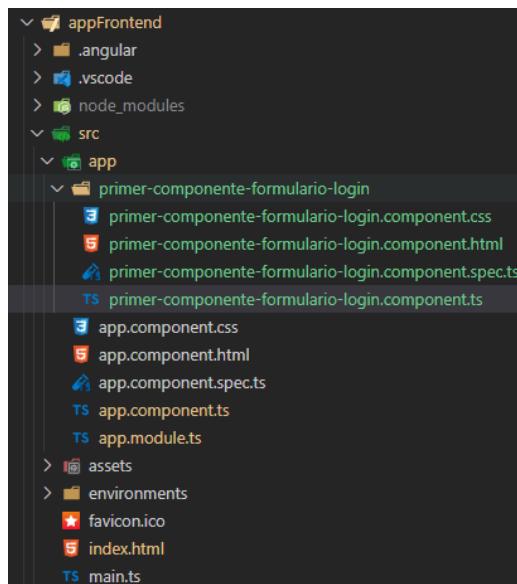


Fig. 4.2.38: Visualización del nuevo componente dentro del componente “app”, así como los 4 ficheros del nuevo componente que vendrán dentro del nuevo componente.

Si vamos al controlador del primer componente creado, es decir, “primer-componente-formulario-login.component.ts”, encontramos el siguiente código:

- Nos importará las clases “Component” y “OnInit”, desde @angular/core.
- Después usa el metadato “@Component”, donde asigna el nombre del selector, la vista HTML y los estilos CSS del componente.
- Cabecera de la clase typescript: con el modificador de acceso “export” al comienzo de la cabecera y al final “implement OnInit”.
- Dentro del código de la clase ya nos vienen las cabeceras del constructor no parametrizado de la clase y de la función “ngOnInit” (donde pondremos los valores iniciales del componente).
- Véase la siguiente imagen, Fig. 4.2.39, para ver un ejemplo con mayor detalle.



```

appFrontend > src > app > primer-componente-formulario-login > TS primer-componente-formulario-login.component.ts
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-primer-componente-formulario-login',
5    templateUrl: './primer-componente-formulario-login.component.html',
6    styleUrls: ['./primer-componente-formulario-login.component.css']
7  })
8  export class PrimerComponenteFormularioLoginComponent implements OnInit {
9
10  constructor() { }
11
12  ngOnInit(): void {
13  }
14}
15

```

Fig. 4.2.39: Contenido inicial del controlador .ts del nuevo componente.

Ahora volvemos a ver el estado de nuestra aplicación Angular en nuestro navegador favorito. Para ello ejecutemos el servidor front-end desde el terminal de comandos disponible en Visual Studio Code, ubicándonos en la raíz del proyecto Angular. El comando que necesitamos ejecutar será: “ng serve”.

Cuando levante el servidor de front-end nos dará la información de la URL donde podemos encontrar la web inicial: <http://localhost:4200/>.

Nos saldrá en el navegador la misma web de la figura “Fig. 4.2.21”, ya que, aunque hayamos creado un nuevo componente, éste aún no se está usando.

Para incluir en nuestra web inicial el nuevo componente, sigue los siguientes pasos:

- 1º) Ve a la vista del nuevo componente (el archivo HTML), llamado “primer-componente-formulario-login.html”, y modifica el contenido del fichero para que solo contenga el siguiente código:

<p>Hola mundo... el componente "primer-componente-formulario-login" FUNCIONA!</p>

- 2º) Despues ir al controlador del nuevo componente, el fichero “primer-componente-formulario-login.ts” y copiar su selector:

- El selector se llama “app-primer-componente-formulario-login”.
- 3º) Ir al fichero que contiene la vista (el HTML) del componente “app” e incluir al final del archivo varias llamadas consecutivas del nuevo componente “primer-componente-formulario-login”. Tenemos un ejemplo de esto en la siguiente imagen, la Fig. 4.2.40.

```

477 <!-- * * * * * * * * * * The content above * * * * * * * * * -->
478 <!-- * * * * * * * * * is only a placeholder * * * * * * * * * -->
479 <!-- * * * * * * * * * and can be replaced. * * * * * * * * * -->
480 <!-- * * * * * * * * * * * * * * * * * * * * * * * * * * * -->
481 <!-- * * * * * * * * * End of Placeholder * * * * * * * * * -->
482 <!-- * * * * * * * * * * * * * * * * * * * * * * * * * * * -->
483
484
485
486
487 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
488 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
489 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
490 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
491 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
492 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
493 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
494 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
495 <app-primer-componente-formulario-login></app-primer-componente-formulario-login>
```

Fig. 4.2.40: Llamadas consecutivas al selector del nuevo componente, dentro de la vista del componente padre (componente “app”).

- 4º) Nos aseguramos de que el nuevo componente está siendo detectado por el módulo principal. Para ello ir al fichero “app.module.ts” y debemos asegurarnos de que se cargue el nuevo componente en:
 - Los imports.
 - Las declaraciones del metadato @NgModule.

Puede ver un ejemplo de esto en la imagen siguiente (Fig. 4.2.41), donde debe aparecer el nuevo componente que usaremos y el componente “app” que ya se estaba usando.

```

1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5 import { PrimerComponenteFormularioLoginComponent } from './primer-componente-formulario-login/';
6
7 @NgModule({
8   declarations: [
9     AppComponent,
10     [PrimerComponenteFormularioLoginComponent]
11   ],
12   imports: [
13     BrowserModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
```

Fig. 4.2.41: configuración del nuevo componente en el módulo principal.

Nota:

- Gracias al comando que nos ofrece @angular/cli llamado “ng generate component primer-componente-formulario-login” se importan los componentes nuevos de forma automática. El código del nuevo componente aparecerá también añadido en el fichero “app.modulo.ts”.
- También se puede crear un nuevo componente manualmente, es decir, sin usar el comando “ng generate component ...” sino creando los archivos manualmente uno a uno. En este caso el proyecto no lo enlazará automáticamente en “app.modulo.ts”, y tendrás que incluirlos manualmente (tanto el componente en el archivo del módulo, como la carpeta de nuevo componente y los 4 ficheros característicos de un componente).

Si los 4 pasos anteriores se realizaron correctamente, se deben guardar todos los archivos. Como teníamos levantado el servidor front-end con “ng serve”, al guardar todos los archivos, la web se actualizará sola,

Ir al navegador, y entonces veremos que se han cargado las líneas del nuevo componente al final de la web.

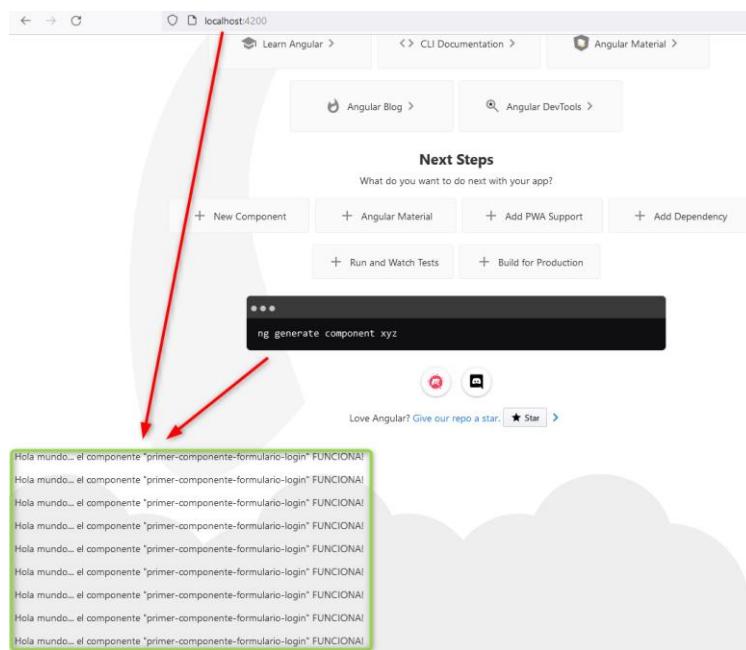


Fig. 4.2.42: Visualización del texto incluido por la vista del nuevo componente, al final de la vista del componente “app”.

Una vez se hayan podido visualizar desde el navegador en la página inicial que el nuevo componente se logra pintar, eliminar las llamadas reiteradas a un solo selector. Para ello de la imagen Fig. 4.2.40 de la línea de código 487 a la 495 solo dejaremos la llamada de la línea 487, dejando solamente uno activo. Se verá entonces una sola vez (Fig. 4.2.43).



Fig. 4.2.43: Visualización de una sola vez del nuevo componente.

Ahora crear un formulario de login, tal como indica el nombre del nuevo componente.

Para crear nuestro primer formulario con Angular en el componente “primer-componente-formulario-login”, como ya se está usando este componente correctamente, realizar solo 2 pasos:

1. Ir al fichero de la vista (HTML) de “primer-componente.component.html” y escribir el siguiente código para crear el formulario:

```

<p>
    User Name
</p>
<input type="text" name="userName">
<p>
    Password
</p>
<input type="text" name="password">
<p>
    <button (click)="alertaDeEnvioFormulario()">
        Entrar
    </button>
</p>

```

2. Ir al controlador Typescript del nuevo componente (“primer-componente-formulario-login.component.ts”) y añadiremos la siguiente función Typescript:

```

alertaDeEnvioFormulario (){

    alert('hola')
}

```

Los archivos typescript y html que acabamos de modificar en nuestro nuevo componente quedan como se ven en la siguiente imagen:

```

1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-primer-componente-formulario-login',
5    templateUrl: './primer-componente-formulario-login.component.html',
6    styleUrls: ['./primer-componente-formulario-login.component.css']
7  })
8  export class PrimerComponenteFormularioLoginComponent implements OnInit {
9
10    constructor() { }
11
12    ngOnInit(): void {
13    }
14
15    alertaDeEnvioFormulario(){
16      alert('El formulario de Login se ha enviado.');
17    }
18
19

```

Fig. 4.2.44: Creación de la función en el controlador Typescript, encargada de generar un Alert cuando se invoque esta función.

```

1  <p>Hola mundo... el componente "primer-componente-formulario-login" FUNCIONA!</p>
2
3  <p>
4    | User Name
5  </p>
6  <input type="text" name="userName">
7  <p>
8    | password
9  </p>
10 <input type="text" name="password">
11 <p>
12   |<button (click)="alertaDeEnvioFormulario()">
13   |   Entrar
14   |</button>
15 </p>

```

Fig. 4.2.45: Creación del HTML necesario para crear el formulario de login en la vista del nuevo componente. Elemento “button” con evento “click” asociado a una función creada en el controlador Typescript.

La prueba de que funciona correctamente la podemos encontrar en el navegador.
Debemos comprobar que:

- El formulario se ve correctamente, además de lo que ya escribía nuestro nuevo componente.
- En el formulario se debe poder escribir.
- El formulario, aunque aun no pueda enviar los datos a un backend, sí podrá generar un mensaje de “alert” cada vez que se pulse al botón “Entrar”.

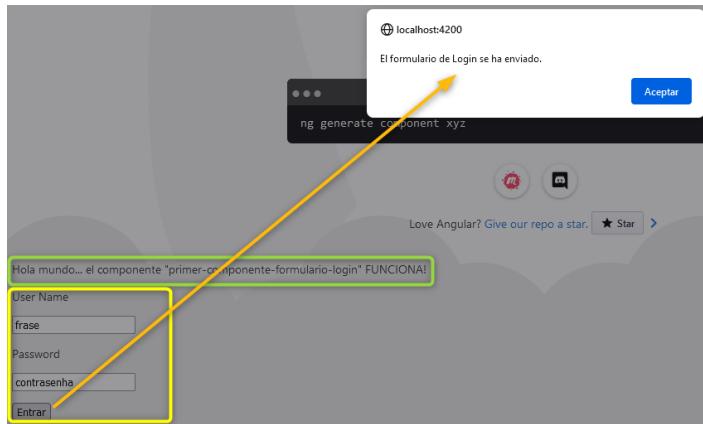


Fig. 4.2.46: Visualización del nuevo formulario de login, que genera mensaje Alert cuando se pulsa el botón “Entrar”.

h) Transferencia de datos entre componentes anidados.

Ya hemos visto que un componente contiene 4 archivos: vistas HTML, estilos CSS, controlador Typescript y un fichero extra para la configuración del componente.

Ahora vamos a transferir datos entre 2 componentes (distintos y anidados uno dentro del otro). Para ello se va a usar la vista HTML y el controlador Typescript.

La comunicación entre componentes puede ser en ambos sentidos: datos generados por el componente padre transferidos al componente hijo, o vice versa. Esta funcionalidad será útil en casos en los que “siendo generados los datos en un componente X, se muestren en el otro componente Y”.

Para explicar esta funcionalidad, los componentes usados serán los ya conocidos:

- Componente padre, llamado “App”.
- Componente hijo, llamado “Primer-componente-formulario-login”.

Para poder hacer uso de esta nueva funcionalidad, también se presentan 2 nuevas nuevas directivas de Angular:

- @Input
- @Output

La secuencia de modificaciones en los archivos que se necesita para conseguir la nueva funcionalidad estará compuesta por 4 pasos (uno por archivo):

Paso 1: Modificar Controlador Typescript del elemento padre (aquí llamado “App”):

- Crear una nueva variable de instancia en el controlador Typescript.

Paso 2: Modificar vista HTML del componente padre (App):

- Añadir un atributo al selector que ya estábamos usando en este HTML (revisar la “Fig. 4.1.40” para recordar la creación por primera vez).

Paso 3: Modificar Controlador Typescript del elemento hijo (aquí llamado “primer-componente-formulario-login”).

- Crear la directiva @Input o @Output, según su caso, en el controlador Typescript del componente hijo.

Paso 4: modificar vista HTML del componente hijo (primer-componente-formulario-login).

- La vista podría usar la directiva que se haya elegido poner en su controlador de 2 formas distintas:
 - 1) Si se usó @Input, pintando los datos recibidos en el controlador (comunicación DESCENDENTE).
 - 2) Si se usó @Output, creando un elemento HTML que se asocia a un evento y llame a una función creada en el controlador (comunicación ASCENDENTE).

El código visual de los 4 pasos recién explicados teóricamente podemos verlo en las 4 imágenes siguientes (desde la Fig. 4.2.47 hasta la Fig. 4.2.50).

```
appFrontend > src > app > ts app.component.ts > ...
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'appFrontend';
10
11   datosEnviadosAHIjoCompApp:String = "Datos del Comp.Padre enviados y escritos en el Comp.Hijo.";
12   datosRecibidosDelHIjoCompApp:String = "";
13 }
```

Fig. 4.2.47: (a) En el controlador de comp. padre, creación de variable de instancia inicializada para transferencia DESCENDENTE de datos (línea 11 color verde). (b) En el controlador de comp. padre, creación de variable de instancia no inicializada para transferencia ASCENDENTE de datos (línea 12 color rojo).

```
appFrontend > src > app > app.component.html > ...
485 {{datosRecibidosDelHijoCompApp}}
486
487 <br>
488
489 <app-primer-componente-formulario-login [datosRecibidosEnComphijo] ="datosEnviadosAlHijoCompApp"
490 | (datosEnviadosPorComphijo) ="datosRecibidosDelHijoCompApp=&event">
491 </app-primer-componente-formulario-login>
```

Fig. 4.2.48: (a) Comunicación DESCENDENTE de datos conseguida en el HTML del comp. padre, añadiendo el atributo de la línea 439 remarcado en verde. (b) Comunicación ASCENDENTE de datos conseguida en el HTML del comp. padre, añadiendo el atributo de la línea 440 remarcado en rojo. También se recupera el valor de la variable de instancia de su controlador en la línea 485 remarcada en rojo.

```

appFrontend > src > app > primer-componente-formulario-login > primer-componente-formulario-login.component.ts > ...
1   import { Component, Input, OnInit, Output, EventEmitter } from '@angular/core';
2
3   @Component({
4     selector: 'app-primer-componente-formulario-login',
5     templateUrl: './primer-componente-formulario-login.component.html',
6     styleUrls: ['./primer-componente-formulario-login.component.css']
7   })
8   export class PrimerComponenteFormularioLoginComponent implements OnInit {
9
10    @Input() datosRecibidosEnCompHijo : any;
11
12    constructor() { }
13
14    ngOnInit(): void {
15      console.log('OnInit() del comp. hijo ejecutado.')
16    }
17
18    alertaDeEnvioFormulario(){
19      alert('El formulario de Login se ha enviado.')
20    }
21
22    @Output() datosEnviaPorCompHijo = new EventEmitter();
23    enviarDatosHaciaAppComponentAlGenerarEvento(){
24      this.datosEnviaPorCompHijo.emit('DATOS DEL HIJO, RECIBIDOS Y ESCRITOS EN EL COMP. PADRE.')
25    }
26  }

```

Fig. 4.2.49: (a) En el controlador de comp. hijo, creación de la directiva @Input para transferencia DESCENDENTE de datos (línea 10 color verde). (b) En el controlador de comp. padre, creación de la directiva @Output para transferencia ASCENDENTE de datos (líneas 22-25 color rojo).

```

appFrontend > src > app > primer-componente-formulario-login > primer-componente-formulario-login.component.html > ...
Go to component
1 <table border>
2   <tr>
3     <td>
4       <p>Frase inicial del "primer-componente-formulario-login".</p>
5       <p>
6         User Name
7       </p>
8       <input type="text" name="userName">
9       <p>
10        Password
11      </p>
12      <input type="text" name="password">
13      <p>
14        <button (click)="alertaDeEnvioFormulario()">
15          Entrar
16        </button>
17      </p>
18
19      <!-- Recepcion de datos con @Input: desde componente padre a componente hijo -->
20      <p>{{datosRecibidosEnCompHijo}}</p>
21
22      <!-- Envio de datos con @Output: desde componente hijo a componente padre, con evento click -->
23      <button (click)="enviarDatosHaciaAppComponentAlGenerarEvento()">
24        DATOS DEL HIJO, RECIBIDOS Y ESCRITOS EN EL COMP. PADRE.
25      </button>
26
27   </td>
28   </tr>
29 </table>
30

```

Fig. 4.2.50: (a) Comunicación de datos DESCENDENTE en el HTML del comp. hijo, en la línea 21 remarcada en color verde. (b) Comunicación de datos ASCENDENTE en el HTML del comp. hijo, en las líneas 23-26 remarcadas en color rojo.

Codificación para el envío DESCENDENTE de datos (del comp. Padre al comp. Hijo):

Paso 1: Controlador Typescript del componente padre (App).

Se crea una variable de instancia, de tipo String, en el controlador del componente “app”. Se le asignará un tipado String y un valor inicial.

La variable de instancia se llamará “datosEnviadosAlHijoCompApp”.

Más info.: ver Fig. 4.1.47, sección verde.

Paso 2: Vista HTML del componente padre (App).

Ir a la vista HTML del componente padre “App”.

En la línea de código donde se llama al selector HTML del comp. hijo dentro de la propia etiqueta del selector, añadir el siguiente atributo:

`[datosRecibidosEnCompHijo] = "datosEnviadosAlHijoCompApp"`

Aclaraciones:

- A la izquierda → nombre de la variable colocada en el controller hijo, donde guardamos los datos que se reciben en el componente hijo.
- A la derecha → nombre de la variable colocada en el controller padre, que usamos para enviar los datos del componente hijo.

Más info.: ver Fig. 4.2.48, sección verde.

Paso 3: Typescript del componente hijo.

Ir al controlador del componente hijo y crear una variable a nivel de clase con la directiva @Input. En esta variable recibiremos los datos del componente padre.

Para ello, poner el siguiente código en el controlador hijo:

`@Input() datosRecibidosEnElHijo : any;`

Más info.: ver Fig. 4.2.49, sección verde.

Paso 4: Vista HTML del componente hijo.

Podremos pintar ahora la nueva variable del controlador de componente hijo en la vista del mismo.

Para ello en la vista HTML “primer-componente-formulario-login.component.html” debemos pintar el valor de la variable “`datosRecibidosEnElHijo`”.

El valor de la variable receptora de los datos puede rescatarse en la vista del componente hijo colocando el nombre de la variable receptora entre dobles llaves, tal que así:

```
<p> {{datosRecibidosEnElHijo}} </p>
```

Más info.: ver Fig. 4.2.50, sección verde.

Resultado final:

Se pintan los datos enviados de la componente padre del componente hijo, dentro de la vista del componente hijo (ver Fig. 4.2.51).

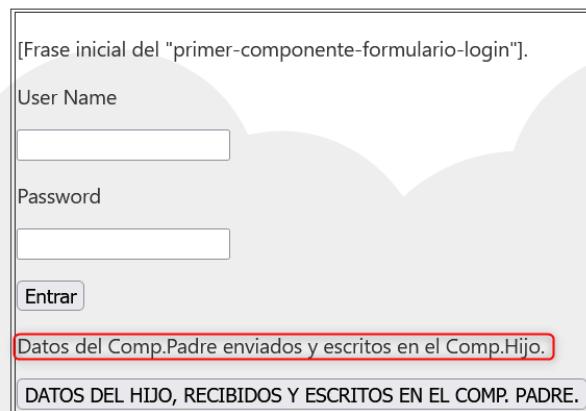


Fig. 4.2.51: Visualización de datos pintados por comp. hijo, enviados desde comp. padre.

Codificación para el envío ASCENDENTE de datos (del comp. Hijo al comp. Padre):

Paso 1: Controlador Typescript del componente padre (App).

Se crea una variable de clase, de tipo String, en el controlador del componente “app”. Se le asignará un tipado String, pero sin inicializarse.

La variable de instancia se llamará “datosRecibidosDelHijoCompApp”.

Más info.: ver Fig. 4.2.47, sección roja.

Paso 2: Vista HTML del componente padre (App).

Ir a la vista HTML del componente padre “App”.

En la línea de código donde se llama al selector HTML del comp. hijo, dentro de la propia etiqueta del selector, añadir el siguiente atributo:

```
[datosEnviadosPorCompHijo] = "datosRecibidosDelHijoCompApp"
```

Aclaraciones:

- A la izquierda → nombre de la variable colocada en el controller hijo, inicializada con los datos que se envían desde el componente hijo.
- A la derecha → nombre de la variable colocada en el controller padre, que usamos para recibir los datos del componente hijo.
- A la derecha se le asocia también una etiqueta de “acción” para asociarlo a un “evento”, mediante la palabra reservada “envent”. El evento será generado en el controller hijo.

Más info.: ver Fig. 4.2.48, sección roja.

Paso 3: Typescript del componente hijo.

Ir al controlador del componente hijo, y crear una variable a nivel de clase con la directiva @Output. En esta variable mandaremos los datos al componente padre y generaremos un evento. Para ello, instanciar la variable creando un objeto de tipo “EventEmitter” al momento de su creación.

A continuación, crear un método que en su interior emita un evento a través de la variable de clase, y ese evento genere un mensaje de tipo String.

Para ello, poner el siguiente código en el controlador hijo:

```
@Output() datosEnviadosPorCompHijo = new EventEmitter();  
enviarDatosHaciaAppComponentAlGenerarEvento(){  
    this.datosEnviadosPorCompHijo.emit('DATOS DEL HIJO,  
    RECIBIDOS Y ESCRITOS EN EL COMP. PADRE.');//  
}
```

Más info.: ver Fig. 4.2.49, sección roja.

Paso 4: Vista HTML del componente hijo.

En la vista HTML del comp. hijo se le incluye un elemento de tipo “botón”. El botón llama a una función de código del controlador hijo cuando se hace “click” sobre este elemento botón.

En este caso no se pintar la nueva variable del controlador de componente hijo en la vista del mismo. Se pintará en el componente padre, que se hizo en el paso 2.

El código que debemos usar para añadir el nuevo botón mediante HTML, es el siguiente:

```

<!-- Envio de datos con @Output: desde componente hijo a
componente padre, con evento click -->

<button
(click)="enviarDatosHaciaAppComponentAlGenerarEvento()">

DATOS DEL HIJO, RECIBIDOS Y ESCRITOS EN EL
COMP. PADRE.

</button>

```

Más info.: ver Fig. 4.2.50, sección roja.

Resultado final:

Inicialmente veíamos algo como lo que tenemos en la Fig. 4.1.51 dentro del borde del formulario, límite del elemento hijo.

Al clicar en el segundo botón del formulario (que llamamos “DATOS DEL HIJO, RECIBIDOS Y ESCRITOS EN EL COMP. PADRE.”) se generará un mensaje idéntico fuera del elemento hijo (fuera del cuadro del formulario).

Cuando el texto del botón se escriba fuera del formulario, entonces sabremos que se ha pintado el mensaje en el elemento padre: eso significará que los datos han sido enviados por el Comp. Hijo, recibidos por el Comp. Padre y escritos en el Comp. Padre. Véase Fig. 4.1.52.

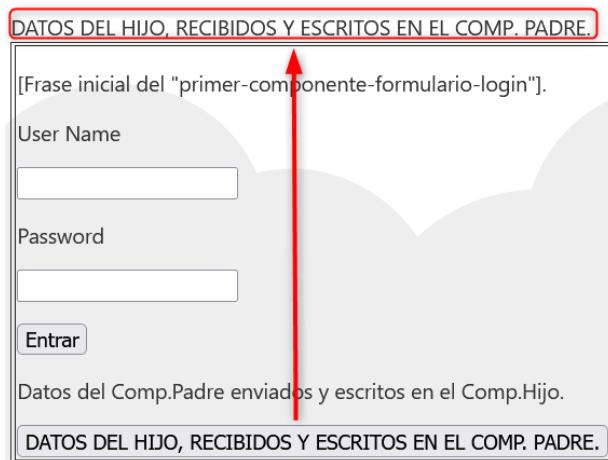


Fig. 4.2.52: Visualización de datos pintados por comp. padre, enviados desde comp. hijo.

i) Directivas para condicionales y bucles: if, switch, for.

Las directivas condicionales empiezan siempre por “ng” y le sigue el nombre del tipo de bucle. De este modo los condicionales quedarían así: ngIf, ngSwitch y ngFor.

Gracias a estas directivas podremos controlar qué añadimos y qué quitamos del árbol DOM, en función de la condición que cumpla nuestro código (generalmente en el código del controlador TypeScript).

- Directiva ngIf:

Esta directiva elimina un elemento del DOM si la expresión es falsa, o si la expresión es verdadera agregará el elemento al DOM.

Para nuestro primer ejemplo, crear una variable booleana en el controlador del componente hijo. La nueva variable la llamaremos “**variableBooleanaNgIf**”, y esta expresión podrá tener el valor “true” o “false” de tipo booleano.

Después, en la vista HTML del componente hijo incluiremos 3 bloques de código:

- Un elemento “**ng-template**” para cuando el resultado del condicional sea “true”. El nombre de este primer ng-template será añadido como atributo de la etiqueta, poniendo “#**formularioNgIfTrue**”.
- Otro elemento “**ng-template**” para cuando el resultado del condicional sea “false”. El nombre de este segundo ng-template será añadido como atributo de la etiqueta, poniendo “#**formularioNgIfFalse**”.
- Añadimos el elemento HTML de tipo DIV que ejecute la directiva ngIf. Para ejecutar esta directiva, poner la palabra reservada precedida por asterisco (*ngIf) como un atributo de la etiqueta div. Seguido de directiva va una asignación con un “=” y un string compuesto de 3 partes:
 - [NombreVariableBooleanaDelControlador];
 - then [NombreNgTemplatePrimera];
 - else [NombreNgTemplateSegunda]

Esta explicación teórica queda visualmente explicada en las imágenes de las Fig. 4.1.53 y Fig. 4.1.54.

- Directiva ngSwitch:

Esta directiva funciona como un bucle Switch-Case usado en Java, el cual puede servirnos para seleccionar una opción entre varias disponibles, y mostrar la opción seleccionada en el árbol DOM.

Para el 2º ejemplo, crear una variable tipo String en el controlador del componente hijo. La nueva variable la llamaremos “**variableStringNgSwitch**”. La nueva variable podrá tener infinitos valores de tipo String, pero solo 2 valores serán controlados: OK y FAIL. Cuando caiga en un valor no controlado se ejecutará el resultado de la opción 3, usada para casos default.

En la vista HTML del componente hijo incluiremos, en un único bloque de código, toda la lógica de programación necesaria para el bucle Switch (y todas las posibles evaluaciones de cada caso).

El seudo-código teórico necesario para el bloque switch será este:

```
<div [ngSwitch]=“nombreVariableEvaluadaAnhadidaEnControlador”>  
    <div *ngSwitchCase=“‘OK’”>     Frase 1           </div>  
    <div *ngSwitchCase=“‘FAIL’”>   Frase 2           </div>  
    .....  
    <div *ngSwitchDefault>          Frase por defecto </div>  
</div>
```

Esta explicación teórica queda visualmente explicada en las imágenes de las Fig. 4.2.53 y Fig. 4.2.54.

- Directiva ngFor:

Esta directiva funciona como un bucle for each de Java, y puede servirnos para listar elementos contenidos en un Array o cualquier elemento de tipo Collection. Conforme se vaya extrayendo cada elemento del Array podemos ir pintándolo en el árbol DOM.

Para el 3º ejemplo, crear una variable tipo Array de elementos String, en el controlador del componente hijo. La nueva variable la llamaremos “**colecciónNombresNgFor**”. Esta variable será inicializada con 3 posiciones de array, de tipo String todas ellas.

La nueva variable se usará en la vista HTML del componente hijo, y para dar uso a esta variable se usará la directiva “ngFor” de la siguiente manera:

```

<div *ngFor="let elemento of colecciónNombresNgFor">
    {{elemento}}
</div>

```

Esta explicación teórica queda visualmente explicada en las imágenes de las Fig. 4.2.53 y Fig. 4.2.54. Ejemplo de resultados visualizado en Fig. 4.1.55.

```

27 | // variable para directivas de bucles condicionales
28 | variableBooleanNgIf : boolean= false;
29 | variableStringNgSwitch : string = "FAIL";
30 | colecciónNombresNgFor :string[]=[ 'Juan', 'Pedro', 'Jose'];
31 |

```

Fig. 4.2.53: Variables creadas a nivel de clase en el controlador TypeScript del componente hijo. Tres nuevas variables, una para cada tipo de bucle If, Switch y For. (a) Línea 28 en verde, para la creación de la variable de la directiva *ngIf. (b) Línea 29 en rojo, para la creación de la variable de la directiva *ngSwitch. (c) Línea 30 en azul, para la creación de la variable de la directiva *ngFor.

```

28 | </tr>
29 | <br>
30 |
31 | <!-- Código de directiva ngIf -->
32 | <div *ngIf="variableBooleanNgIf; then formularioNgIfTrue; else formularioNgIfFalse">
33 | </div>
34 | <ng-template #formularioNgIfTrue>
35 |   <div> La directiva ngIf es true.</div>
36 | </ng-template>
37 | <ng-template #formularioNgIfFalse>
38 |   <div> La directiva ngIf es false.</div>
39 | </ng-template>
40 |
41 | <!-- Código de directiva ngSwitch -->
42 | <div [ngSwitch]="variableStringNgSwitch">
43 |   <div *ngSwitchCase="OK"> Switch Case tipo OK seleccionado.</div>
44 |   <div *ngSwitchCase="FAIL"> Switch Case tipo FAIL seleccionado.</div>
45 |   <div *ngSwitchDefault> Switch Case ROTO.</div>
46 | </div>
47 |
48 | <!-- Código de directiva ngFor -->
49 | Colección de nombres ejecutada con directiva ngFor:
50 | <div *ngFor="let nombre of colecciónNombresNgFor">
51 |   {{nombre}}
52 | </div>
53 |
54 | </table>

```

Fig. 4.2.54: Código de la vista HTML del comp. hijo, desde donde ejecutamos las directivas ngIf, ngSwitch y ngFor. (a) Primer bloque de código en recuadro verde, para directiva *ngIf. (b) Segundo bloque de código en recuadro rojo, para directiva *ngSwitch. (c) Tercer bloque de código en recuadro azul, para directiva *ngFor.

[Frase inicial del "primer-componente-formulario-login"].

User Name

Password

Entrar

Datos del Comp.Padre enviados y escritos en el Comp.Hijo.

DATOS DEL HIJO, RECIBIDOS Y ESCRITOS EN EL COMP. PADRE.

La directiva ngIf es false.
 Switch Case tipo FAIL seleccionado.
 Colección de nombres ejecutada con directiva ngFor:
 Juan
 Pedro
 Jose

 A screenshot of an Angular application's login form. At the top, there are input fields for 'User Name' and 'Password'. Below them is a button labeled 'Entrar'. Underneath the button, a message says 'Datos del Comp.Padre enviados y escritos en el Comp.Hijo.' followed by a bolded section 'DATOS DEL HIJO, RECIBIDOS Y ESCRITOS EN EL COMP. PADRE.'. Three pieces of text are highlighted with colored boxes: a green box contains 'La directiva ngIf es false.', a red box contains 'Switch Case tipo FAIL seleccionado.', and a blue box contains 'Colección de nombres ejecutada con directiva ngFor:' followed by a list of names: Juan, Pedro, and Jose.

Fig. 4.2.55: Resultado visual de la ejecución de cada una de las 3 directivas *ngIf (recuadro verde), *ngSwitch (recuadro rojo) y *ngFor (recuadro azul).

j) Interpolación de String y Databinding en Angular

La interpolación de String es un mecanismo de programación, el cual permite una sustitución de “una expresión por el valor de dicha expresión”, en un template.

La interpolación simple (o bidireccional) de String en Angular se consigue con la doble llave: `{ { variable } }`.

Podría conseguirse, al menos, 4 tipos de formas de comunicación:

- Binding simple
- Property Binding
- Variables de referencia del template
- Doble Databinding

En resumen, la ventaja que tiene la interpolación en Angular es que es permite cambios dinámicos: al cambiar el valor de la propiedad o expresión concreta, Angular modificar el resultado que sale en el DOM del HTML automáticamente.

Tipo 1: INTERPOLACIÓN SIMPLE

Aquí se presenta la primera funcionalidad relacionada con el Binding: la interpolación simple. Esta funcionalidad puede consistir, por ejemplo, en lograr que alguna parte del front-end se modifique automáticamente cuando accionemos un botón (o se genere una acción).

¿Cuál podría ser el pseudo-código para conseguir la interpolación simple:

1º) En el Controller del componente hijo:

Crear una variable nueva de tipo String, y se inicializa con una frase de nuestra elección. La variable tendrá un alcance a nivel de clase.

```
statusDeEnvioFormulario : String = "no se envió el formulario;
```

2º) En el Controller del componente hijo:

Crear una nueva función de código. Cada vez que llamemos a la nueva función, esta modificará el valor de la variable.

```
fActualizarInfoEstadoEnvio(){  
    this.statusDeEnvioFormulario = "el formulario SÍ se  
    envió";  
}
```

3º) En la vista HTML del comp. hijo:

Crear un formulario similar al anterior, pero solo con los campos “User Name”, “Password”.

Debajo llevará un botón que apuntará ahora a otra función: la nueva función del paso 2.

Debajo del botón rescatará el valor de la nueva variable del paso 1, que contendrá una frase inicialmente mostrando que el formulario no se ha enviado. Cuando se pulse el botón, como está apuntando a la nueva función, en la nueva función cambiaremos el valor de la frase a algo que diga que ya se envió el formulario.

```
<p style="padding-right: 2em;">
```

Formulario para Interpolación y Binding

```
</p>
```

```
<p>
```

User Name

```

</p>

<input type="text" name="userNume">

<p>
    Password
</p>

<input type="text" name="password">

<p>
    <button (click)="fActualizarInfoEstadoEnvio()">
        Entrar
    </button>
    <br>
    {{ statusDeEnvioFormulario }}
</p>

```

Puedes ver la frase inicial antes de enviar el formulario en la imagen Fig. 4.2.56. Después de enviar el formulario pulsando el botón “Entrar” la frase inferior cambiará automáticamente, como podrás ver en la segunda imagen “Fig. 4.2.57”.

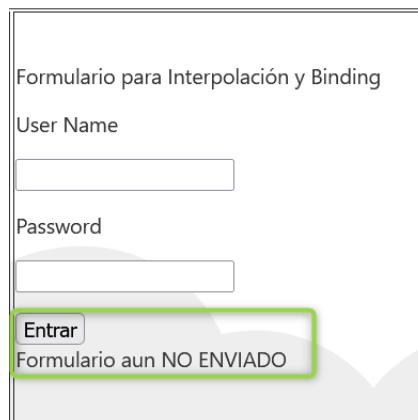


Fig. 4.2.56: Frase inicial antes de enviar el formulario.

Fig. 4.2.57: Frase modificada automáticamente después de pulsar el botón “Entrar”.

Tipo 2: PROPERTY BINDING

La segunda funcionalidad relacionada con el Binding, es también un mecanismo de una única dirección (como la anterior). Sirve para actualizar el valor de una propiedad del formulario (del componente), y enlazarlo con un elemento en la vista HTML.

En nuestro formulario teníamos 2 campos input: el de username y el de password. Cada uno de estos campos de entrada ya tenía las propiedades de “type” para texto y la de “name” correspondiente.

Ahora incluiremos a estos 2 campos de entrada de texto una nueva propiedad: [value].

La nueva propiedad será la de asignar el valor del campo de texto y debe ir siempre entre corchetes. A esta propiedad llamada “[value]” se le asignará un valor de tipo String que debe ser el nombre de una variable creada en el controller.

Ejemplo:

- 1º) En el controller hijo: crear una variable, llamada “userNameDefault”, inicializada a cadena vacía.
- 2º) En el controller hijo: crear otra variable, llamada “passwordDefault”, inicializada a cadena vacía.
- 3º) En el controller hijo: crear una función que al llamarla asigne un valor a ambas variables.
- 4º) En la vista HTML del hijo: añadir a cada campo de entrada de texto que tenemos en la vista, la propiedad [value]= “nombreVariableValorDefault”, donde nombreVariableValorDefault será el nombre de la variable que guardó el valor default del campo concreto.
- 5º) Creación del botón en el formulario, y en la propiedad (click) asignaremos el nombre de la función creada en el paso 3.

El resultado en código del ejemplo anterior lo podemos ver en las imágenes Fig. 4.2.58 y Fig. 4.2.59.

La funcionalidad visual se puede ver en las imágenes Fig. 4.2.60 (formulario inicialmente vacío) y Fig. 4.2.61 (formulario lleno con valores Default al clicar en el 2º botón).

```
42 // Parte 2: property Binding
43 userNameDefault: String = "";
44 passwordDefault: String = "";
45 fDefaultForm(){
46     this.userNameDefault = "Juan";
47     this.passwordDefault = "1234";
48 }
```

Fig. 4.2. 58: Código del archivo del controller. Puede verse la creación de 2 variables (una para cada campo de texto del formulario) y una nueva función que asigna valores default a las variables).

```
<!-- Formulario para las pruebas de la interpolación y el Binding -->
<td>
    <p style="padding-right: 2em;">
        Formulario para Interpolación y Binding
    </p>
    <p>
        User Name
    </p>
    <input type="text" name="userName2" [value]="userNameDefault"[value]="passwordDefault"
```

Fig. 4.2.59: Código del archivo de la vista HTML. En verde rodeadas las propiedades [value] y el nuevo botón. En rojo, rodeadas las asignaciones del nombre de variables y la función creada en el controller.

Formulario para Interpolación y Binding

User Name

Password

Entrar

Formulario aun NO ENVIADO

Poner valores Default en formulario

Fig. 4.2.60: Campos inicialmente vacíos al inicializar el formulario.

Formulario para Interpolación y Binding

User Name

Password

Entrar

Formulario aun NO ENVIADO

Poner valores Default en formulario

Fig. 4.2.61: Campos con valores default, colocados automáticamente al clicar botón inferior.

Tipo 3: TEMPLATE REFERENCE VARIABLE

Esta tercera funcionalidad sirve para recuperar los datos de entrada en los campos de un formulario, antes de enviarse estos datos o durante el envío de ellos.

Para hacer uso de esta funcionalidad, añade a la etiqueta del campo input correspondiente del formulario lo siguiente: “#”+ “nombreDeseado” (como un atributo más del elemento “input”). Después podremos hacer referencia a este nombre en el código HTML, como si de una variable se tratara, para rescatar los datos escritos en el campo input, procesarlos o realizar cualquier otra cosa.

Haciendo uso del ejemplo del formulario que teníamos, se realizaría el siguiente pseudo-código:

1º) En la vista HTML del comp. hijo:

Añado “#recogidaUserName” a los atributos de la etiqueta input del “userName”.

A partir de aquí Angular reconocerá el nombre “recogidaUserName” como una variable más del controller (pero que se creó en el HTML). El valor que tendrá esta variable será el texto que contenga (en ese instante) este input text.

2º) En la vista HTML del comp. hijo:

Añadir el valor de la variable, como variable de entrada de la función llamada al clicar el botón “Entrar” del formulario.

Para indicar que entre solo el valor del campo debo poner como variable de entrada esto: “recogidaUserName.value”.

3º) En el controller del comp. hijo:

Se crea una variable de entrada en la interfaz de la función que llamo cada vez que clico en el botón “Entrar” del formulario.

Dentro de esta función voy a usar la variable de entrada para imprimirla por consola con instrucción “console.log(variableEntrante)”.

Un ejemplo visual de código puede verse en las Fig. 4.2.62 y Fig. 4.2.63.

```
<!-- Formulario para las pruebas de la interpolación y el Binding -->
<td>
    <p style="padding-right: 2em;">
        Formulario para Interpolación y Binding
    </p>
    <p>
        User Name
    </p>
    <input type="text" #recogidaUserName name="userName2" [value]="userNameDefault">
    <p>
        Password
    </p>
    <input type="text" #recogidaPassword name="password2" [value]="passwordDefault">
    <p>
        <button (click)="fActualizarInfoEstadoEnvio(recogidaUserName.value)">
            Entrar
        </button>
        <br>
        {{statusDeEnvioFormulario}}
        <br><br>
        <button (click)="fDefaultForm()">
            Poner valores Default en formulario
        </button>
    </p>
</td>
```

Fig. 4.2.62: Modificaciones en vista HTML para crear variable de referencia, recoger los datos del input text y meterlos como variable de entrada a la función llamada.

```

// Parte 1: interpolacion simple
statusDeEnvioFormulario : string = "Formulario aun NO ENVIADO"
fActualizarInfoEstadoEnvio(datoEntrante: string){
    console.log(datoEntrante);
    this.statusDeEnvioFormulario = "Formulario enviado EXITOSAMENTE";
}

// Parte 2: property Binding
userNameDefault: String = "";
passwordDefault: String = "";
fDefaultForm(){
    this.userNameDefault = "Juan";
    this.passwordDefault = "1234";
}

```

Fig. 4.2.63: Modificaciones en Controller de la función que enviaría el formulario web, para que ahora muestre por consola los datos recogidos antes de mandarlos. Mostramos resultado por consola.

Tipo 4: DOBLE DATABINDING

La cuarta funcionalidad relacionada con Binding es la “Doble Databinding”. Es una mezcla entre la primera funcionalidad y las funcionalidad 2 y 3.

Permite “visualizar en la web el texto que vamos escribiendo en un campo de datos de un formulario (funcionalidad 1), y el texto visual se actualice en tiempo real según se modifique el campo input text (combinación de funcionalidades 2 y 3)”.

Para hacer uso del “doble databinding” necesitamos modificar 3 archivos:

- El fichero de configuración del módulo, del componente App (comp. padre).
- El fichero del controller del componente hijo.
- El fichero de la vista HTML del componente hijo, para crear el código del “Simple Binding” y el código del “Doble Databinding” (usando directiva “NgModulo”).

El pseudo-código sería el siguiente:

- 1) Fichero config. de modulo: añadir en sección de “import” la librería “FormsModule” de “@angular/forms” y colocar también en el array de import dicha librería (FormsModule).
- 2) Controller: creando únicamente la nueva variable que usaremos en la vista. La variable se inicializará como String vacío. Ejemplo: **miVariableDelController: String = ""**.
- 3) La vista HTML: en un elemento de tipo “entrada de texto” de nuestro formulario, colocar la nueva directiva ngModel y La asignación con el nombre de la variable creada en el controller; como nuevo atributo del elemento HTML, entre corchetes y paréntesis **<input [(ngModel)]="miVariableDelController">**. Fuera del elemento HTML “input”, crear una binding simple que recoja el valor: **{ {"miVariableDelController"} }**

El código necesario para conseguir esta nueva funcionalidad, de Doble Databinding, se puede visualizar en las imágenes Fig. 4.2.64 (Controller), 4.2.65 (fichero de configuración del módulo) y 4.2.66 (vista HTML).

```

43 // Parte 2: property Binding
44 userNameDefault: String = "";
45 passwordDefault: String = "";
46 fDefaultForm(){
47     this.userNameDefault = "Juan";
48     this.passwordDefault = "1234";
49 }
50
51 // Parte 4: doble databinding
52 salidaDobleDatabingFieldUsername : String = "";
53
54 }

```

Fig. 4.2.64: Creación de la nueva variable en el Controller, inicialmente con valor “cadena vacía” (para el doble databinding). El fichero del controller es “primer-componente-formulario-login.component.ts”.

```

ts primer-componente-formulario-login.component.ts U ✘ primer-componente-formulario-login.component.html U ts app.module.ts M ✘
appFrontend > src > app > ts app.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5 import { PrimerComponenteFormularioLoginComponent } from './primer-componente-formulario-login/primer-com
6
7 import { FormsModule } from "@angular/forms"
8
9 @NgModule({
10   declarations: [
11     AppComponent,
12     PrimerComponenteFormularioLoginComponent
13   ],
14   imports: [
15     BrowserModule,
16     FormsModule
17   ],
18   providers: [],
19   bootstrap: [AppComponent]
20 })
21 export class AppModule { }

```

Fig. 4.2.65: Importación de la nueva librería “FormsModule” en el módulo principal de la aplicación (módulo de App). El fichero de configuración del módulo debe ser “app.module.ts”.

```

29 <!-- Formulario para las pruebas de la interpolación y el Binding -->
30 td>
31 <p style="padding-right: 2em;">
32 | Formulario para Interpolación y Binding
33 </p>
34 <p>
35 | User Name
36 </p>
37 <input type="text" [(ngModel)]="salidaDobleDatabingFieldUsername" #recogidaUserName name="userName2" [value]="userNameDefault">
38 <p>
39 | Password
40 </p>
41 <input type="text" #recogidaPassword name="password2" [value]="passwordDefault">
42 <p>
43 | <button (click)="fActualizarInfoEstadoEnvio(recogidaUserName.value)">
44 | Entrar
45 </button>
46 <br>
47 {{statusDeEnvioFormulario}}
48 <br><br>
49 <button (click)="fDefaultForm()">
50 | Poner valores Default en formulario
51 </button>
52
53 <!-- Parte de la salida del doble Databinding -->
54 <br><br>
55 El texto de la input text del USER NAME es: <br>
56 {{salidaDobleDatabingFieldUsername}}
57

```

Fig. 4.2.66: modificación del código necesario para usar el Doble Databinding en el archivo de Vista HTML. El fichero de la vista es “primer-componente-formulario-login.component.html”.

k) Los servicios en Angular

Un servicio en Angular (frontend) será como un servicio de Spring (backend), pero mientras que en un servicio de Spring (Web Service) puede ser “conectar con una Base de Datos”, un servicio en Angular podría ser “pedir datos al Web Service de Spring”.

Desde el lado de infraestructura del software, cada servicio en Angular está conformado por una clase de código Typescript, que hará una cosa y la hará bien.

A la hora de crear un nuevo servicio en Angular y crear el nuevo archivo Typescript, dicho archivo puede crearse por línea de comandos: en la consola del propio IDE Visual Studio Code crearemos los archivos necesarios haciendo uso de comando de la librería “@angular/cli”.

El comando que nos permite crear los archivos del nuevo servicio automáticamente (llamado “NOMBRE_NewService”), será el siguiente:

“ng generate service NOMBRE_NewService”.

Como resultado, se habrá creado 2 archivos nuevos en nuestro directorio (dentro del componente raíz “App”). Esto implicará que el servicio ha sido creado. Los 2 archivos nuevos podrían ser los siguientes:

- NOMBRE_NewService.service.ts
- NOMBRE_NewService.service.spec.ts

Algunos propósitos que desempeñan los servicios son los siguientes:

- contener la lógica de negocio
- capa de acceso a datos
- acceso y conexión a web services (del backend)

Otro concepto que viene asociado a los servicios en Angular es el de “inyección de dependencias”. La inyección de dependencias tiene 2 aspectos principales:

- Es un patrón de diseño que crea código modular. Nos desacopla el código.
- La clase que use los objetos externos no creará las instancias de éstos, sino que se les suministrará los objetos desde el exterior (se le inyecta la implementación deseada de la clase externa).

Beneficio adicional:

- La inyección de dependencias aporta simplicidad en los test unitarios.

A continuación, se presentan los 4 pasos necesarios para crear un pseudo-código que nos permita usar un nuevo servicio en Angular:

- 1) Crear el nuevo servicio de Angular mediante línea de comandos:
 - a. Usar el comando: “`ng generate service NOMBRE`”.
 - b. Comprobar que se han creado los 2 nuevos ficheros .ts y .spec.ts en nuestro directorio de trabajo (dentro del directorio `./app/`):
 1. `NOMBRE.service.spec.ts`
 2. `NOMBRE.service.ts`.
 - c. Visualizar el ejemplo realizado sobre el propio código dirigiéndose a las imágenes **Fig. 4.2.67** y **Fig. 4.2.68**.
 - d. Abrir el 2º ficheros (con formato “`NOMBRE.service.ts`”) y copiar el nombre de la clase que contenga ya creada en su interior (que será el nombre del archivo + “Service” = “`NOMBREService`”).
- 2) Configurar los Providers dentro del archivo de configuración del módulo (“`app.module.ts`”).
 - a. Pegar el nombre de la clase del 2º fichero que conforma el nuevo servicio, dentro del array de “providers”.
 - b. Si no se ha incluido como nuevo “import”, añadir el import correspondiente.
 - c. Ver ejemplo del código añadido en el fichero de config. del módulo
Por ejemplo, para un nombre de clase “**NombreService**” del archivo **`NOMBRE.service.ts`** que esté dentro de componente raíz “`./app/`” generar la siguiente línea de import:
 - `import { NombreService } from './NOMBRE.service'`
- Ver imagen de la **Fig. 4.2.69** para ver un ejemplo del código, en el archivo de configuración del componente.
- 3) Ir al fichero de nuestro nuevo servicio (`NOMBRE.service.ts`), donde tendremos una sección de “inyectables” y la nueva clase (`NombreService`) con su constructor no parametrizado ya incluido.
 - Añadir un método que devuelva un array de elementos de la Base de Datos (simular el array de la BBDD manualmente).
 - Con esto ya tenemos terminado nuestro servicio (solo contendría getter y setter).
- Ver la imagen **Fig. 4.2.70** para ver el ejemplo del código incluido en el fichero del nuevo servicio.
- 4) Ir al controlador TypeScript del componente que usará el nuevo servicio (`primer-componente-formulario-login.component.ts`).
Inyectar el servicio nuevo en el controller del componente hijo. Para ello se debe crear el parámetro de entrada al constructor de la clase del componente

como variable privada y tipo igual al nombre de la clase del servicio. Luego usar esa variable de entrada en el propio constructor o en el método OnInit().

- a. No crear la variable de tipo de objeto idéntico al nombre de la clase del servicio.
 - b. Crear un variable de tipo array de elementos “any”, a nivel de clase. Inicializarla también como array vacío.
 - c. En su lugar, inyectar el servicio dentro del componente a través del constructor de la clase del controller del comp. hijo (como parámetro de entrada del constructor, como variable “privated”).
 - d. Usar la variable privada que se le introduce al constructor: usarla en el mismo constructor o en la función OnInit().
 - e. Para usar la variable de clase que iniciamos como array vacío se puede crear de 2 formas:
 1. Dentro del constructor:

2. Dentro del método OnInit():

```
this.variableAnivelDeclase = parametroEntradaConstructorPrivated.getAllUser();  
console.log('this.variableAnivelDeclase');
```

2. Dentro del método OnInit():

```
this.variableAnivelDeclase = this.parametroEntradaConstructorPrivated.getAllUser();  
  
console.log('this.variableAnivelDeclase');
```

Ver la imagen **Fig. 4.2.71** para ver el ejemplo del código modificado del componente hijo que deseemos usar.

Puede encontrarse las pruebas de funcionamiento en la última figura del apartado, la **Fig. 4.2.72**.

Por último, todas las imágenes de los ejemplos de códigos realizados en este apartado de servicios en Angular:

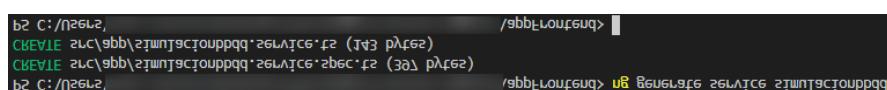


Fig. 4.2.67: Creación del nuevo servicio a través de comandos, con “@angular/cli”.

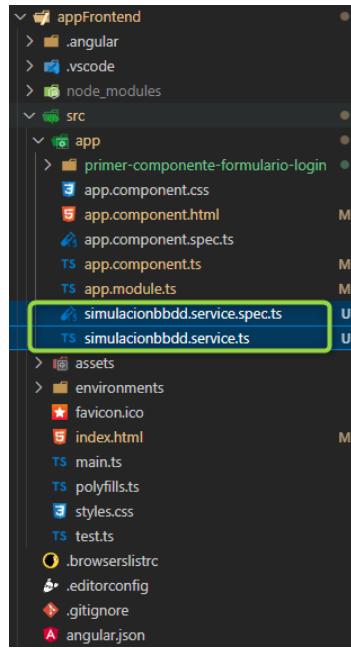


Fig. 4.2.68: Árbol del directorio del proyecto front-end, con los nuevos archivos creados del nuevo servicio.

```
TS app.module.ts M X
appFrontend > src > app > TS app.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5 import { PrimerComponenteFormularioLoginComponent } from './primer-componente-formulario-login';
6
7 import { FormsModule } from '@angular/forms';
8 import { SimulacionbddService } from './simulacionbdd.service';
9
10 @NgModule({
11   declarations: [
12     AppComponent,
13     PrimerComponenteFormularioLoginComponent
14   ],
15   imports: [
16     BrowserModule,
17     FormsModule
18   ],
19   providers: [SimulacionbddService],
20   bootstrap: [AppComponent]
21 })
22 export class AppModule { }
```

Fig. 4.2.69: Configuración del proveedor de dependencias de Angular, para configurar el inyector de dependencias.

```

ts simulacionbdd.service.ts U X
appFrontend > src > app > TS simulacionbdd.service.ts ...
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class SimulacionbddService {
7
8   constructor() { }
9
10  getAllUsersFromDB(){
11    return [
12      {'id':0, 'name':'Pedro', 'fechaInscripcion':'02-01-2022'},
13      {'id':1, 'name':'Paco', 'fechaInscripcion':'15-07-2022'},
14      {'id':2, 'name':'Pepe', 'fechaInscripcion':'20-12-2022'}
15    ];
16  }
17}
18

```

Fig. 4.2.70: Creamos métodos getter y setter del nuevo servicio, en el fichero del propio service.

```

ts primer-componente-formulario-login.component.ts U X
appFrontend > src > app > primer-componente-formulario-login > TS primer-componente-formulario-login.component.ts ...
1 import { Component, Input, OnInit, Output, EventEmitter } from '@angular/core';
2 import { SimulacionbddService } from '../simulacionbdd.service';
3
4 @Component({
5   selector: 'app-primer-componente-formulario-login',
6   templateUrl: './primer-componente-formulario-login.component.html',
7   styleUrls: ['./primer-componente-formulario-login.component.css']
8 })
9 export class PrimerComponenteFormularioLoginComponent implements OnInit {
10
11   @Input() datosRecibidosEnCompHijo : any;
12
13   usuariosConseguidos : any = [];
14
15   constructor(private miNuevoServicio: SimulacionbddService) { }
16
17   ngOnInit(): void {
18     //console.log('OnInit() del comp. hijo ejecutado.')
19     this.usuariosConseguidos = this.miNuevoServicio.getAllUsersFromDB();
20     console.log(this.usuariosConseguidos);
21   }
22
23 }
24
25

```

Fig. 4.2.71: Uso del nuevo servicio, en el controller del componente que use el componente.

```

4) ngDoCheck
5) ngAfterContentInit
6) ngAfterContentChecked
7) ngOnInit() del comp. hijo ejecutado.
> Array(3) [ { }, { }, { } ]
> 0: Object { id: 0, name: "Pedro", fechaInscripcion: "02-01-2022" }
> 1: Object { id: 1, name: "Paco", fechaInscripcion: "15-07-2022" }
> 2: Object { id: 2, name: "Pepe", fechaInscripcion: "20-12-2022" }
> length: 3
> <prototype>: Array []
7) ngAfterViewInit
8) ngAfterViewChecked
Angular is running in development mode. Call enableProdMode() to enable production mode.

```

Fig. 4.2.72: Prueba de funcionamiento del código realizado, visualizando en la consola del navegador el array de los 3 usuarios que hemos simulado traer de la BBDD.

I) Llamadas HTTP y gestión de errores HTTP

En esta sección se presenta un ejemplo de llamada HTTP de tipo GET, la gestión o manejo de errores en peticiones GET y las pruebas de la funcionalidad conseguida sobre el código realizado.

Conceptos básicos de HTTP:

- Debemos realizar llamadas HTTP si queremos escribir o leer una API REST.
- Una API REST es un conjunto de EndPoint's que proporciona un servidor backend.

Conceptos básicos sobre peticiones HTTP con Angular:

- Para realizar la llamada HTTP en Angular necesitamos importar algunas clases en el “app.module.ts” y en el “simulacionbbdd.service.ts”.
- La llamada HTTP de tipo GET se va a construir dentro del Service, archivo llamado “simulacionbbdd.service.ts”.
- La llamada HTTP para la petición GET creada en el servicio solo devolverá un “Observable”, no la respuesta la petición GET.
- La respuesta de la petición GET podremos recibirla después de subscribirnos al Observable que devuelve la petición GET HTTP.
- ¿Qué es esto de un Observable y para qué nos subscribimos a él?
 - o Un Observable es un “patrón reactivo” y programación asíncrona en Angular.
 - o Un Observable puede ser creado por una llamada HTTP, y cada Observable viene con 1 o varios observer's asociados a él.
 - o El Observable emite los datos de la petición GET y los observer's observan o leen los datos que emite su Observable.
- En la propia llamada HTTP que devuelve el Observable se indican los datos que esperamos obtener de la petición GET (en el observador), cuando un observador se subscriba al Observable y este Observable después emita datos. El tipo de datos que esperamos recibir, en el observador que se subscribe al Observable, será indicado entre llaves de “<” y “>” en la propia llamada GET HTTP.
- Después de implementar la llamada HTTP GET que devuelve un Observable en el servicio, podemos subscribirnos al Observable devuelto en el “controller typescript del componente” que deseemos que reciba los datos. En este caso el controller del componente que reciba los datos hace el papel de “observador” del Observable.
- En el componente (observador) conseguiré recibir los datos que devuelva el observable al que he suscrito.
- Podré cancelar la suscripción a un Observable mediante la función “unsubscribe()”.
- HttpClient, permite una fácil gestión de errores en llamadas HTTP fallidas en Angular.

Detalles a tener en cuenta para nuestro ejemplo de codificación de petición HTTP GET:

- Vamos a usar como URL la dirección del fichero de donde leo los datos JSON, aunque lo normal lo normal sería usar una URL que fuera unEndPoint real.
- Los archivos usados son:
 - o MODIFICAR 4 archivos, ya existentes:
 - 1) El fichero de config. de módulo (“app.module.ts”): añadir librerías necesarias.
 - 2) El fichero del Service (“simulacionbdd.service.ts”): modificaciones principales.
 - 3) El fichero controller del componente hijo (el controller de “primer-componente-formulario-login”): llama y uso al servicio, se subscribe al observable devuelto por el servicio.
 - 4) La vista HTML asociada al controller del componente que observa al observable, para mostrar los datos devueltos de la petición HTTP GET, obtenidos en el controller y pintados en el árbol DOM del navegador.
 - o CREAR 2 nuevos archivos, no existentes:
 - Crear 1 interfaz (“IAllUserFromDB.ts”): dará la estructura de los elementos del array de JSON’s que vendrán de la BBDD.
 - Crear 1 archivo JSON (en la ruta “./assets/data/usuarios.json”): en lugar de hacer una llamada a BBDD y que nos traiga los datos en formato JSON, se genera una lectura del archivo JSON directamente para simular la respuesta que daría la BBDD.

CODIFICACIÓN DE LA LLAMADA “HTTP GET”:

Paso 1 – Modificación fichero “app.module.ts”:

- Crear el “import” de la clase “HttpClientModule” from “./@angular/common/http”.
- Añadir también el “HttpClientModule” al array de imports del módulo.
- +info y codificación realizada en el archivo “app.module.ts”: ver Fig. 4.2.73.

```

  1  import { NgModule } from '@angular/core';
  2  import { BrowserModule } from '@angular/platform-browser';
  3
  4  import { AppComponent } from './app.component';
  5  import { PrimerComponenteFormularioLoginComponent } from './primer-componente-formulario-login/primer-componente-formulario-login.component';
  6
  7  import { FormsModule } from "@angular/forms"
  8
  9  import { SimulacionbddService } from './simulacionbdd.service';
 10
 11 import { HttpClientModule } from "@angular/common/http";
 12
 13 @NgModule({
 14   declarations: [
 15     AppComponent,
 16     PrimerComponenteFormularioLoginComponent
 17   ],
 18   imports: [
 19     BrowserModule,
 20     FormsModule,
 21     HttpClientModule
 22   ],
 23   providers: [SimulacionbddService],
 24   bootstrap: [AppComponent]
 25 })
 26 export class AppModule { }
 27
 28
 29
 30

```

Fig. 4.2.73: Añadimos la clase “HttpClientModule” en la sección de import y del array de imports.

Paso 2 – Ir al archivo del servicio “simulacionbdd.service.ts”:

- Copiar en el portapapeles de Windows el array de datos hardcodeados que estaba devolviendo el método “getAllUsersFromDB()”, del servicio simulacionbdd.service.ts.
- Más info.: ver Fig. 4.2.74.

```

  1  import { Injectable } from '@angular/core';
  2
  3  @Injectable({
  4    providedIn: 'root'
  5  })
  6  export class SimulacionbddService {
  7
  8    constructor() { }
  9
 10   getAllUsersFromDB(){
 11     return [
 12       { 'id':0, 'name':'Pedro', 'fechaIncripcion':'02-01-2022'},
 13       { 'id':1, 'name':'Paco', 'fechaIncripcion':'15-07-2022'},
 14       { 'id':2, 'name':'Pepe', 'fechaIncripcion':'20-12-2022'}
 15     ];
 16   }
 17 }
 18

```

Fig. 4.2.74: Array de datos JSON que tendremos que copiar a un archivo externo JSON.

Paso 3 – Crear el nuevo archivo JSON:

- Mantener en el portapapeles el array de datos hardcodeados.
- Buscar en el árbol de archivos del proyecto el directorio “assets” (está a la altura del componente raíz “app”).
- Crear el subdirectorío “data”, dentro del directorio “assets”.
- Crear el nuevo JSON externo (llamado “”dentro del directorio nuevo (“data”), crear un fichero JSON llamado “allUsersFromDB.json”.
- Nota aclaratoria:

Los elementos de tipo texto o string en formatos JSON deben ir con doble comilla y nunca con comilla simple. Cambiaremos todas las comillas simples.

- Puede verse en la siguiente Fig. 4.2.75 el array de datos JSON copiados del servicio y también todos los string ahora con doble comillas.
- Note que se añade en el fichero JSON, externo al servicio, una posición adicional para poder asegurarnos de que está tomando los datos de este JSON externo, cuando realicemos futuras pruebas.



```
{}
allUsersFromDB.json U X
appFrontend > src > assets > data > {} allUsersFromDB.json > {} 3
1 [ 
2   {"id":0, "name":"Pedro", "fechaInscripcion":"02-01-2022"}, 
3   {"id":1, "name":"Paco", "fechaInscripcion":"15-07-2022"}, 
4   {"id":2, "name":"Pepe", "fechaInscripcion":"20-12-2022"}, 
5   {"id":3, "name":"NUEVO", "fechaInscripcion":"21-12-2022"} ]
```

Fig. 4.2.75: Externalizamos los datos del array de JSON’s hardcodeado en el Service hacia un archivo JSON externo.

Paso 4 – Crear un nuevo archivo INTERFACE en Angular:

- La ruta donde se debe crear la nueva interface será dentro del componente raíz (./app/).
- El archivo de la interface se debe llamar “IAllUsersFromDB.ts”.
- Dentro del nuevo archivo debemos crear la estructura de código de una interface en Angular:

```
“export interface NOMBREDEINTERFACE {  
    VARIABLE1: TIPOvariable1,  
    VARIABLE2: TIPOvariable2,  
    ...  
    VARIABLEn : TIPOvariables  
}”
```

```

  TS IAllUsersFromDB.ts U X
  appFrontend > src > app > TS IAllUsersFromDB.ts > ...
  1
  ● 2 ✓ export interface IAllUsersFromDB{
  3     id:number,
  4     name: string,
  5     fechaInscripcion:string
  6   }
  
```

Fig. 4.2.76: Código de la nueva interfaz “IAllUsersFromDB.ts” de Angular.

Paso 5 – Modificación del método “getAllUsersFromDB”, del Service llamado “simulacionbbdd.service.ts”:

- Crear una variable a nivel de clase, de tipo string, para guardar la URL:
`url:string = “assets/data/allUsersFromDB.json”`
- Traer los 2 imports usados en llamadas HTTP:
`import { HttpClient } from '@angular/common/http'
import { Observable } from 'rxjs';`
- Cargar el archivo de la interfaz JSON que hemos preparado en el proyecto:
`import { IAllUsersFromDB } from './IAllUsersFromDB'`
- El constructor de la clase se inyecta una variable privada de tipo HttpClient. El código del constructor sería este:

```
constructor(private _http : HttpClient){ }
```

- Modificar la cabecera del método “getAllUsersFromDB()”, añadiendo ahora el tipo de dato devuelto. Debe ser un tipo “Observable” de tipo “IAllUsersFormDB[]”. La cabecera del método entonces sería la siguiente:
`getAllUsersFromDB(): Observable<IAllUsersFromDB[]> { ... }`
- Modificación de la línea del return para el método “getAllUsersFromDB”, de forma que se pondrá una línea del estilo a: `http.get(url)`.
A la hora de formar la línea del return tendremos en cuenta 3 cosas:
 - Como la variable “http” era privada, esta se nombra con barra baja (_). Además se inyecta en el constructor, así que la llamaremos como “`this._http`”.
 - La variable “url” la llamaré también como “`this.url`” porque es una variable a nivel de clase.
 - El método “get” devuelve un observable y un tipo concreto que se especifica entre <tipo>.

Podremos suscribirnos al observable y recibir una respuesta de forma asíncrona.

La respuesta obtenida será un array de varios “IAllUsersFromDB” (cada elemento del array es un JSON con estructura idéntica a la descrita por IAllUsersFromDB).

- El código final del return sería el siguiente:

```
return this._http.get<IAllUsersFromDB[]>(this.url);
```

Nota aclaratoria:

Como esa llamada GET devuelve un tipo de datos que es un Observable de Array de IAllUsersFromDB, esto deberá indicarse en la llamada GET. Por eso, la llamada GET quedaría como se indica a continuación:

```
this._http.get<IAllUsersFromDB>(this.url);
```

Puede verse los cambios realizados en el código del servicio “simulacionbdd.service.ts” en la Fig. 4.2.77 (o compare con la anterior modificación de la Fig. 4.2.74).

```

1  import { Injectable } from '@angular/core';
2
3  import { HttpClient } from '@angular/common/http';
4  import { IAllUsersFromDB } from './IAllUsersFromDB';
5  import { Observable } from 'rxjs';
6
7  @Injectable({
8    providedIn: 'root'
9  })
10 export class SimulacionbddService {
11
12   url:string = "assets/data/allUsersFromDB.json";
13
14   constructor(private _http: HttpClient) { }
15   getAllUsersFromDB():Observable<IAllUsersFromDB[]>{
16     /*return [
17       //{'id':0, 'name':'Pedro', 'fechaIncripcion':'02-01-2022'},
18       //{'id':1, 'name':'Paco', 'fechaIncripcion':'15-07-2022'},
19       //{'id':2, 'name':'Pepe', 'fechaIncripcion':'20-12-2022'}
20     ]; */
21
22     // EL METODO DEVUELVE UN OBSERVABLE DE TIPO "ARRAY DE IAllUsersFromDB"
23     return this._http.get<IAllUsersFromDB[]>(this.url);
24     // CADA OBSERVABLE VIENE ASOCIADO CON 1 O MAS OBSERVER'S Y EL QUE LO VA A OBSERVAR SERA NUESTRO
25     // COMPONENTE
26   }
27 }

```

Fig. 4.2.77: modificaciones en el código del servicio, para realizar la llamada HTTP de tipo GET. Véase sección de import's, además de líneas 12, 14, 15 y 23.

Paso 6 – En el componente, subscribirme al observable que devuelve la llamada al método del servicio. También crear una lectura del valor devuelto actualizado (clicando un botón):

Se modificarán 2 métodos en el componente: el OnInit (o su constructor de clase) y una función creada que se activa al clicar un botón.

El código de la suscripción al observable devuelto del método del servicio, se explica a continuación:

- Desde el controller del componente (“primer-componente-formulario-login.ts”) se llama al método “getAllUsersFromDB()”, que está en el archivo del servicio (“simulacionbdd.ts”).
- Como la respuesta de mi llamada a “getAllUsersFromDB()” devuelve un “Observable”, el componente es un “observer” del observable, y entonces puedo subscribirme al Observer devuelto.
- Para subscribirme, desde el controller del componente, llamo al método del servicio y me subscrito a este método del servicio que llamo.
- Código de ANTES (sin observable) vs Código de AHORA (con observable):

ANTES:

```
this.miNuevoServicio.getAllUsersFromDB();
```

AHORA:

```
this.miNuevoServicio.getAllUsersFromDB().subscribe(data=>
```

```
    this.usuarios Conseguidos=data);
```

El código del botón que, al clicarlo, saca por consola del navegador los datos que lee del observable, sería el siguiente:

```
console.log(this.usuariosConseguidos);
```

En la Fig. 4.1.78 puede verse todo el código modificado en el archivo del controller del componente: el código de función OnInit y de la función del botón:

```

ts primer-componente-formulario-login.component.ts ✘
appFrontend > src > app > primer-componente-formulario-login > TS primer-componente-formulario-login.component.ts > Print
1 import { Component, Input, OnInit, Output, EventEmitter } from '@angular/core';
2 import { SimulacionbddService } from '../simulacionbdd.service';
3
4 @Component({
5   selector: 'app-primer-componente-formulario-login',
6   templateUrl: './primer-componente-formulario-login.component.html',
7   styleUrls: ['./primer-componente-formulario-login.component.css']
8 })
9 export class PrimerComponenteFormularioLoginComponent implements OnInit {
10
11   @Input() datosRecibidosEnCompHijo : any;
12
13   usuariosConseguidos : any = [];
14
15   constructor(private miNuevoServicio: SimulacionbddService) { }
16
17   ngOnInit(): void {
18     console.log('OnInit() del comp. hijo ejecutado.')
19     //this.usuariosConseguidos = this.miNuevoServicio.getAllUsersFromDB();
20     //console.log(this.usuariosConseguidos);
21     this.miNuevoServicio.getAllUsersFromDB().subscribe(data=>this.usuariosConseguidos=data);
22   }
23
24   alertaDeEnvioFormulario(){
25     console.log(this.usuariosConseguidos);
26     alert('El formulario de Login se ha enviado.');
27   }

```

Fig. 4.2.78: modificación del código del componente que usaba nuestro servicio antes. Ahora se añade subscripción al observable que devuelve la llamada al método del servicio y un console.log al clicar el botón de ENVIAR FORMULARIO para consultar los datos que se están leyendo (actualizándose variable llamada “usuariosConseguidos”).

Nota informativa de la figura anterior:

1º) Si desea comparar la diferencia entre la recepción del JSON, observe la llamada a los datos JSON hardcodeados en el servicio (antes, Fig. 4.2.71) con la subscripción a la recepción asíncrona de datos mediante HTTP GET (ahora, Fig. 4.2.78).

2º) En la última figura, Fig. 4.2.78, se subscribe al Observable de la petición GET HTTP de forma asíncrona en la función OnInit, y se muestran los datos observados cada vez que se clica el botón “Entrar” que ejecuta la función “alertaDeEnvioFormulario()”.

3º) Si la línea de código donde voy a solicitar que se muestren los datos que recibo (de la petición GET HTTP) estuviera a continuación de la línea del “subscribe().Observable”, podría mostrarse resultados incorrectamente vacíos. Como la función “subscribe(...)” es asíncrona, empieza a ejecutar el subscribe() y sigue en paralelo por la siguiente (mostrado de datos, que resultará dato vacío). Solución: colocar la recepción de datos de petición GET HTTP OK y el mostrados de los mismos dentro de la función “subscribe()”.

CODIFICACIÓN DE GESTIÓN DE ERRORES EN LLAMADAS “HTTP”:

A la hora de codificar las funcionalidades relacionadas con “Gestión de Errores” de una petición HTTP concreta, los cambios serán mínimos y afectarán a solo 3 archivos:

- El controller del componente que se suscribió al Observable de la petición HTTP.
- La vista vinculada con el controller anterior de ese componente.
- El servicio que implemente la llamada HTTP GET.

1º) En el controller del componente:

Limpiar de comentarios la función OnInit.

Crearemos una variable de clase que guarde la respuesta de error, llamada “errorMessage”.

En la función OnInit crear la asignación de los valores de ERROR para cuando resultado devuelto sea FAIL, tras la función que guarda los datos si la llamada si resultado es OK.

En la función flecha que guarda los datos cuando resultado es OK, incorporar mostrado de datos con 2 “console.log()” (un console para texto y otro para mostrar dato). Esto se usará para las pruebas.

Crear un console adicional que la función “alertaDeEnvioFormulario()”, de solamente texto, para conocer en la fase de pruebas que los datos mostrados cuando clique el botón ENTRAR se muestran gracias a esta función externa a la de “subscribe()”.

Ver ejemplo de codificación en la “Fig. 4.2.79”.

```

1 appFrontend > src > app > primer-componente-formulario-login > TS primer-componente-formulario-login.component.ts
2 import { Component, Input, OnInit, Output, EventEmitter } from '@angular/core';
3 import { SimulacionbddService } from './simulacionbdd.service';
4
5 @Component({
6   selector: 'app-primer-componente-formulario-login',
7   templateUrl: './primer-componente-formulario-login.component.html',
8   styleUrls: ['./primer-componente-formulario-login.component.css']
9 })
10 export class PrimerComponenteFormularioLoginComponent implements OnInit {
11   @Input() datosRecibidosEnComPhijo: any;
12   usuariosConseguidos: any = [];
13   errorMessage: string = "";
14
15   constructor(private miNuevoServicio: SimulacionbddService) { }
16
17   ngOnInit(): void {
18     console.log('OnInit() del comp. hijo ejecutado.')
19     this.miNuevoServicio.getAllUsersFromDB()
20       .subscribe(
21         data => { this.usuariosConseguidos = data;
22                   console.log("Cuando se crea llamada GET: ");
23                   console.log(this.usuariosConseguidos)
24                 },
25         error => {this.errorMessage = error}
26       );
27   }
28 }
29
30 alertaDeEnvioFormulario(){
31   console.log("Cuando se clica el botón ENTRAR de la izquierda: ");
32   console.log(this.usuariosConseguidos);
33   alert('El formulario de Login se ha enviado.');
34 }

```

Fig. 4.2.79: Gestión de errores en la subscripción del observable, dentro del controller del componente. Para ello, añadida variable para almacenar error, nueva función flecha para recuperar error dentro del subscribe y adicionalmente se añaden mensajes por consola dentro de la parte del subscribe() ok y fuera del subscribe().

2º) En la vista HTML del componente:

Colocación de mensaje de error incrustado en el HTML y el mensaje de OK, mostrando en el navegador el valor que tenga las variables de ERROR y OK en el controller. Usamos para ello binding simple de doble llave en ambos casos: “{{variable}}”.

Ver ejemplo de codificación en la “Fig. 4.2.80”.

```

1   <table border>
2     <tr>
3       <td>
4         |   <p>[Frase inicial del "primer-componente-formulario-login"].</p>
5         |   <p>| User Name</p>
6         |   <input type="text" name="userName">
7         |   <p>| Password</p>
8         |   <input type="text" name="password">
9         |   <p>
10        |   | <button (click)="alertaDeEnvioFormulario()">
11        |   | Entrar
12        |   | </button>
13        |   </p>
14
15        |   <!-- Recepcion de datos con @Input: desde componente padre a componente hijo -->
16        |   <p>{{datosRecibidosEnCompHijo}}</p>
17
18        |   <!-- Envio de dato con @Output: desde componente hijo a componente padre, c-->
19        |   <button (click)="enviarDatosHaciaAppComponentGenerarEvento()">
20        |   | DATOS DEL HIJO, RECIBIDOS Y ESCRITOS EN EL COMP. PADRE.
21        |   </button>
22        |   <br><br>
23
24        |   <!-- Mensaje de error en la petición GET de HTML impreso por pantalla -->
25        |   <p>Mis usuarios conseguidos son: {{usuariosConseguidos}}</p>
26        |   <br>
27        |   <p>Mis errores de la petición GET HTTP son: {{errorMessage}}</p>
28
29
30
31
32
</td>

```

Fig. 4.2.80: Vista HTML del componente. Se traen los datos de la petición GET almacenados en variable del controller y datos de errores capturados.

3º) En el servicio donde se crea la petición HTTP GET:

Traer 3 imports necesarios para la gestión de errores en peticiones HTTP:

```

import { HttpErrorResponse } from '@angular/common/http'
import { catchError } from 'rxjs/operators'
import { throwError } from 'rxjs';

```

Después, añadir una segunda parte de la función get http, con la función “pipe()”. Esta segunda parte (función pipe en adelante) se manejará los eventos generados por la primera (captura algún tipo de error que haya sido lanzado). Como parámetro de entrada a la función pipe se le pone la función “**catchError(this.funcionErrorHandler)**”.

En resumen, esto hará que al lanzar un error la primera parte, la función pipe() maneje el evento de error, con una función catchError. Esta función catchError() vendrá con un parámetro de entrada que será una función creada por mí mismo, y cuyo objetivo de mi nueva función será tomar el error capturado y mostrarlo en un mensaje al usuario.

Crear la función “funcionErrorHandler” tal como se indica a continuación:

```

funcionErrorHandler ( error : HttpErrorResponse){

    return throwError(error.message);

}

```

Ver ejemplo de codificación en la “Fig. 4.1.81”.

```

ts simulacionbdd.service.ts U X
appFrontend > src > app > ts simulacionbdd.service.ts
1 import { Injectable } from '@angular/core';
2
3 import { HttpClient } from '@angular/common/http';
4 import { IAllUsersFromDB } from './interfaces/IAllUsersFromDB';
5
6 import { Observable } from 'rxjs';
7
8 import { HttpErrorResponse } from '@angular/common/http';
9 import { catchError } from 'rxjs/operators';
10 import { throwError } from 'rxjs';

11 @Injectable({
12   providedIn: 'root'
13 })
14 export class SimulacionbddService {
15
16   url:string = "assets/data/allUsers.json";
17
18   constructor(private _http: HttpClient) {
19
20     getAllUsersFromDB(): Observable<IAllUsersFromDB[]> {
21       // EL METODO DEVUELVE UN OBSERVABLE DE TIPO "ARRAY DE IAllUsersFromDB"
22       return this._http.get<IAllUsersFromDB[]>(this.url)
23         .pipe(
24           catchError(this.funcionErrorHandler)
25         );
26     }
27   }
28
29   funcionErrorHandler( error : HttpErrorResponse ) {
30     return throwError(error.message);
31   }
32 }

```

Librerías importadas de gestión de errores, en llamadas HTTP.

1ª) Respuesta de error http.
2ª) Capturar error http.
3ª) Lanzar error hacia el controller llamante, con el correspondiente mensaje de error.

Función pipe que se ejecuta si genera error la 1ª parte de la llamada GET HTTP.
Captura errores y entra en una función personalizada.

Función personalizada que devuelve en un mensaje el error producido.

Fig. 4.2.81: Modificación del servicio, para implementar la gestión de errores en la llamada HTTP GET. Se incluyen 3 import's en ella, además de la función pipe() ejecutada cuando genere error y la función final que gestiona el error finalmente.

PRUEBAS CON LA GESTIÓN DE ERRORES EN LLAMADAS “HTTP”:

Se deben pasar 3 pruebas para verificar que lo indicado anteriormente es así:

- 1- Cuando aparezcan los datos devueltos por la llamada GET HTTP OK no aparecerán datos de error, y vice versa.

Ver ejemplo de codificación en la “Fig. 4.2.82”.

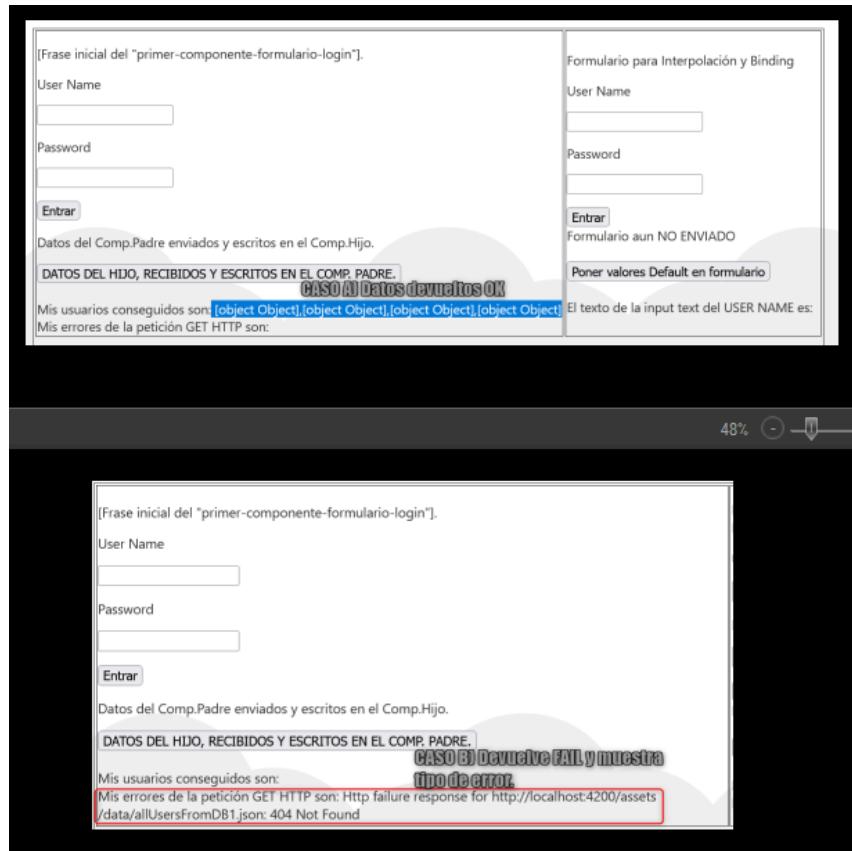


Fig. 4.2.82: Datos devueltos de la petición HTTP GET, mostrados en el navegador. Arriba (caso A) mostrados datos cuando HTTP GET es OK, abajo (caso B) mostrados datos cuando HTTP GET es FAIL.

- 2- Al cargar la página web se verá en la consola del navegador un par de líneas con los datos obtenidos OK en la función flecha del subscribe, y se indica claramente.

Ver ejemplo de codificación en la “Fig. 4.2.83”.

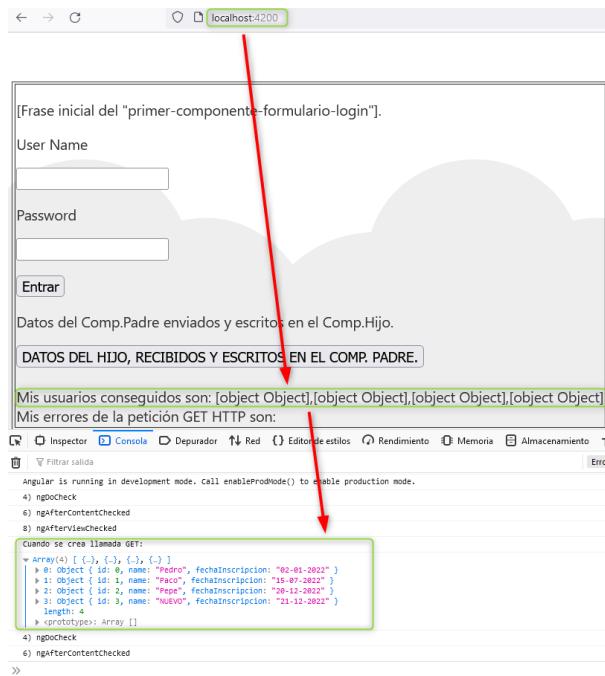


Fig. 4.2.83: Datos devueltos por petición HTTP GET desde dentro de la función subscribe(). Mostrado por consola.

- 3- Al clicar en el botón ACEPTAR del formulario izquierdo aparecerá en la consola del navegador 2 líneas con los datos OK, pero con mensaje distinto al que se muestra al cargar página sin clicar botón. Mensaje ejecutado fuera de función “subscribe()” y en paralelo sin que “subscribe()” haya terminado su ejecución.

Ver ejemplo de codificación en la “Fig. 4.2.84”.

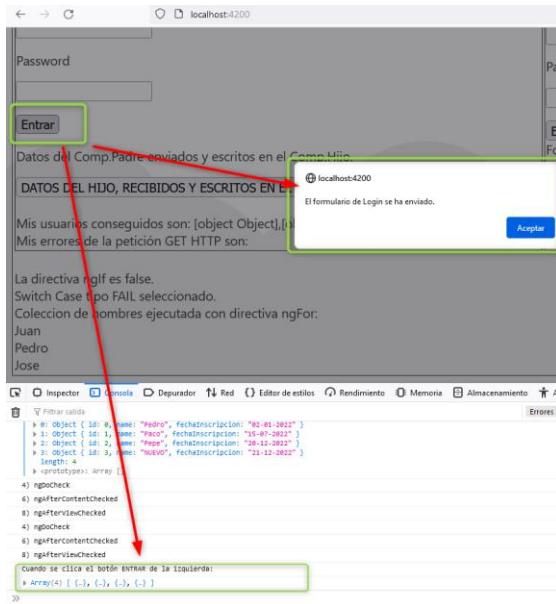


Fig. 4.2.84: Datos devueltos por petición HTTP GET desde fuera de la función subscribe(). Mostrados por consola al clicar el botón ENTRAR. Esta solución mostrará resultados incorrectos cuando se pidan datos antes de que hayan llegado (la acción de clicar el botón se realiza mientras función subscribe() no terminó).

m) Enrutamiento interno

Para configurar el enrutamiento interno en Angular debemos seguir estos 10 pasos:

- Paso 1: Crear el nuevo módulo de enrutamiento interno (creando una aplicación nueva con el módulo incluido o incluyendo dicho módulo a la aplicación actual).

CASO A) Crear una nueva aplicación con enrutamiento y css activos, durante la instalación de la nueva aplicación en línea de comandos.

- Ejecutar en terminal de comandos el comando siguiente, fuera de cualquier aplicación Angular:
“ng new route”
- Revisar posteriormente, en el terminal de comandos, que se han creado todos los archivos necesarios sin errores, encontrar un nuevo archivo “app-routing.module.ts” en nuestro directorio y comprobar que este trae un código mínimo.
- Puedes comparar tus resultados con los que puede encontrar aquí, desde la Fig. 4.2.85 a Fig. 4.2.87.

```
PS C:\          \Codigo\Frontend Angular\appFrontend> ng new route
Error: This command is not available when running the Angular CLI inside a workspace.
PS C:\          \Codigo\Frontend Angular\appFrontend> cd ..
PS C:\          \Codigo\Frontend Angular> ng new route
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE route/angular.json (2917 bytes)
CREATE route/package.json (1036 bytes)
CREATE route/README.md (1059 bytes)
CREATE route/tsconfig.json (863 bytes)
CREATE route/.editorconfig (274 bytes)
CREATE route/.gitignore (548 bytes)
CREATE route/.browserslistrc (600 bytes)
CREATE route/karma.conf.js (1422 bytes)
CREATE route/tsconfig.app.json (287 bytes)
CREATE route/tsconfig.spec.json (333 bytes)
CREATE route/.vscode/extensions.json (130 bytes)
```

Fig. 4.2.85: Instalación de una nueva aplicación que llamamos “route”. Durante las configuraciones de instalación indicar YES cuando pregunte si instalar “Angular routing” y “css”.

```
Frontend Angular > route > src > app > TS app-routing.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3
4 const routes: Routes = [];
5
6 @NgModule({
7   imports: [RouterModule.forRoot(routes)],
8   exports: [RouterModule]
9 })
10 export class AppRoutingModule {}
```

Fig. 4.2.86: En la nueva aplicación, ver archivo “app-routing.module.ts”.

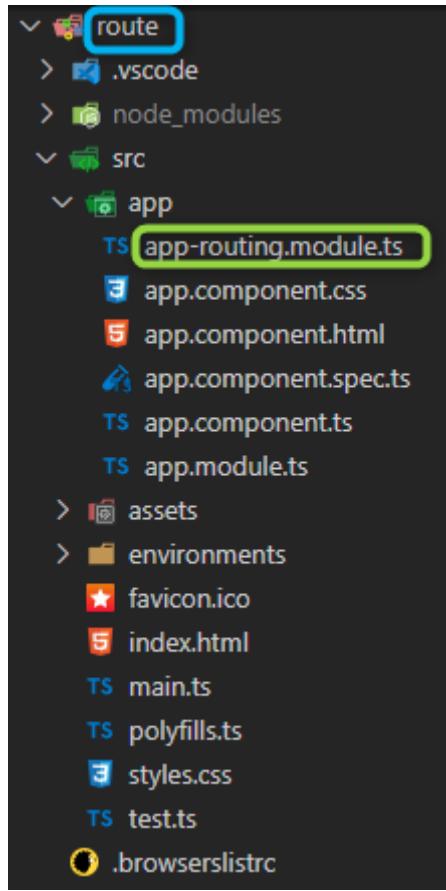


Fig. 4.2.87: Directorio de archivos de nueva aplicació con el módulo de routing.

CASO B) A partir de la aplicación ya creada (appFrontend), y con el enrutamiento inactivo, instalar el módulo enrutamiento interno únicamente (por línea de comandos).

- Usar el siguiente comando, dentro de la aplicación Angular que se desea configurar:
“ng generate module app-routing --flat --module-app”
- Revisar posteriormente, en el terminal de comandos, que se han creado todos los archivos necesarios sin errores, encontrar un nuevo archivo “app-routing.module.ts” en nuestro directorio y comprobar que este trae un código mínimo.
- Puedes comparar tus resultados con los que puede encontrar aquí desde la Fig. 4.2.88 a la Fig. 4.2.90.

```

PS C:\          \Codigo\Frontend Angular\appFrontend>
PS C:\          \Codigo\Frontend Angular\appFrontend> ng generate module app-routing --flat --module=app
>>
CREATE src/app/app-routing.module.ts (196 bytes)
UPDATE src/app/app.module.ts (816 bytes)
PS C:\          \Codigo\Frontend Angular\appFrontend>

```

Fig. 4.2.88: Instalación únicamente del módulo “app-routing”. Indicamos con “--flat” que el nuevo archivo se coloque debajo de “./src/app”. También se pide con “--module=app” que el nuevo módulo se registre en la matriz del módulo “app”.

```

Frontend Angular > appFrontend > src > app > ts app-routing.module.ts > AppRoutingModule
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4
5 @NgModule({
6   declarations: [],
7   imports: [
8     CommonModule
9   ]
10 })
11 export class AppRoutingModule { }
12

```

Fig. 4.2.89: Código del archivo “app-routing.module.ts”.

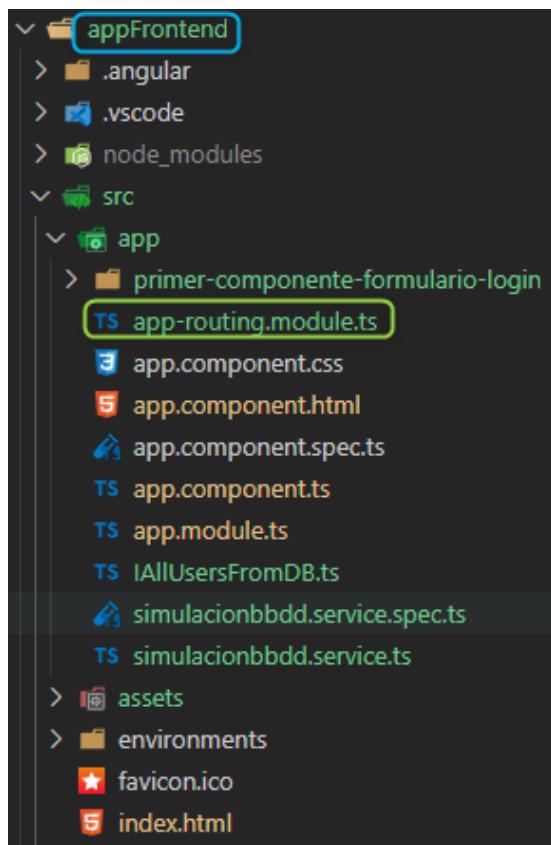


Fig. 4.2.90: Directorio de la antigua aplicación, con el nuevo archivo “app-routing.module.ts”.

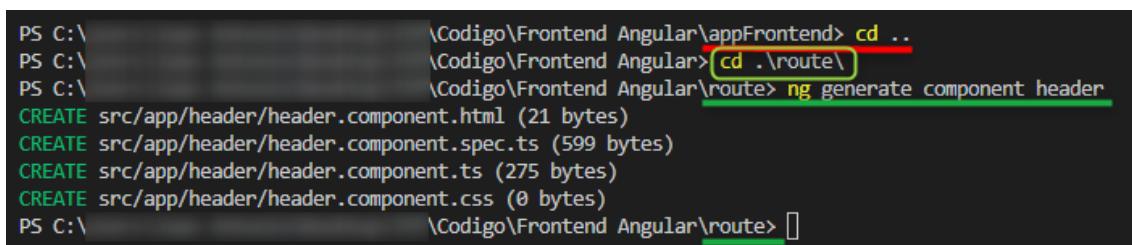
Caso elegido para el caso que nos ocupará (caso A):

- El caso A (crear una aplicación nueva con el enrutamiento activado).
- El caso A, al partir de una nueva aplicación, no nos creará confusión en las pruebas de la vista HTML.

· Paso 2: Como estábamos dentro de la aplicación anterior (llamada appFrontend), muévete con el terminal de comandos del propio Visual Studio Code, a la ruta de la nueva aplicación (llamada “route”). Ver las primeras 2 líneas de la Fig. 4.2.91.

· Paso 3: Crear un componente llamado “header”, que se usará para una barra de la cabecera con links superiores de la página.

- Esta barra será común en todas las páginas o rutas de la aplicación.
- El componente se creará en el directorio “./app/header”.
- Para crear el componente, usar comando “**ng generate component header**”
- Ver un caso práctico, ver tercera línea de Fig. 4.2.91.



```
PS C:\          \Codigo\Frontend Angular\appFrontend> cd ..  
PS C:\          \Codigo\Frontend Angular> cd ./route  
PS C:\          \Codigo\Frontend Angular\route> ng generate component header  
CREATE src/app/header/header.component.html (21 bytes)  
CREATE src/app/header/header.component.spec.ts (599 bytes)  
CREATE src/app/header/header.component.ts (275 bytes)  
CREATE src/app/header/header.component.css (0 bytes)  
PS C:\          \Codigo\Frontend Angular\route> []
```

Fig. 4.2.91: Caso práctico de la terminal de comandos de Visual Studio, para las tareas realizadas en los pasos 2 y 3.

· Paso 4: Modificar el HTML que viene por defecto en los componentes “app” y “header”:

- Componente “app” (archivo “app.component.html”):
 - Borrar todo el código que venía por defecto (Fig. 4.2.92).
 - Añadimos el selector del componente “header” (Fig. 4.2.93).
- Componente “header” (archivo “header.component.html”):
 - Borrar todo el código que venía por defecto (Fig. 4.2.94).
 - Añadimos un par de LINKS a la cabecera (Fig. 4.2.95).

```
Frontend Angular > route > src > app > app.component.html
Go to component
1  <!-- * * * * * * * * * * * * * * * * * * * * * * * *
2  <!-- * * * * * * * * * * * * * The content below *
3  <!-- * * * * * * * * * * * * * is only a placeholder
4  <!-- * * * * * * * * * * * * * and can be replaced.
5  <!-- * * * * * * * * * * * * * * * * * * * * * * *
6  <!-- * * * * * * * * * * * * Delete the template be
7  <!-- * * * * * * * to get started with your pr
8  <!-- * * * * * * * * * * * * * * * * * * * * * *
9
10 <style>
11   :host {
12     font-family: -apple-system, BlinkMacSystem
13     font-size: 14px;
14     color: □#333;
15     box-sizing: border-box;
16     -webkit-font-smoothing: antialiased;
17     -moz-osx-font-smoothing: grayscale;
```

Fig. 4.2.92: Código que traía inicialmente la vista del componente App.



Frontend Angular > route > src > app > **app.component.html**

Go to component

```
1 | <app-header></app-header>
```

Fig. 4.2.93: Nuevo código de la vista, del componente App.

```
Frontend Angular > route > src > app > header > header.component.html
Go to component
1  <p>header works!</p>
```

Fig. 4.2.94: Código que traía inicialmente la vista del componente Header.

```
Frontend Angular > route > src > app > header > header.component.html
Go to component
1  <ul>
2    |   <a routerLink=""> LISTA</a>
3  </ul>
4  <ul>
5    |   <a routerLink="create"> CREAR</a>
6  </ul>
```

Fig. 4.2.95: Nuevo código de la vista, del componente Header.

- Paso 5: Crear 2 nuevos componentes, que será los 2 links dentro del componente Header.

- Los componentes que vamos a crear son:
 - “listaComponente”
 - “createComponente”.
- Ambos componentes estarán dentro del directorio “task”, quedando entonces en:
 - task/createComponente
 - task/listaComponente
- Los comandos del terminal de comandos que se usan para crearlos serán:
 - “ng generate component task/createComponente”
 - “ng generate component task/listaComponente”
- A continuación, un caso práctico en el terminal de comandos (Fig. 4.2.96).

```
PS C:\Users\Juan Antonio\Desktop\TFM\Codigo\Frontend Angular\route> ng generate component task/createComponente
CREATE src/app/task/create-componente/create-componente.component.html (32 bytes)
CREATE src/app/task/create-componente/create-componente.component.spec.ts (670 bytes)
CREATE src/app/task/create-componente/create-componente.component.ts (318 bytes)
CREATE src/app/task/create-componente/create-componente.component.css (0 bytes)
UPDATE src/app/app.module.ts (604 bytes)
PS C:\Users\Juan Antonio\Desktop\TFM\Codigo\Frontend Angular\route> ng generate component task/listaComponente
CREATE src/app/task/lista-componente/lista-componente.component.html (31 bytes)
CREATE src/app/task/lista-componente/lista-componente.component.spec.ts (663 bytes)
CREATE src/app/task/lista-componente/lista-componente.component.ts (314 bytes)
CREATE src/app/task/lista-componente/lista-componente.component.css (0 bytes)
UPDATE src/app/app.module.ts (729 bytes)
PS C:\Users\Juan Antonio\Desktop\TFM\Codigo\Frontend Angular\route>
```

Fig. 4.2.96: Creación de los 2 componentes, desde el terminal de comandos.

- Paso 6: Revisar fichero de configuración del módulo App (llamada “app.module.ts”):

- Comprobar que estén colocados los 3 import's de nuestros 3 nuevos componentes: header, createComponente, listaComponente.
- Caso práctico en fichero “[app.module.ts](#)”, visualizado en la Fig. 4.2.97.

```
Frontend Angular > route > src > app > ts app.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { HeaderComponent } from './header/header.component';
7 import { CreateComponenteComponent } from './task/create-componente/create-componente.component';
8 import { ListaComponenteComponent } from './task/lista-componente/lista-componente.component';
9
10 @NgModule({
11   declarations: [
12     AppComponent,
13     HeaderComponent,
14     CreateComponenteComponent,
15     ListaComponenteComponent
16   ],
17   imports: [
18     BrowserModule,
19     AppRoutingModule
20   ],
21 })
```

Fig. 4.2.97: Visualización de los 3 los import's y las 3 declaraciones en el fichero de configuración del módulo App. Deberían venir automáticamente ya incluidos.

· Paso 7: Cerrar el “app.module.ts”. Hasta aquí es todo lo que necesitamos modificar en el fichero “app-module.ts”.

· Paso 8: Configurar el fichero del nuevo módulo “app-routing.module.ts”.

- Con esta acción se prepara el enrutamiento dinámico de la aplicación.
- Importar en este fichero los 3 nuevos componentes recién creados: header, create y list. Ejemplo del nuevo código:
 - import { HeaderComponent } from './header/header.component';
 - import { CreateComponenteComponent } from './task/create-componente/create-componente.component';
 - import { ListaComponenteComponent } from './task/lista-componente/lista-componente.component';
- Revisar que vienen ya importadas las clases para hacer routing con Angular (clases “Routes” y “RouterModule”).
- La clase RouteModule se usa para cargar el módulo de enrutamiento y la clase Routes permite usar las rutas. Ejemplo del nuevo código:
 - import { RouterModule, Routes } from '@angular/router';
- Inmediatamente seguido de los imports, debes crear la variable de tipo Routes, (va a ser el array de rutas de la aplicación). Ejemplo del nuevo código:

```
const routes: Routes = [
  {path:'list', component : ListaComponenteComponent},
  {path:'create', component : CreateComponenteComponent},
  {path:"/", redirectTo: '/list', pathMatch: 'full'},
]
```

- En la sección de @NgModule tenemos un array de imports y otro de exports. Esta sección debería venir ya configurada así:

- imports : [RouterModule.forChild(routes)]
- exports: [RouteModule]
- El código de la sección @NgModule sería el siguiente:

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouteModule]
})
```

- Mostramos el código completo del archivo “app-routing.module.ts”, en la siguiente imagen. Ver Fig. 4.2.98.

```

Frontend Angular > route > src > app > ts app-routing.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { HeaderComponent } from './header/header.component';
3 import { CreateComponenteComponent } from './task/create-componente/create-componente.component';
4 import { ListaComponenteComponent } from './task/lista-componente/lista-componente.component';
5 import { RouterModule, Routes } from '@angular/router';
6
7
8
9 const routes: Routes = [
10   {path:'list', component : ListaComponenteComponent},
11   {path:'create', component : CreateComponenteComponent},
12   {path:'', redirectTo: '/list', pathMatch: 'full'},
13 ]
14
15
16 @NgModule({
17   imports: [RouterModule.forRoot(routes)],
18   exports: [RouterModule]
19 })
20 export class AppRoutingModule { }
21

```

Fig. 4.2.98: Código completo del archivo de configuración del módulo de app-routing.

- Paso 9: Ya tenemos completada la configuración necesaria para enrutar dinámicamente toda la aplicación. Sin embargo, no funcionarán aun los links de la cabecera de la página (header).

- No hemos llamado todavía, desde la vista HTML del componente “App”, al selector del componente del módulo de enrutamiento (app-routing).
- Colocar el selector “router-outlet” para llamar a componente “app-routing”.
- Veremos un ejemplo práctico en la Fig. 4.2.99.

A la figura anterior Fig 4.1.93 le faltaba la llamada al componente “router-outlet”. Le añadiremos esta llamada del componente del nuevo módulo de enrutamiento, como podrá ver a continuación en la Fig. 4.1.99:

```

Frontend Angular > route > src > app > html app.component.html
Go to component
1 <app-header></app-header>
2 <router-outlet></router-outlet>

```

Fig. 4.2.99: Se incluye componente de enrutamiento en la vista del componente “app”. Llamada al nuevo componente de enrutamiento usando el selector “router-outlet” después de cada llamada al componente “app-header”.

- Paso 10: Realización de pruebas sobre el navegador, de los distintos enrutamientos realizados.

- Fig. 4.2.100: link “/list”
- Fig. 4.2.101: link “create”
- Fig. 4.2.102: link “”

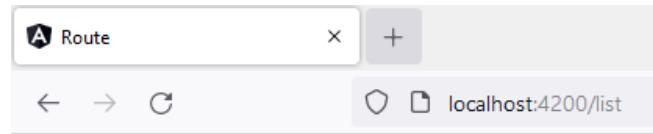


Fig. 4.2.100: Link de “list” y home. Cambia a mensaje de pag. Central “LISTA”.

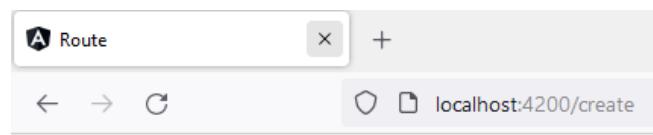
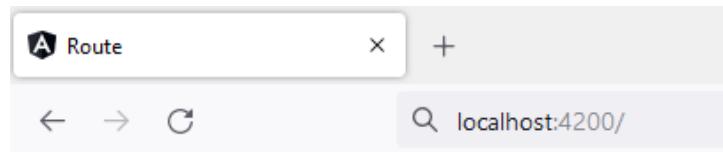


Fig. 4.2.101: Link de “crear”. Cambia a mensaje de pag. Central “CREATE”.



LISTA-componente works!

Fig. 4.2.102: Link de home page, apunta a link “list” y cambia URL vacía a “/list”.

n) Instalación y uso de librerías de estilos: “Angular Materials” y “Bootstrap”.

Para el diseño de la aplicación desarrollada en Angular puede usar CSS, Javascript y librerías de estilos frontend. Ahí es donde entran en juego tanto “Angular Materials” y “Bootstrap”.

Partiendo de la aplicación en el estado que estaba el final del apartado anterior (desde Fig. 4.1.100 a Fig. 4.1.102), ahora se aportará diseño al Header de la web.

Caso A: desarrollar el Header usando “Angular Material”:

Paso 1: Instalar en el terminal de comandos de Visual Studio Code “Angular Materials”.

- Usar el siguiente comando desde la terminal:
 - ❖ “ng add @angular/material”
- Ver ejecución de comando y opciones configuradas en la siguiente imagen (Fig. 4.2.103).

```
PS C:\          \Codigo\Frontend Angular\route> ng add @angular/material
i Using package manager: npm
✓ Found compatible package version: @angular/material@14.2.4.
✓ Package information loaded.

The package @angular/material@14.2.4 will be installed and executed.
Would you like to proceed? Yes
✓ Packages successfully installed.
? Choose a prebuilt theme name, or "custom" for a custom theme: Indigo/Pink [ Preview: https://material.angular.io?theme=indigo-pink ]
? Set up global Angular Material typography styles? Yes
? Include the Angular animations module? Include and enable animations
UPDATE package.json (1102 bytes)
✓ Packages installed successfully.
UPDATE src/app/app.module.ts (916 bytes)
UPDATE angular.json (3081 bytes)
UPDATE src/index.html (573 bytes)
UPDATE src/styles.css (181 bytes)
PS C:\          \Codigo\Frontend Angular\route>
```

Fig. 4.2.103: Instalación de “Angular Materials” desde el terminal de comandos y opciones adicionales seleccionadas.

Paso 2: Ir a la página oficial de Angular Materials, sección “components” → Toolbar:

<https://material.angular.io/components/toolbar/examples>

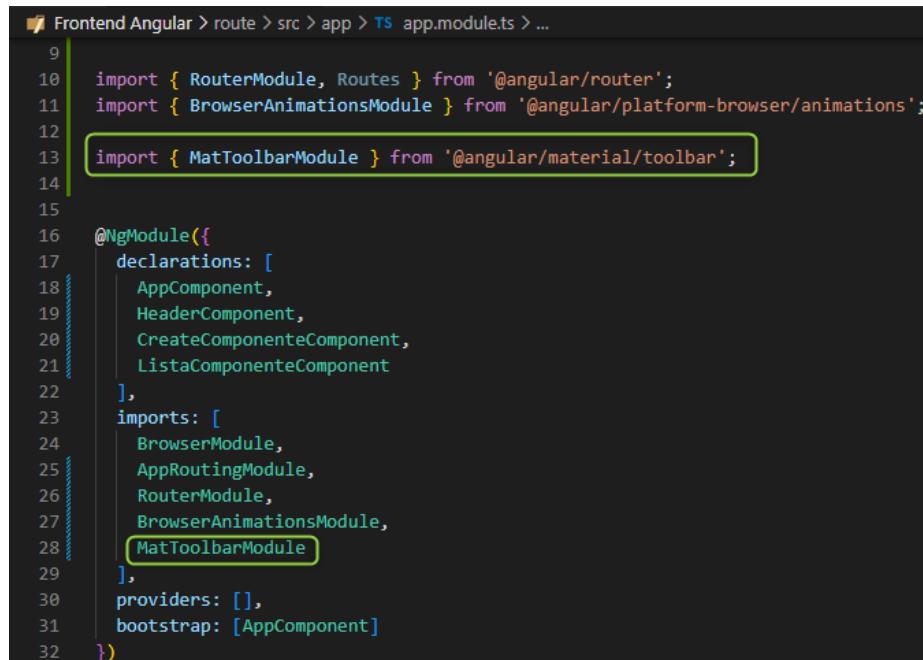
Paso 3: Importar con Angular los componentes HTML que vayamos a usar de “Angular Materials”.

- Entrar al archivo de configuración del módulo principal (“app.module.ts”).

- Importar en “app.module.ts” el elemento “Toolbar”. Para ello generar el siguiente import:

```
import { MatToolbarModule } from '@angular/material/toolbar'
```

- Después añadir al array de imports, que está algo más abajo, la nueva clase importada (esto permitirá el uso de Toolbar en todos los componentes que pertenezcan al módulo App).
- Ver Fig. 4.2.104, para ver el código final que se añadió en “app.module.ts”.



```

Frontend Angular > route > src > app > app.module.ts > ...
9
10 import { RouterModule, Routes } from '@angular/router';
11 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
12
13 import { MatToolbarModule } from '@angular/material/toolbar'; // Line 13 highlighted
14
15
16 @NgModule({
17   declarations: [
18     AppComponent,
19     HeaderComponent,
20     CreateComponenteComponent,
21     ListaComponenteComponent
22   ],
23   imports: [
24     BrowserModule,
25     AppRoutingModule,
26     RouterModule,
27     BrowserAnimationsModule,
28     MatToolbarModule // Line 28 highlighted
29   ],
30   providers: [],
31   bootstrap: [AppComponent]
32 })

```

Fig. 4.2.104: Modificación en archivo “app.module.ts”, para poder usar nuevo elemento Toolbar de Angular Materials.

Paso 4: Volver a la web de Angular Material que usamos en el paso 2, y conseguir el código del elemento “Toolbar” (solo copiaremos el código seleccionado en azul de la Fig. 4.2.105):

```

<p>
  <mat-toolbar color="primary">
    <mat-toolbar-row>
      <span>My App</span>
      <span class="example-spacer"></span>
      <button mat-icon-button class="example-icon" aria-label="Example icon-button with menu icon">
        <mat-icon>menu</mat-icon>
      </button>
    </mat-toolbar-row>

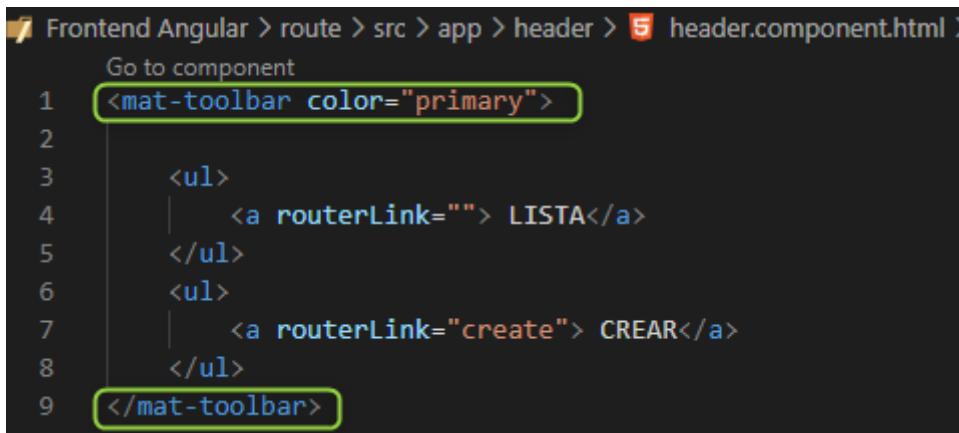
    <mat-toolbar-row>
      <span>Second Line</span>
      <span class="example-spacer"></span>
      <button mat-icon-button class="example-icon favorite-icon" aria-label="Example icon-button with heart icon">
        <mat-icon>favorite</mat-icon>
      </button>
      <button mat-icon-button class="example-icon" aria-label="Example icon-button with share icon">
        <mat-icon>share</mat-icon>
      </button>
    </mat-toolbar-row>
  </mat-toolbar>
</p>

```

Fig. 4.2.105: Código de ejemplo de elemento Toolbar de web oficial “material.angular.io”. Nos interesa solo la etiqueta seleccionada.

Paso 5: Copia y pega la línea seleccionada al inicio de la vista HTML del componente “header” (y su correspondiente etiqueta de cierre al final de la vista).

- Código copiado en la vista HTML del elemento Toolbar en la Fig. 4.2.106.
- Resultados visuales en el navegador en la Fig. 4.2.107.



```

Frontend Angular > route > src > app > header > 5 header.component.html
Go to component
1  <mat-toolbar color="primary">
2
3    <ul>
4      <a routerLink=""> LISTA</a>
5    </ul>
6    <ul>
7      <a routerLink="create"> CREAR</a>
8    </ul>
9  </mat-toolbar>

```

Fig. 4.2.106: Código modificado de la vista HTML en archivo “header.component.html”. Dos nuevas líneas incluidas señaladas en verde.

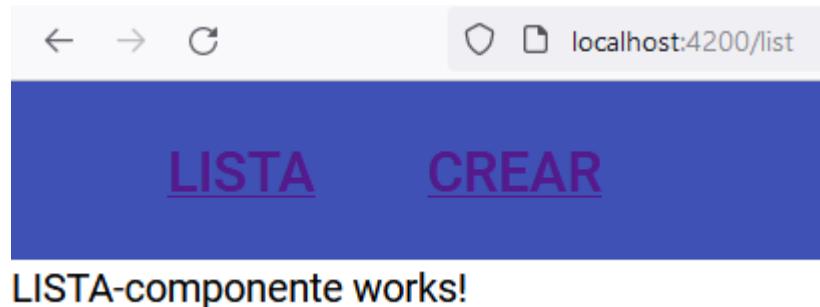


Fig. 4.2.107: Resultado visual del nuevo código incluido de Angular Materials.

Paso 6: si decidimos aportar algo de CSS al Header (en archivo llamada “header.component.css”, quedaría así:

- CSS creado del componente Header, en la Fig. 4.2.108.
- Resultados visuales en el navegador, en la Fig. 4.2.109.

A screenshot of a code editor showing the 'header.component.css' file. The code is:

```
Frontend Angular > route > src > app > header > header.component.css
1 a{
2   |   text-decoration: none;
3   |   color: white;
4 }
```

The code editor interface is visible at the top, showing the file path and a preview icon.

Fig. 4.2.108: Código de CSS añadido, archivo “header.component.css”.



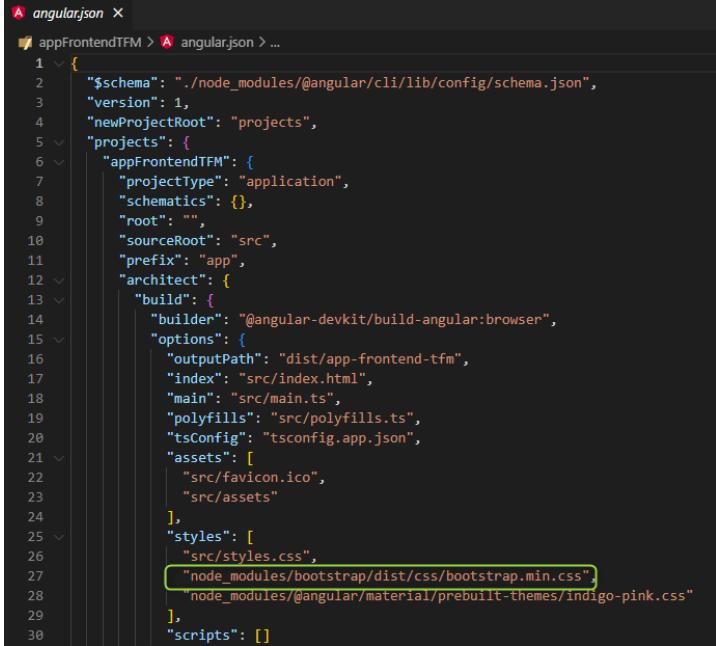
Fig. 4.2.109: Resultado nuevo del Header tras añadir las CSS.

Caso B: Creación de componentes y páginas enteras con Bootstrap:

Paso 1: Ir a la web de Bootstrap y selecciona los ejemplos que más se adapten a lo que se busque para el nuevo diseño web. La web oficial de Bootstrap es la siguiente: <https://getbootstrap.com/>.

Paso 2: Instalar la librería de Bootstrap en Angular:

- Usar el siguiente comando desde la terminal:
 - ❖ `npm install bootstrap`
- Añadir la librería de CSS dentro del fichero “angular.json”, como se muestra en la Fig. 4.1.110 (no se olvide de añadir la coma al final si procede):



```
angular.json
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "appFrontendTFM": {
      "projectType": "application",
      "schematics": {},
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/app-frontend-tfm",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": "src/polyfills.ts",
            "tsConfig": "tsconfig.app.json",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/styles.css",
              "node_modules/bootstrap/dist/css/bootstrap.min.css",
              "node_modules/@angular/material/prebuilt-themes/indigo-pink.css"
            ],
            "scripts": []
          }
        }
      }
    }
  }
}
```

Fig. 4.2.110: Añadida la ruta de los CSS de Bootstrap en “angular.json”, para poder usar los estilos en el proyecto.

Paso 3: Ir a la página oficial de Boostrap, sección de EXAMPLES y los ejemplos de HEADERS.

Paso 4: Como no da el código propio pero sí dan los modelos ya pintados en la web, clica F12 para abrir el inspector de código. Entonces seleccione el tipo de Header elegido con el inspector. Abajo verá cómo se le marca el código del Header elegido. Sobre el código remarcado, haz botón derecho, y decirle “copiar código externo” (para copiar todo el código del Header). Véase Fig. 4.2.111:

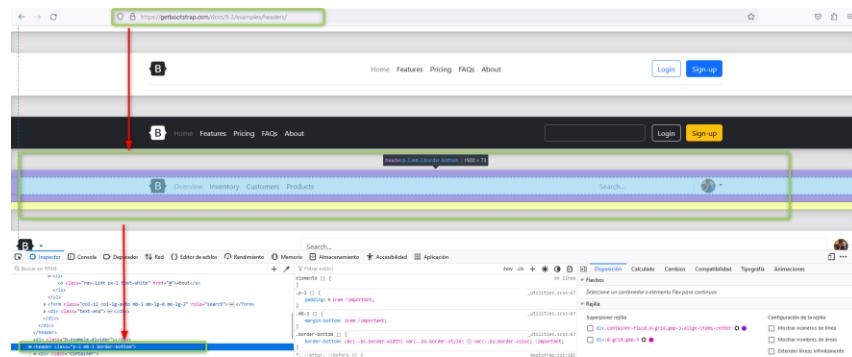


Fig. 4.2.111: Copiar código de Header seleccionado del inspector de código.

Paso 5: Pegar el código (el que acabamos de copiar) sobre el HTML del proyecto Angular, que sea del componente elegido. Véase Fig. 4.2.112 para saber cómo quedaría la vista de un componente Angular con esta plantilla Bootstrap:



Fig. 4.2.112: Plantilla Bootstrap colocada en nuestro proyecto Angular.

NOTA: Tenga presente que Bootstrap es una librería de CSS, así que una vez instalada, con solo llamar a las clases Bootstrap conocidas en su documentación, te aportará nuevos colores y diseños CSS.

o) Uso de Guardianes en Angular.

Un Guardian en Angular es una herramienta que se utiliza para proteger páginas.

En caso de que una página protegida no cumpla la condición para ser accesible no se podrá acceder a ella.

Pero, ¿cómo se crea un Guardian en Angular? Seguir los siguientes pasos:

Paso 1: En Visual Studio Code, dentro del componente Angular que queremos proteger, crea el siguiente archivo: “guardian-si-no-logeado.ts”.

Paso 2: Crea la nueva clase como se ve en la siguiente Fig. 4.2.113:

- El nombre de la nueva clase puede ser el que quieras, pero obligatoriamente debe implementar la clase CanActivate.

```
guardian-si-no-logeado.ts
appFrontendITM > src > app > componentesIndependientes > login > guardian-si-no-logeado > GuardianSiNoLogeado
1 import { Injectable } from '@angular/core';
2 import { ActivatedRouteSnapshot, CanActivate, CanActivateFn, Router, RouterStateSnapshot, UrlTree } from "@angular/router";
3 import { CookieService } from "ngx-cookie-service";
4 import { Observable } from "rxjs";
5
6 @Injectable()
7 export class GuardianSiNoLogeado implements CanActivate{
8
9   constructor(private router : Router, private cookie : CookieService) { }
10
11  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
12
13    if(this.cookie.get("JWt_relishiu") != ""){
14      return true;
15    } else{
16      this.router.navigate(["login"]);
17      return false;
18    }
19  }
}
```

Fig. 4.2.113: Implementación completa de la clase que define el Guardián.

Paso 3: Registrar, en el fichero “app.module”, la nueva clase guardián (Fig. 4.2.114):

Fig. 4.2.114: Modificación del archivo app.module.ts para añadir la importación del nuevo guardián.

Paso 4: Ir al fichero donde definíamos las rutas de cada página, y le indicamos que esa ruta irá protegida por el guardián (la rutas estaban en “app-routing.module.ts”).

Paso 5: Queremos proteger una página interna de la aplicación, a la que llamaremos “Dashboard”. Para proteger “Dashboard”, colocaré el guardián en el array de rutas, a continuación del nombre del componente (ver Fig. 4.2.115).

```
1  app-routing.module.ts
2
3  appFrontendTFRM > src > app > TS app-routing.module.ts ...
4
5  import { NgModule } from '@angular/core';
6  import { RouterModule, Routes } from '@angular/router';
7
8  import { DashboardComponent } from './componentesIndependientes/dashboard/dashboard.component';
9  import { ErrorComponent } from './componentesIndependientes/error/error.component';
10 import { LoginComponent } from './componentesIndependientes/login/login.component';
11 import { RecuperarPasswordComponent } from './componentesIndependientes/recuperar-password/recuperar-password.component';
12 import { RegistrarUsuarioComponent } from './componentesIndependientes/registrar-usuario/registrar-usuario.component';
13 import { VerificarCorreoComponent } from './componentesIndependientes/verificar-correo/verificar-correo.component';
14
15 import { GuardianSiNoLogeado } from './componentesIndependientes/login/guardian-si-no-logeado';
16
17 const routes: Routes = [
18  {path: 'login', component: LoginComponent},
19  {path: 'signup', component: RegistrarUsuarioComponent},
20  {path: 'recuperar-password', component: RecuperarPasswordComponent},
21  {path: 'dashboard', component: DashboardComponent, canActivate: [GuardianSiNoLogeado]},
22  {path: 'verificar-correo', component: VerificarCorreoComponent},
23  {path: '*', redirectTo: '/dashboard', pathMatch: 'full'},
24  {path: '**', component: ErrorComponent}
25 ]
26
27 @NgModule({
28   imports: [RouterModule.forRoot(routes)],
29   exports: [RouterModule]
30 })
31 export class AppRoutingModule {}
```

Fig. 4.2.115: Configurando que la página “Dashboard” está protegido por el guardián.

Paso 6: Si deseas proteger más páginas, localiza sus rutas y repite el proceso (con ese guardián o usando otro distinto).

- Nota: nunca protejas páginas que sean accesibles desde fuera de la aplicación. (daría errores).

p) Instalación de Firebase en un proyecto Angular.

A la hora de instalar Firebase en Angular, usaremos el “proyecto Firebase” que ya creamos (en la nube), el cual ya usábamos en el apartado 4.1.

Para instalar y poder usar Firebase en Angular, siga los siguientes pasos:

Paso 1: Entrar en el proyecto Firebase creado, e ir a la sección “Configuración del proyecto”.

- En esta sección se verá la información para instalar el “SDK de Firebase” con NPM, y después la info para configurar la base de datos.
- Puede ver un ejemplo visual de la información proporcionada en “Configuración del proyecto” (en Firebase), en la Fig. 4.2.116.

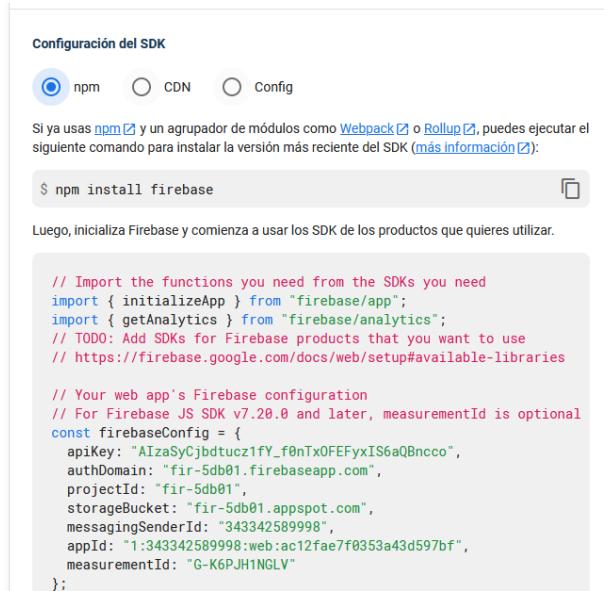


Fig. 4.2.116: Información proporcionada por Firebase, sección “Configuración del proyecto”, para instalar el SDK de Firebase en Angular y Java.

Nota importante: no vamos a usar el comando propuesto en la Fig. 4.2.116: **npm install firebase**. En su lugar, usaremos otro comando: **npm i --sabe firebase @angular/fire**.

Paso 2: Después de instalar las librerías para Firebase en nuestro terminal de consola, comprobar que se hayan incluido correctamente en la sección de dependencias que tiene Angular (en el archivo “package.json”). Tenemos un ejemplo de las nuevas dependencias en la Fig. 4.2.117).

```

{
  "name": "appFrontend",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^15.2.3",
    "@angular/common": "^15.2.3",
    "@angular/compiler": "^15.2.3",
    "@angular/core": "^15.2.3",
    "@angular/fire": "^7.5.0",
    "@angular/forms": "^15.2.3",
    "@angular/platform-browser": "^15.2.3",
    "@angular/platform-browser-dynamic": "^15.2.3",
    "@angular/router": "^15.2.3",
    "firebase": "^9.18.0",
    "rxjs": "^7.5.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.11.4"
  }
}

```

Fig. 4.2.117: Modificación del archivo “package.json”, a la hora de añadir la libería de Firebase.

Paso 3: Ir al fichero “environment.ts”, que contiene el proyecto Angular, y colocar la configuración presente en la Fig. 4.2.116. Esto nos permitirá conectar el proyecto Angular a la base de datos de Firebase ubicada en la nube.

Si se dirige a las siguientes imágenes Fig. 4.2.118 y Fig. 4.2.119 puede ver la configuración que copiamos señalada, y cómo quedaría esta configuración ya incluida en el fichero “environment.ts”.

```

$ npm install firebase
Luego, inicializa Firebase y comienza a usar los SDK de los productos quequieres utilizar.

// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyjbdtucz1fY_f0nTxOFEFyxIS6aQBncc0",
  authDomain: "fir-5db01.firebaseio.com",
  projectId: "fir-5db01",
  storageBucket: "fir-5db01.appspot.com",
  messagingSenderId: "343342589998",
  appId: "1:343342589998:web:ac12fae7f0353a43d597bf",
  measurementId: "G-K6PJH1NGLV"
};

```

Fig. 4.2.118: Configuración dada por Firebase, necesaria para conectar Angular a la BBDD.

```

TS environment.ts
appFrontend > src > environments > TS environment.ts > ...
1 // This file can be replaced during build by using the 'fileReplacements' ...
2 // "ng build" replaces 'environment.ts' with 'environment.prod.ts'.
3 // The list of file replacements can be found in 'angular.json'.
4
5 export const environment = {
6   production: false,
7   firebaseConfig : {
8     apiKey: "AIzaSyCjbdtucz1fY_f0nTx0FEFyx1S6aQBncco",
9     authDomain: "fir-5db01.firebaseio.com",
10    projectId: "fir-5db01",
11    storageBucket: "fir-5db01.appspot.com",
12    messagingSenderId: "34334258998",
13    appId: "1:34334258998:web:a12fae7f0353a43d597bf",
14    measurementId: "G-K6PJHINGLV"
15  }
16};

```

Fig. 4.2.119: Configuración ya añadida al proyecto Angular, en el archivo “environment.ts”, la cual conecta Angular a la base de datos Firebase.

Paso 4: Cargar la nueva dependencia de Firebase en el fichero “app.module.ts” (Fig. 4.2.120).

- Esto nos permitirá poder usar las clases de esta nueva librería en todo el proyecto Angular.

```

TS app.module.ts
appFrontend > src > app > TS app.module.ts > AppModule
10
11 import { HttpClientModule } from '@angular/common/http';
12 import { AppRoutingModule } from './app-routing.module';
13
14 import { ReactiveFormsModule } from '@angular/forms';
15
16 import { AngularFireModule } from '@angular/fire/compat';
17 import { environment } from 'src/environments/environment';
18
19
20
21 @NgModule({
22   declarations: [
23     AppComponent,
24     PrimerComponenteFormularioLoginComponent
25   ],
26   imports: [
27     BrowserModule,
28     FormsModule,
29     ReactiveFormsModule,
30     HttpClientModule,
31     AppRoutingModule,
32     AngularFireModule.initializeApp(environment.firebaseioConfig)
33   ],
34   providers: [SimulacionbddService],
35   bootstrap: [AppComponent]
36 })
37 export class AppModule { }
38

```

Fig. 4.2.120: Incluida la nueva dependencia en todo el proyecto Angular.

Paso 5: Donde queramos usar la autenticación de nuestra base de datos Firebase, hacer la inyección de dependencias correspondiente (ver Fig. 4.2.121):

```

1  app.component.ts ✘
2  app/ frontend > src > app > app.component.ts ...
3  import { Component, OnInit } from '@angular/core';
4  import { FormGroup, FormBuilder, Validators } from '@angular/forms';
5  import { Router } from '@angular/router';
6  import { AngularFirestoreAuth } from '@angular/fire/auth';
7
8  @Component({
9   selector: 'app-root',
10  templateUrl: './app.component.html',
11  styleUrls: ['./app.component.css']
12 })
13 export class AppComponent {
14   title = 'appFrontend';
15
16   loading : boolean = false;
17   mostrarMensajeErrorRegistro : boolean = false;
18   mostrarMensajeExitoRegistro : boolean = false;
19   mensajeInfo : String = '';
20
21   loginUsuario : string = '';
22   correoUsuarioAlHijoComApp : string = "Datos del Comp.Padre enviados y escritos en el Comp.Hijo.";
23   datosRecibidosDelHijoComApp : string = "";
24
25   constructor(
26     private afAuth : AngularFireAuth,
27     private fb : FormBuilder,
28     private router : Router ) {
29     this.loginUsuario = this.fb.group({
30       email : ['', [Validators.required, Validators.email]],
31       password : ['', [Validators.required]],
32     });
33   }

```

Fig. 4.2.121: Inyección de dependencias para usar autentización en Firebase desde Angular.

Paso 6: Si queremos realizar un proceso de login desde Angular, usar la nueva dependencia proporcionada por Firebase e inyectada por constructor en la clase que se use. Puede verse un ejemplo en la Fig. 4.2.122, donde se usa dentro de un nuevo método que llamaremos “login()”:

```

1  login(){
2   const email = this.loginUsuario.value.email;
3   const password = this.loginUsuario.value.password;
4   this.loading = true;
5   this.afAuth.signInWithEmailAndPassword(email, password).then((user)=>{
6     console.log(user)
7     if(user.user?.emailVerified){
8       /*
9        * Recuperamos el JWT del usuario, recien conseguido al iniciar sesión
10       */
11      user.user.getIdToken().then(token=>{
12        this.router.navigate(['/dashboard']);
13      });
14    }
15    else{
16      this.router.navigate(['/verificar-correo']);
17    }
18  }.catch(error)=>{
19    this.loading = false;
20    console.log('error');
21    this.loading=false;
22    this.mostrarMensajeErrorRegistro=true;
23    this.mostrarMensajeExitoRegistro=false;
24    this.mensajeInfo=this.firebaseioError(error.code)
25  })
26
27  firebaseError(code:String){
28    switch(code){
29      // password invalida
30      case 'auth/invalid-password':
31        return 'Contraseña inválida'
32      // password incorrecta
33      case 'auth/wrong-password':
34        return 'Contraseña incorrecta'
35      // correo invalido
36      case 'auth/invalid-email':
37        return 'Correo inválido'
38      case 'auth/user-not-found':
39        return 'El usuario no existe'
40      // cualquier otro error no contemplado
41      default:
42        return 'Email/Password incorrecto'
43    }
44  }

```

Fig. 4.2.122: Realización del login en la base de datos, usando la nueva dependencia, con gestión de errores incluida.

Paso 7: Con todos estos pasos realizamos el login contra la base de Datos Firebase, gestionamos los errores personalizados y contemplamos la detección de errores propios aportados por Firebase.

Esto podría extenderse a procesos no solo de Login, sino también de Logout, SignUp, recuperación de password, etc.

q) Formularios Reactivos.

Partiendo de las 2 últimas imágenes (Fig. 4.2.121 y Fig. 4.2.122), tenga en cuenta 2 cosas:

- 1) En el constructor se introducen los campos de “email” y “password” como vacíos, y se indican los campos como “Validators.required”.
- 2) En el código se usa la dependencia “FormGroup”, la cual nos servirá para recuperar los datos de los campos del formulario que tengamos en el HTML.

Para realizar un formulario reactivo, el cual se asocie al fichero typescript anterior, realizar el código HTML de este componente, como el que se muestra en la siguiente Fig. 4.2.123.

```
<div class="wrapper">
  <div *ngIf="!loading" class="body">
    
    <h1>Login</h1>
    <form (ngSubmit)="login()" [formGroup]="loginUsuario">
      <span *ngIf="!mostrarMensajeErrorRegistro" class="alert alert-danger" *ngIf="!{mensajeInfo}*>{mensajeInfo}</span>
      <span *ngIf="mostrarMensajeErrorRegistro" class="alert alert-success" *ngIf="!{mensajeInfo}*>{mensajeInfo}</span>
      <div class="form-floating mb-3">
        <input type="email" formControlName="email" class="form-control" placeholder="name@example.com">
        <label>Email</label>
        <span *ngIf="!loginUsuario.get('email')?.hasError('required') && loginUsuario.get('email')?.touched" class="text-danger errorLabel">
          El <strong>correo</strong> es requerido.
        </span>
        <span *ngIf="!loginUsuario.get('email')?.hasError('email') && loginUsuario.get('email')?.touched" class="text-danger errorLabel">
          El <strong>formato del <strong>correo</strong> es incorrecto.
        </span>
      </div>
      <div class="form-floating mb-3">
        <input type="password" formControlName="password" class="form-control" placeholder="Password">
        <label>Password</label>
        <span *ngIf="!loginUsuario.get('password')?.hasError('required') && loginUsuario.get('password')?.touched" class="text-danger errorLabel">
          La <strong>nueva contraseña</strong> es requerida.
        </span>
      </div>
      <div class="d-grid gap-2">
        <button type="submit" class="btn btn-primary btn-lg" [disabled]=!loginUsuario.invalid>Entrar</button>
      </div>
      <p class="mt-3">
        <a href="/signup">Registrarse</a>
      </p>
      <a class="mt-3" href="/recuperar-password">Recuperar Contraseña</a>
    </form>
  </div>
</div>
```

Fig. 4.2.123: Código HTML necesario para el formulario reactivo, asociado a la clase Typescript de las imágenes Fig. 4.2.121 y Fig. 4.2.122.

La visualización en el navegador del código HTML creado en la figura 4.2.123 dará como resultado un formulario reactivo.

Puede verse el resultado de este formulario reactivo, visto desde el navegador, en la siguiente imagen (Fig. 4.2.124).

- Este formulario no tendrá CSS.
- Este formulario reactivo activa cada error asociado a su campo: cuando cliques en un campo y después cliques fuera sin haberlo llenado saltará el error, pero el error desaparece automáticamente cuando rellenes el campo correctamente.

The screenshot shows a 'Login' form with two fields: 'Email' and 'Password'. The 'Email' field contains 'name@example.com' and has a red border with the error message 'El correo es requerido.' (The email is required). The 'Password' field is empty and has a red border with the error message 'La nueva contraseña es requerida.' (A new password is required). Below the fields are buttons for 'Entrar' (Enter), 'Registrarse' (Register), and 'Recuperar Contraseña' (Reset Password).

Fig. 4.2.124: Resultado web del Formulario Reactivo en HTML.

4.3 Aplicación Final (Full Stack).

Ahora se presentará la aplicación final.

La información necesaria para lograr realizar la aplicación final será la ya aportada en tutoriales anteriores, además de la que se irá aportado sobre la marcha.

Para el Back-end se parte del código del apartado 4.1. Por otra parte, el Front-end se empezará con un proyecto Angular desde cero, al que incorporamos mejoras basadas en el apartado 4.2.

a) Incompatibilidades a las que te enfrentarás: solucionadas.

Caso A: incompatibilidad del paquete “Firebase”.

- Cuando deseamos conectar el frontend con la base de datos Firebase, lo normal sería instalar el módulo o paquete “Firebase” (ya sea con la instrucción “ng” o “npm”). El comando que inicialmente se prueba lo da la documentación de Firebase (que suele fallar habitualmente por errores de la conexión):
 - ❖ **npm install firebase**
- Alternativa: usar el módulo “Fire”, instalando el siguiente comando (siempre dentro de la carpeta del proyecto Angular):
 - ❖ **npm i --save firebase @angular/fire**
- Asegurarnos que la instalación no genera errores, y ya tendremos el módulo de Firebase instalado bajo “@angular/Fire”.

Caso B: errores al cargar CSS de Bootstrap, aun con instalación correcta.

- En muchas ocasiones, aunque Bootstrap se instale correctamente, no carga los CSS.
 - El comando de instalación de Bootstrap era este:
 - ❖ **npm install Bootstrap**
- Cuando teniendo la instalación terminada, no nos cargue los CSS llamados desde el HTML que toque, deberemos ir al archivo “angular.json” del proyecto Angular. En este archivo ir a la parte de “styles” bajo “architect”->“build” -> “options” -> “styles”.
- Ahora (en “angular.json”) deberá incluir la ruta de todos los CSS de Bootstrap. La ruta todos esos CSS es la siguiente:
 - ❖ **"node_modules/bootstrap/dist/css/bootstrap.min.css"**

- El caso ejemplo queda como se muestra a continuación (Fig. 4.3.1):

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "appFrontendTFM": {
      "projectType": "application",
      "schematics": {},
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/app-frontend-tfm",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": "src/polyfills.ts",
            "tsConfig": "tsconfig.app.json",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/styles.css",
              "node_modules/bootstrap/dist/css/bootstrap.min.css",
            ]
          }
        }
      }
    }
  }
}
```

Fig. 4.3.1: Añadidos los CSS de Bootstrap, al añadir la ruta que los contiene.

Caso C: errores al cargar CSS de Angular Material, aun con instalación correcta.

- Similar al caso B anterior, porque instalando correctamente el módulo de Angular Materials cuando intento usar uno de sus elementos en un HTML de cierto componente, no carga las CSS.
- Podría cargar el elemento genérico de un componente “AngularMaterial”, pero sin darle color al mismo por carecer del CSS.
- Debemos ir entonces de nuevo al archivo “angular.json” para incluir la ruta de los CSS (en la parte de “styles” bajo “architect”->“build” -> “options” -> “styles”).
- La ruta que debería incluirse es la siguiente:
 - ❖ [node_modules/@angular/material/prebuilt-themes/indigo-pink.css](#)
- Un ejemplo sobre código de la ruta añadida en “angular.js” sería el siguiente (Fig. 4.3.2):

```
angular.json > {} projects > {} appFrontendTFM > {} architect > {} build > {} options > [ ] styles > ↗ 2
1  {
2    "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
3    "version": 1,
4    "newProjectRoot": "projects",
5    "projects": {
6      "appFrontendTFM": {
7        "projectType": "application",
8        "schematics": {},
9        "root": "",
10       "sourceRoot": "src",
11       "prefix": "app",
12       "architect": {
13         "build": {
14           "builder": "@angular-devkit/build-angular:browser",
15           "options": {
16             "outputPath": "dist/app-frontend-tfm",
17             "index": "src/index.html",
18             "main": "src/main.ts",
19             "polyfills": "src/polyfills.ts",
20             "tsConfig": "tsconfig.app.json",
21             "assets": [
22               "src/favicon.ico",
23               "src/assets"
24             ],
25             "styles": [
26               "src/styles.css",
27               "node_modules/bootstrap/dist/css/bootstrap.min.css",
28               "node_modules/@angular/material/prebuilt-themes/indigo-pink.css"
29             ]
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
207
208
209
209
210
211
212
213
214
215
216
216
217
218
218
219
219
220
221
222
223
224
225
225
226
226
227
227
228
228
229
229
230
231
232
233
233
234
234
235
235
236
236
237
237
238
238
239
239
240
241
242
242
243
243
244
244
245
245
246
246
247
247
248
248
249
249
250
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
131
```

Caso D: errores al intentar ejecutar el proyecto de Angular.

- Éste sería el error producido en el terminal de comandos, al intentar arrancar el servidor Forntend en Local:
 - “*Could not find the implementation for builder @angular-devkit/build-angular:dev-server*”
- El error normalmente suele ocurrir cuando movemos código de rama en git y este está en una versión deprecated, o cuando existe una versión más nueva y estamos corriendo una versión vieja.
- La siguiente URL explica claramente cómo podemos corregir el error, y así volver a lograr que Angular arranque correctamente sin tener que perder el código conseguido.
 - <https://stackoverflow.com/questions/56623458/could-not-find-the-implementation-for-builder-angular-devkit-build-angulardev>
- En mi caso lo que me permitió arreglar este error fueron los siguientes comandos y en este orden:
 - **npm install @angular-devkit/build-angular** → Primera propuesta, FAIL.
 - **npm uninstall @angular-devkit/build-angular** → Borrado del módulo, OK.
 - **npm install --save-dev @angular-devkit/build-angular** → reinstall módulo, FAIL.
 - **npm uninstall @angular-devkit/build-angular** → Volvemos a borrar módulo, OK.
 - **ng update @angular/cli @angular/core --allow-dirty -force** → limpiamos caché de forma forzosa, OK.
 - **npm install @angular-devkit/build-angular@14** → Vuelvo a instalar mi módulo, pero terminando el comando con “@14” porque vi que en fichero “angular.json” indica que mi versión de Angular es la 14.X.x

En mi caso con el último comando resolví el problema del arranque del servidor local (que se ejecuta con “ng serve”).

b) Metodología CI/CD en proyectos Full Stack.

Como las técnicas de subida a producción con CD/CI no es tema de este trabajo, no se creará una arquitectura real para CD/CI.

Sin embargo, dejo creado un grupo de ramas (al menos 3), que servirán para darnos una idea de los que en el futuro podría ser usado para implementar arquitecturas CD/CI.

Se detallan 4 ramas:

- main
- master
- feature/app-Front-definitiva
- feature/app-Backend-definitiva

En nuestro caso de ejemplo, la rama “main” podrá mergearse con las siguientes: “app-Front-definitiva” y “app-Front-definitiva”. No pasará lo mismo con la rama “master”, la cual queda siempre aislada de todas las demás sin unirse nunca con otra rama.

Veamos la siguiente imagen para que nos sirva de aclaración (Fig. 4.3.3):



Fig. 4.3.3: Ejemplo de ramas git utilizadas, vistas en un flujo de trabajo habitual.

La funcionalidad que se debería seguir es la siguiente:

- Todo el código de la rama “main” está en producción (entorno PRO).
- Todo el código de la rama “master” está en entornos DES o PRE-productivos.
- Todo el código de la rama “feature/app-Front-definitiva” será de la parte de Angular, y estará en proceso de desarrollo. Cuando lance un commit de código no estable, el commit empezará por “WIP” (work in progress), y si el commit es estable empezará por “Estable”.
- Todo el código de la rama “feature/app-Backend-definitiva” será de la parte de Sprint Boot, y estará en proceso de desarrollo. Cuando lance un commit de código no estable, el commit empezará por “WIP” (work in progress), y si el commit es estable empezará por “Estable”.
- Cuando la rama “feature/app-Front-definitiva” o “feature/app-Backend-definitiva” esté estable se copia/pega todo su código en “master”. En máster se

probará idealmente sobre un servidor y base de datos similar a la que tengamos en el entorno de PRO.

- Si las pruebas en rama “master” son positivas, se considerará adecuado llevar el código entonces a entorno productivo (PRO). Para hacer esto, mergear “feature/app-Front-definitiva” o “feature/app-Front-definitiva” sobre la rama “main”.

Puede encontrar algo más información sobre la metodología CI/CD en el ANEXO I.

c) Arquitectura de la aplicación Full Stack.

- **Arquitectura general:**

El esquema de la arquitectura general de toda la aplicación Full Stack es como se muestra en la Fig. 4.3.4.

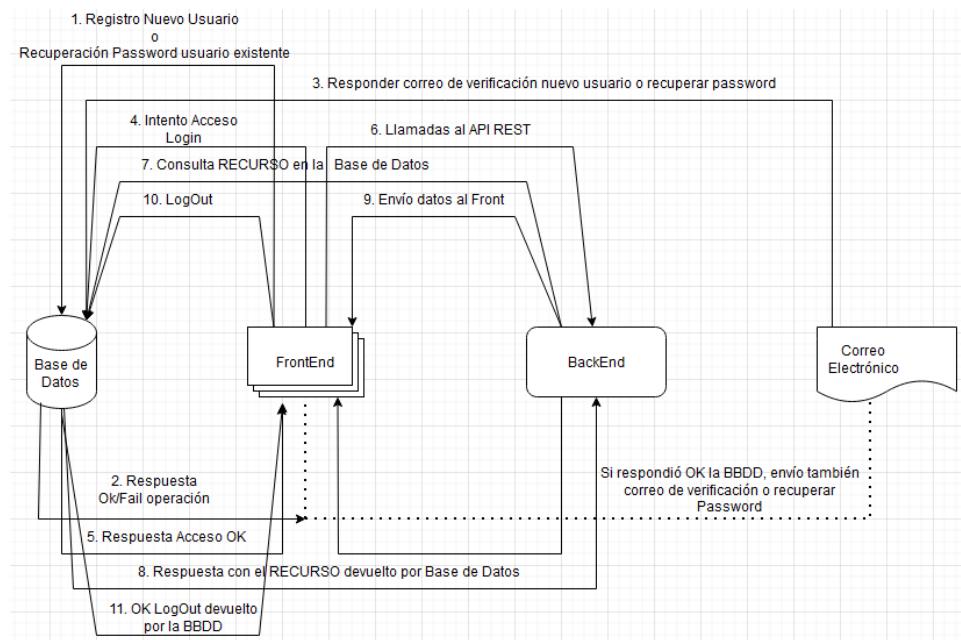


Fig. 4.3.4: Esquema arquitectura general web completa.

- **Arquitectura esquemática del correo electrónico:**

El esquema del envío de mensajes entre el correo electrónico y la base de datos puede verse reflejado en la Fig. 4.3.5.

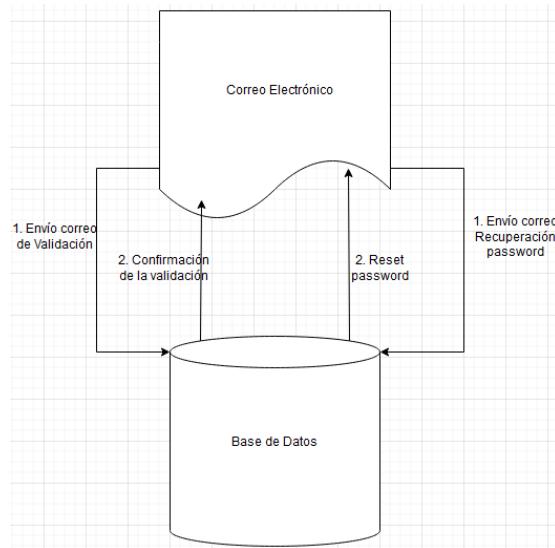


Fig. 4.3.5: Esquema arquitectura de comunicaciones por correo electrónico

- **Estructura de llamadas REST en el BackEnd:**

La aplicación Backend ya conocida, pero modificando las llamadas actuales del API REST, y ampliando el nº de las mismas.

Las llamadas que finalmente definitivas serán las siguientes:

- Llamada 1. `getCatalogo()`
 - Sin inputs.
 - Output: todas las películas/series existentes, ordenadas por id.
- Llamada 2. `getPeliculasAccion()`
 - Sin inputs.
 - Output: 200 OK, todas las películas con género Acción, ordenadas por id.
- Llamada 3. `getPeliculasComedia()`
 - Sin inputs.
 - Output: 200 OK, todas las películas con género Comedia, ordenadas por id.

- Llamada 4. getPelículasDrama()
 - Sin inputs.
 - Output: 200 OK, todas las películas con género Drama, ordenadas por id.
- Llamada 5. getPelículasTendencia()
 - Sin inputs.
 - Output: 200 OK, todas las películas en tendencia, ordenadas por id.
- Llamada 6. getSeries()
 - Sin inputs.
 - Output: 200 OK, sólo todas las series del catálogo, ordenadas por id.
- Llamada 7. getDatosUsuarioActual()
 - Input: correo de usuario actual.
 - Output: 200 OK, todos los datos del usuario actual, para modo lectura.
- Llamada 8. updateDatosUsuarioActual()
 - Inputs: correo de usuario actual, y todos los datos actualizados.
 - Output: OK 204, sin datos devueltos.
- Llamada 9. deleteUsuario()
 - Input: correo de usuario actual.
 - Output: 204 OK, sin datos devueltos tras eliminar usuario de tabla “Usuarios”.
- Llamada 10 createUsuario()
 - Input: body JSON con todos los datos disponibles.
Output: 200 OK, devuelve el Timestamp que será identificador de contenido.
- Llamada 11 createContenido()
 - Input: body JSON con todos los datos disponibles.
Output: 200 OK, devuelve el Timestamp que será identificador de contenido.

Nota:

Cuando creo un usuario durante un nuevo registro, este se añade en la tabla de autenticación y en la tabla de usuarios.

Sin embargo, cuando borro ese usuario solo se borra éste de la tabla de usuarios, y cuando vuelva a intentar pasar el login no podrá si el usuario no existe en ambas tablas (“usuarios” y “autenticación”).

Tampoco podrá volver a usarse un correo conocido de una cuenta eliminada para crear otra cuenta en la web.

- Arquitectura del FrontEnd:

En el apartado 4.3.e, donde se explicará una implementación primitiva del Login, puede verse las distintas pantallas del login necesarias, y su funcionamiento.

Una vez entremos al interior de la aplicación web, tras pasar el login, debería de ser una página idéntica en visualización durante toda la experiencia, pero que muestre un contenido distinto en el body según el link clicado (ya sea respecto a los contenidos o respecto a la información de usuario).

- Arquitectura de la Base de Datos Firebase:

Las tablas que usaremos en Firebase serán 3: “Autenticación de usuarios” (creada automáticamente durante el proceso de Login), “Usuarios” y “Contenidos”.

Como observación, aclarar que tanto “Usuarios” como “Contenidos” no son tablas relacionales, sino Colecciones de Datos (no relacionales).

Separando el uso de la aplicación en 3 partes, cada tabla tendría las siguientes implicaciones:

1. User Authentication: Utilizada para el proceso de Login y Logout, con un identificador que corresponde con el correo electrónico utilizado.
2. Usuarios: colección de datos que contiene en cada elemento “usuario”, nombrado por correo electrónico, todos los atributos de usuario.
3. Contenidos: colección de datos que contiene en cada elemento “contenido”, nombrado por su “id” de contenidos (id = TimeStamp de creación).

Las tablas finales quedarían como se muestran de las Figuras 4.3.6 a la 4.3.8.

The screenshot shows the Firebase Authentication console under the 'Users' tab. At the top, there are navigation links: 'Users' (which is active and underlined), 'Sign-in method', 'Templates', 'Usage', 'Settings', and 'Extensões NUEVA'. Below the navigation bar is a search bar labeled 'Buscar por dirección de correo electrónico, número de teléfono o UID de usuario'. To the right of the search bar are buttons for 'Agregar usuario' (Add user) and a three-dot menu icon. The main area displays a table with columns: 'Identificador', 'Proveedores', 'Fecha de creación', 'Fecha de acceso', and 'UID de usuario'. There are two rows of data in the table. The first row shows an email address (@.com) and was created on 9 abr 2023, last accessed on 9 abr 2023. The second row shows another email address (@.co...) and was created on 19 mar 2023, last accessed on 9 abr 2023. At the bottom of the table, there are pagination controls: 'Filas por página: 50', '1 - 2 of 2', and navigation arrows.

Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario
@.com		9 abr 2023	9 abr 2023	
@.co...		19 mar 2023	9 abr 2023	

Fig. 4.3.6: Tabla de Autentificación de Usuarios, para el Login en Firebase.

Fig.4.3.7: Tabla de Contenidos, de Cloud Firestore.

Fig.4.3.8: Tabla de Usuarios, de Cloud Firestore.

d) Seguridad de la aplicación Full Stack.

Ahora vamos a tratar el tema de la seguridad mínima que debería incorporar una aplicación web, y las partes que deberían asegurarse, que como hemos visto ya, la aplicación web consta de una base de datos NO-SQL, un BackEnd y un FrontEnd.

Se da por supuesto que, si lanzamos la aplicación a producción, estas 3 partes claramente diferenciadas se colocarán en ubicaciones remotas separadas.

Por parte de la base de datos sabemos que se contratará con Google, ya que es Firebase. Sin embargo, en cuanto al BackEnd y al FrontEnd tenemos más libertad de elección: servicios cloud de Microsoft, Amazon Web Service, Google, OpenShift o incluso

GitHubActions. Si queremos realizar un despliegue a producción más sencillo, optaremos por tecnologías como Rancher.

En cualquier caso, al elegir un fabricante para lanzar nuestra web a producción no salirse de los grandes fabricantes. De lo contrario, una futura falla de seguridad o caída de servicio podría repercutir muy negativamente a nuestra imagen de marca, y esto no tendría tanta repercusión con los grandes fabricantes.

Una vez que tenemos claro que nuestra aplicación web se divide en 3 grandes partes y los fabricantes que debemos elegir para alojar cada parte, ahora toca abordar cómo asegurar cada una de esas 3 partes, explicar la tecnología JWT y dar una perspectiva global de la seguridad de la aplicación.

- **Asegurar la Base de Datos (implementado):**

- Colocar las reglas de acceso a la Base de Datos adecuadas, no autorizando el acceso a los datos para una petición desde la API REST por un usuario que no esté logeado, o no permitiendo la lectura de un documento del usuario X por un usuario Y.
- No subir a GitHub ni otro medio online las claves proporcionadas en la sección de configuración del SDK de Firebase, tales como: apiKey, authDomain y projectId. En caso de publicar algún dato de estos 3, podrían conectarse a nuestra base de datos sin permiso y ésta sería vulnerable.

*Vaya al [Anexo II](#) para ver los links de la documentación sobre configuración de reglas de acceso con Firebase (documentación oficial).

- **Asegurar el FrontEnd (implementado):**

- No subir a GitHub ni otro medio online las claves proporcionadas en la sección de configuración del SDK de Firebase, tales como: apiKey, authDomain y projectId. En caso de publicar algún dato de estos 3, podrían conectarse a nuestra base de datos sin permiso y ésta sería vulnerable.
- Configurar correctamente la seguridad de las páginas disponibles/rechazadas para usuarios logeados, así como para la alternativa cuando el usuario no esté logeado. Para ello, Angular cuenta con la herramienta de los “Guardianes”.
- Inicio del proceso de registro y acceso a la aplicación web desde el Front-End.
- Cuando Firebase verifique el acceso exitosamente, se devolverá el token JWT desde Firebase con la información del usuario logeado.
- Uso del token JWT devuelto por Firebase para mantener vivos los permisos de acceso a las páginas internas de la web (vigiladas por Guardianes).
- Uso del token JWT devuelto por Firebase para conseguir la información del usuario logeado. Después, el propio JWT se mandará a la API REST desde el

Front-End, dónde se consultarán datos a Firebase, y acorde a las reglas de la base de datos esta devolverá los datos solicitados o no.

• **Asegurar el BackEnd (no implementado):**

- No subir a GitHub ni otro medio online las claves proporcionadas en la sección de configuración del SDK de Firebase, tales como: apiKey, authDomain y projectId. En caso de publicar algún dato de estos 3, podrían conectarse a nuestra base de datos sin permiso y ésta sería vulnerable.
- Configurar las direcciones de los orígenes que el BackEnd debe considerar de confianza. Por eso, al BackEnd se añade la dirección del FrontEnd para que no rechace las peticiones que les llegue desde el Front.
- Uso de librerías Java “Spring Security” y “OAuth2 Client”, que permiten asegurar los endpoint del BackEnd.
- Creación de una clase en Java que se llame por ejemplo “SecurityConfig”, donde crearemos varios @Bean, los cuales se usarán para configurar la seguridad de la aplicación Backend. En esta clase configuraremos todas las funciones de seguridad del BackEnd.
- Crear otra clase en Java, a la que llamaremos “KeyGeneratorUtils”, la cual nos permitirá crear un JWT con la información que queramos incluir en el interior de ese JWT. Para generar el JWT es necesario hacer uso de la clase previa “SecurityConfig” y generar una clave privada. Con la clave privada podré crear entonces el JWT.
- Para iniciar, se propone crear 2 servicios asociados a 2 controllers del API REST: el 1º que haga Login con user/pass y devuelva el JWT generado, el 2º que permita obtener los datos que queremos consumir de la base de datos si se logea con el JWT correcto que se consiguió en del primer servicio.
- Configurar el código Front-End para utilizar “interceptores” en Angular.

*Del Back-End solo se configuran los orígenes de confianza, pero no se llega a asegurar la API REST en el proyecto (excede mucho el tamaño del proyecto).

**Vaya al [ANEXO II](#) para conseguir la documentación recomendada para asegurar la API REST correctamente.

• **Funcionamiento de la tecnología JWT:**

- Un JWT es un JSON Web Token, y tiene 3 partes importantes, que cada una de ellas vienen separadas por un punto:
 - Encabezado: indica el tipo de cifrado.
 - Payload: datos de usuario, uid, etc.
 - Firma del JWT: verificación, firma de autenticidad del servidor que verifica la veracidad de este JWT, etc.

- Con JWT cubrimos tanto la parte de autenticidad como privacidad.
- Un JWT no puede modificarse y seguir siendo válido: el servidor verificador devolvería que no reconoce ese JWT.
- Mientras no se tenga la clave privada del verificador JWT nadie podrá firmar nuevos tokens (ni falsearlos).
- Cuidado con la cantidad de datos que ponemos en el JWT, un exceso de datos podría saturar el canal. Es crítico el peso de este JWT.
- Los certificados de JWT no pueden invalidarse, pero podríamos comparar por fecha de expiración de ese JWT en el verificador mediante creación de tablas de estados. Sin embargo, no interesa tampoco esta solución, ya que requiere sistemas centralizados, y eso presenta problemas de rendimiento para entornos con micro-servicios.
- Lo más correcto es no enviar los datos mínimos en el JWT, como hemos hecho nosotros: el uid de Firebase, correo electrónico y poco más.
- Existen herramientas online para desencriptar un JWT: <https://jwt.io/>

*Puede encontrar en el [ANEXO II](#) un ejemplo de la información decodificada de un JWT.

e) Implementación primitiva del Login en el Front-End.

Como se vio en el apartado C, en esta aplicación web se inicia el proceso del login contra la base de datos directamente desde el FrontEnd. Por ese motivo, ahora se explicará con cierto detalle la funcionalidad de cada pantalla implicada en el Login.

El proceso de Login consta de 3 pantallas: login de acceso a la web, registro de usuarios y recuperación de contraseña (las 2 últimas envían un correo electrónico real para poder terminar su funcionalidad).

Los formularios utilizados en las 3 pantallas serán formularios reactivos, es decir, reaccionarán con mensajes de alerta en tiempo real ante cambios en el contenido de sus campos.

Pantalla 1. Acceso a la Web:

- Inicialmente sin alertas (Fig. 4.3.6):



Fig. 4.3.6: Pantalla Login, visualización inicial.

- Generación de alertas por mensaje en tono rojo, cuando se clica en un campo y después clicamos fuera sin rellenarlo (Fig. 4.3.7):

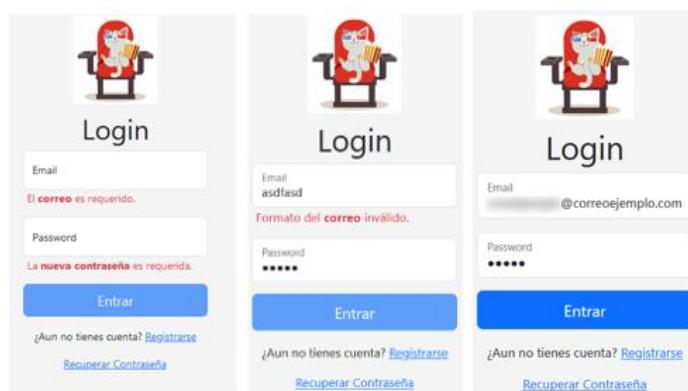


Fig. 4.3.7: Pantalla Login, con errores en formularios reactivos.

- Alerta de error enviado desde Firebase al ser rechazado un intento de acceso (Fig. 4.3.8):

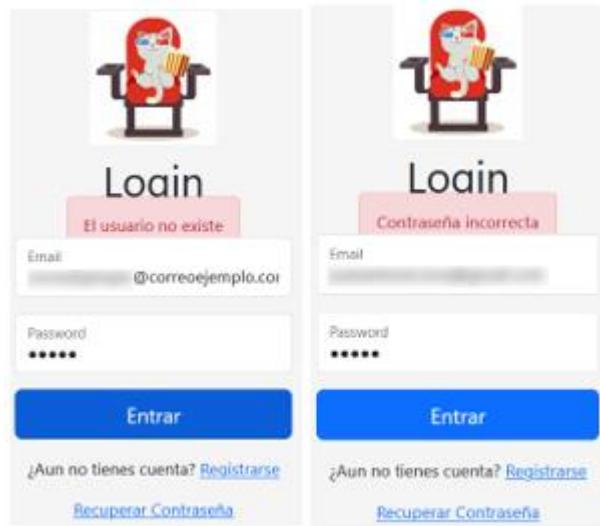


Fig. 4.3.8: Pantalla Login, con errores devueltos desde Firebase.

- Acceso a la web, pero sin haber verificado el correo electrónico previamente (Fig. 4.3.9):



Fig. 4.3.9: Acceso web aceptado, pero correo no verificado.

- Acceso a la web, con todas las funcionalidades activas tras haber verificado el correo (Fig. 4.3.10):



Fig. 4.3.10. Interior de la web al tener verificado el correo.

Pantalla 2. Registro de Nuevos Usuarios:

- Inicialmente, sin alertas (Fig. 4.3.11):



Registro

Email

Password

Repetir Password

Registrarme

[Volver Atrás](#)

Fig. 4.3.11. Pantalla de registro de usuarios vacía.

- Con errores en el formulario reactivo (Fig. 4.3.12):



Registro

Registro

Email
inventado@inventado.z
El **correo** es requerido.

Password

La **nueva contraseña** de tener al menos 6 caracteres.

Repetir Password
Repetir contraseña es requerido.

Registrarme

[Volver Atrás](#)

Registrarme

[Volver Atrás](#)

Fig. 4.3.12: Pantalla de registro de usuarios, con errores en formulario reactivo.

- Con errores recibidos desde Firebase, al rechazar el alta del nuevo usuario (Fig. 4.3.13):

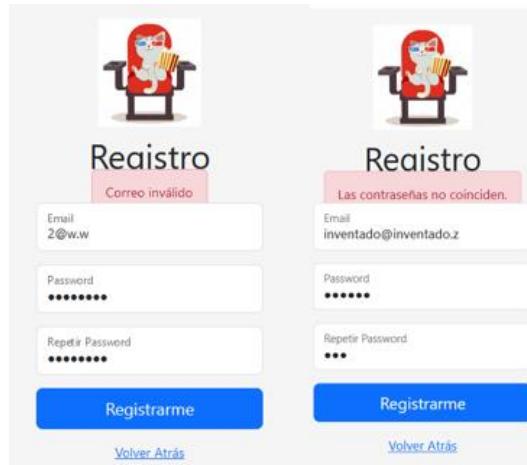


Fig. 4.3.13: Pantalla de registro de usuarios, con errores enviados por Firebase.

- Éxito devuelto desde Firebase, alta realizada del nuevo usuario (Fig. 4.3.14):

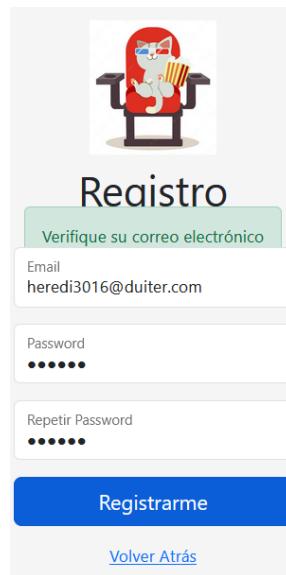


Fig. 4.3.14: Pantalla de registro de usuarios, éxito en el registro del nuevo usuario.

- Aparición de Spinner dinámico mientras se procesa la petición en la Base de Datos (Fig. 4.3.15):

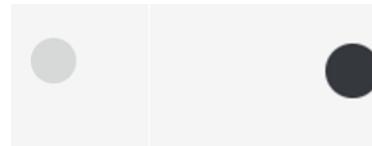


Fig. 4.3.15: Capturas en 2 fases del Spinner de procesamiento en curso.

- Verificación de correo electrónico, plantilla (Fig. 4.3.16):

The screenshot shows the Firebase Authentication console. On the left, the 'Authentication' tab is selected. In the center, under 'Plantillas' (Templates), the 'Verificación de dirección de correo...' (Email verification) template is highlighted with a green border. To the right, the detailed configuration for this template is shown, including fields for 'Nombre del remitente' (From name), 'De' (From), 'Responder a' (Reply-to), 'Asunto' (Subject), and 'Mensaje' (Message). The message body contains placeholders like '%DISPLAY_NAME%' and a link to verify the email address.

Fig. 4.3.16: Plantilla del correo de verificación.

Pantalla 3. Recuperación de Contraseña (usuario existente):

- Inicialmente, sin alertas (Fig. 4.3.17):



Fig. 4.3.17: Pantalla Recuperación password, inicialmente.

- Con errores en el formulario reactivo (Fig. 4.3.18):

Two side-by-side screenshots of the password recovery screen, demonstrating reactive form validation. Both screens feature the same layout: a cat character at the top, the text "Recuperar contraseña" in the center, an "Email" input field below it, a blue "Recuperar" button, and a "Volver Atrás" link at the bottom. The left screenshot shows an empty "Email" field with the error message "El correo es requerido." displayed below it. The right screenshot shows an "Email" field containing "123" with the error message "Formato del correo inválido." displayed below it.

Fig. 4.3.18: Pantalla Recuperación password, con errores formularios reactivos.

- Con errores recibidos desde Firebase, al rechazar el alta del nuevo usuario (Fig. 4.319):



Fig. 4.3.19: Pantalla Recuperación password, con errores devueltos por Firebase.

- Éxito devuelto desde Firebase, alta realizada del nuevo usuario (Fig. 4.3.20):



Fig. 4.3.20: Pantalla Recuperación password, éxito en la operación.

- Aparición de Spinner dinámico mientras se procesa la petición en la Base de Datos (Fig. 4.3.21):



Fig. 4.3.21: Capturas en 2 fases del Spinner de procesamiento en curso.

- Correo de restablecimiento de contraseña, plantilla (Fig. 4.3.22):

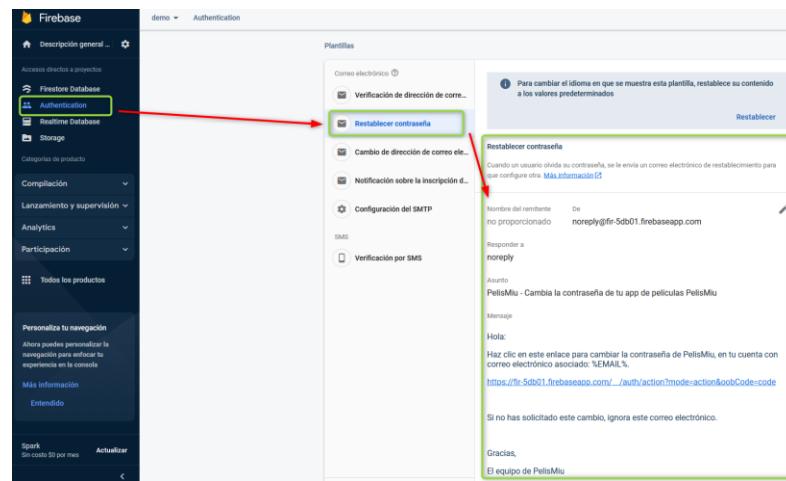


Fig. 4.3.22: Plantilla del correo para restablecimiento de contraseña.

f) Implementación de la conexión entre cliente Angular y API REST Java.

En el apartado 4.1 teníamos un proyecto en el Back-end donde creamos la API REST. Era esta API REST la que realiza las llamadas REST y rebía las peticiones generadas por un cliente DevOp llamado “Insomnia Core”.

Ahora ha llegado el momento de sustituir el cliente DevOp por un cliente real: el front-end realizado en Angular.

Para conectar el back-end con Angular, seguir los siguientes pasos:

Paso 1: Copiamos, en el directorio de la aplicación final, el proyecto Back-end del tutorial 4.1. La carpeta de la aplicación final debe contener ahora 2 proyectos: el de back y el de front. Sírvase de ejemplo la Fig. 4.3.23.

Escritorio > TFM > Código > Aplicación Final				
Nombre	Fecha de modificación	Tipo	Tamaño	
Backend	19/03/2023 13:40	Carpeta de archivos		
Frontend	19/03/2023 13:30	Carpeta de archivos		

Fig. 4.3.23: Nueva estructura de la Aplicación final con los 2 proyectos.

Paso 2: Cargar en Visual Studio Code los 2 proyectos de forma separa, de manera que se añadan al workspace primero la carpeta “Backend” y después “Frontend (no cargar el directorio padre de ambas para evitar errores futuros con VSC). Ver Fig. 4.3.24.

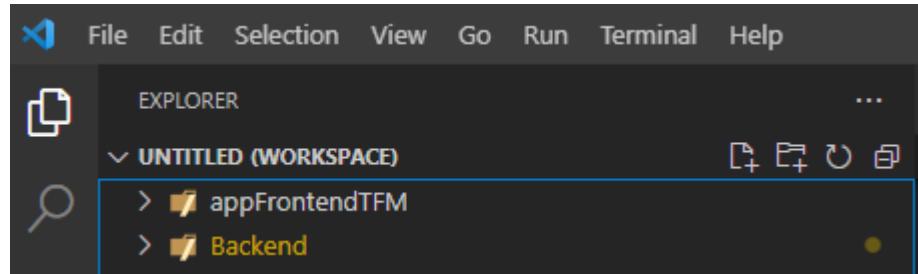


Fig. 4.3.24: Cargados los 2 proyectos individualmente en VSC.

Paso 3: Arranca la aplicación de Backend en Visual Studio Code, y abriendo el cliente “Insomnia Core” probaremos que todas las consultas REST siguen funcionando correctamente. Véase Fig. 4.3.25 y Fig. 4.3.26.

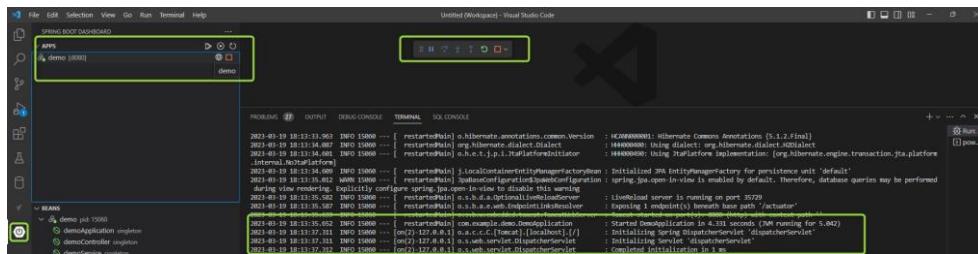


Fig. 4.3.25: Arrancada app Backend sin error en VSC.

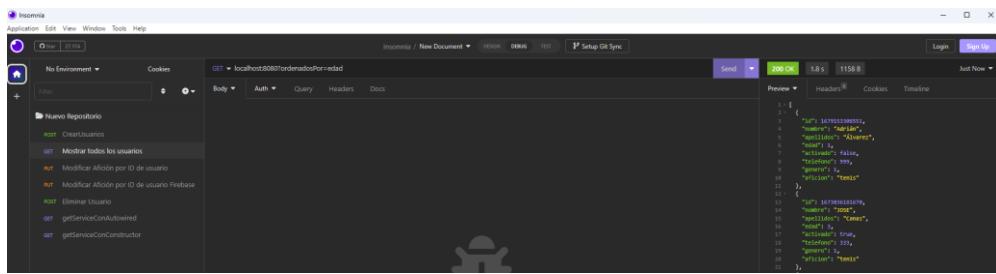


Fig. 4.3.26: Realización de llamada REST de AllsUsers, desde InsomniaCore.

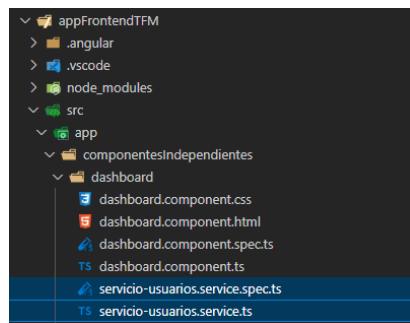
Paso 4: Ahora nos debemos asegurar que el proyecto FrontEnd sigue funcionando correctamente desde el Visual Studio Code. Para ello, abrir una terminal desde el IDE, ejecutar el comando “ng serve” y probar que la web sigue arrancando en el navegador correctamente (en <http://localhost:4200>).

Paso 5: Ahora crearemos un servicio Angular, el cual realizarán las llamadas al API REST Java. En este caso llamaremos el nuevo servicio Angular “servicioUsuarios”, que irá dentro del componente “Dashboard”.

Para instalar el nuevo servicio Angular, ejecutar el siguiente comando en la terminal de VSC:

**ng generate service componentesIndependientes/dashBoard/servicio-
usuarios**

Podemos ver los archivos del nuevo componente instalado en la Fig. 4.3.27:



4.3.27: Servicio Angular que conecta con API REST.

Paso 6: Crear un objeto de tipo “Usuario” en Angular, donde solo pondremos las propiedades que devuelve la llamada get de todos los usuarios por Insomnia Core Si requiere inicializar los valores, ponerlos a null o vacíos. Ver Fig. 4.3.28:

```
export class Usuario {
  id:number=0;
  nombre:string="";
  apellidos:string="";
  edad:number=0;
  activado:boolean=false;
  telefono:number=0;
  genero:number=-1;
  aficion:string="";
}

constructor()
```

Fig. 4.3.28: Clase TypeScript “usuario.ts”, para modelo de datos Usuario que existe en el Back.

Paso 7: Configurar el uso del nuevo servicio Angular en “app.module.ts”. Usar la librería de Angular llamada “HttpClient”: importamos “HttpClientModule” en “app.module.ts”. No olvide guardar los cambios realizados. Ver Fig. 4.3.29:

```

File Edit Selection View Go Run Terminal Help
... app.module.ts ...
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { AppComponent } from './app.component';
4 import { DashboardComponent } from './componentesIndependientes/dashboard/dashboard.component';
5 import { ErrorComponent } from './componentesIndependientes/error/error.component';
6 import { LoginComponent } from './componentesIndependientes/login/login.component';
7 import { RecuperarPasswordComponent } from './componentesIndependientes/recuperar-password/recuperar-password.component';
8 import { RegistrarsuarioComponent } from './componentesIndependientes/registro-usuario/registro-usuario.component';
9 import { SpinnerComponent } from './shared/spinner/spinner.component';
10 import { environment } from 'src/environments/environment';
11 import { MatMenuModule } from '@angular/material/menu';
12 import { MatButtonModule } from '@angular/material/button';
13 import { CookieService } from 'ng2-cookieservice';
14 import { GuardiansNoLogeado } from './componentesIndependientes/login/guardian-si-no-logeado';
15
16 import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent,
    DashboardComponent,
    ErrorComponent,
    LoginComponent,
    RecuperarPasswordComponent,
    RegistrarsuarioComponent,
    SpinnerComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    AngularFireModule.initializeApp(environment.firebaseioConfig),
    BrowserAnimationsModule,
    MatMenuModule,
    MatButtonModule,
    HttpClientModule
  ]
})

```

Fig. 4.3.29: Configuración de nuevo servicio en “app.mudule.ts”.

Paso 8: Crear la implementación del nuevo servicio, el cual hará las funciones de cliente REST anteriormente realizadas por “Insomnia Core”. Como ya hemos creado el objeto “Usuario” y cargado la librería “HttpClient” en app.module, podremos usarla en la implementación del nuevo servicio sin problemas. Vea el diseño del servicio propuesto en la Fig. 4.3.30:

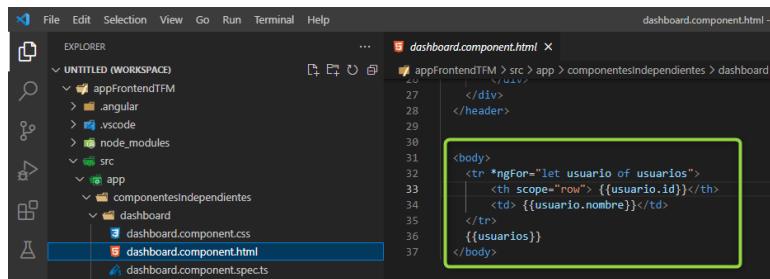
```

servicio-usuarios.service.ts
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { Usuario } from './usuario';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9
10 export class ServicioUsuariosService {
11
12   private url:string = "http://localhost:8080"
13
14   constructor(private http:HttpClient) { }
15
16   getAll():Observable<Usuario[]>{
17     console.log("entro en el servicio");
18     return this.http.get<Usuario[]>(this.url+"?ordenadosPor=edad");
19   }
20 }

```

Fig. 4.3.30: Diseño del servicio Angular que llamará al API REST.

Paso 9: Probar visualmente la llamada realizada (getAll) desde Angular al API REST que teníamos en Java. Realizarlo en el Front con un código HTML dentro del body del componente Dashboard. El HTML debe cargar los valores de id y nombre de todos los usuarios existentes en la BBDD. Para ello podemos hacer uso de un bucle FOR de Angular y los elementos HTML tr, th y td dentro del body. Véase la Fig. 4.3.31:



```

<body>
  <tr *ngFor="let usuario of usuarios">
    <th scope="row"> {{usuario.id}}</th>
    <td> {{usuario.nombre}}</td>
  </tr>
</tbody>
</table>

```

Fig 4.3.31: Código HTML que muestra los datos obtenido de la llamada getAll().

Paso 10: Cuando guarde todos los cambios realizados en este apartado, pare tanto el servidor de backend como el de frontend. Posteriormente reactive ambos servidores.

Paso 11: Pase el acceso a la web desde el login correctamente, hasta llegar al Dashboard. Note que inicialmente no nos traerá los datos correctamente (se verá objeto vacío o no verá nada). Al abrir la consola del navegador nos aparecerán errores.

Estos errores visuales y desde la consola del navegador son buena señal: entro al servicio, aunque aún no tenga acceso a los datos. Ver Fig. 4.3.32 para más detalles.

Los errores presentes en el frontend nos están informando de algo llamado “solicitud de origen cruzado bloqueada”, esto quiere decir que el back está bloqueando la respuesta por no considerar la dirección de origen de su confianza.

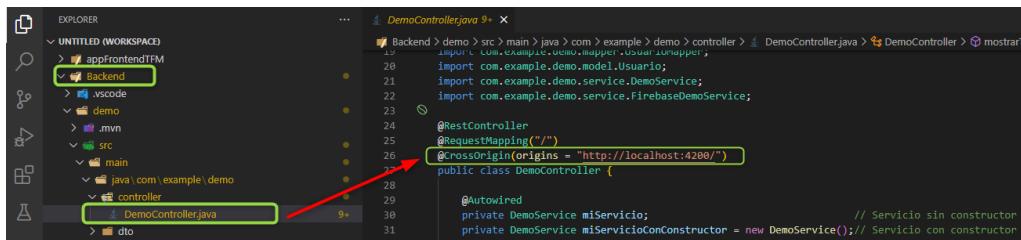


Fig. 4.3.32: Datos erróneamente visualizados (o no visibles) y error en consola del navegador.

Paso 12: Configurar el Backend para que reconozca como segura a la dirección de origen que corresponde con la del Front-end.

Por eso, ir a la clase “DemoController.java” del Backend y añadir esta línea justo encima de la definición de la clase: @CrossOrigin(origins = "http://localhost:4200/").

Tenemos un ejemplo en la Fig. 4.3.33:



```
... DemoController.java 9+ ...
Backend > demo > src > main > java > com > example > demo > controller > DemoController.java > DemoController.java > mostrarTodos
17 import com.example.demo.mapper.UsuarioMapper;
18 import com.example.demo.model.Usuario;
19 import com.example.demo.service.DemoService;
20 import com.example.demo.service.FirebaseDemoService;
21
22 @RestController
23 @RequestMapping("/")
24 @CrossOrigin(origins = "http://localhost:4200/")
25
26 public class DemoController {
27
28     @Autowired
29     private DemoService miServicio;
30
31     private DemoService miServicioConConstructor = new DemoService(); // Servicio sin constructor
32     private DemoService miServicioConConstructor2 = new DemoService(); // Servicio con constructor
33 }
```

Fig. 4.3.33: Configuración de orígenes seguros en el BackEnd.

Paso 13: Apague el servidor Backend y vuélvalo a encender correctamente. Si ahora sigue apareciendo el error anterior solo podría deberse a que no está encendido el Backend.

Paso 14: Recargar la página del DashBoard que ejecuta el Front. Ahora si deberíamos ver correctamente una lista de todos los ID's y nombres de todos los usuarios que tiene la base de datos. Podríamos tener un caso como el de la Fig. 4.3.34.

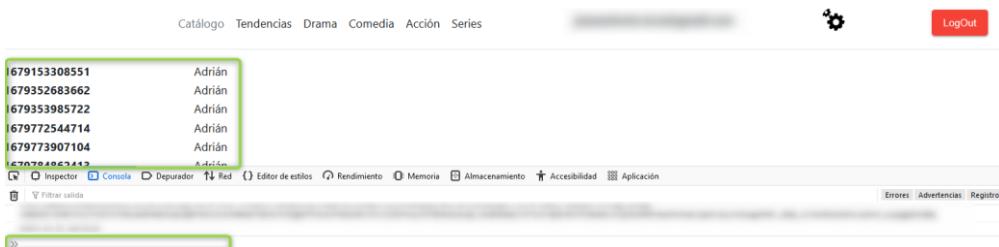


Fig. 4.3.34: Datos del GetAll() visualizados correctamente.

Con esto ya hemos dejado conectado el Front al Back correctamente.

Si deseamos conectar otro tipo de llamada ajena a la de tipo GET, puede servirse de los siguientes ejemplos en la Fig. 4.3.35 (operaciones Crear, Actualizar y Eliminar).

```

36 // CREAR USUARIO NUEVO (retorna el id como string)
37 crearUsuario(usuarioDTO:any, idNuevoUsuario:string): Observable<any> {
38 | return this.http.post<ResponseString>(this.url+"/usuarios/crearusuario/"+idNuevoUsuario, usuarioDTO);
39 }
40
41 // ACTUALIZAR USUARIO (no retorna nada OK 204)
42 actualizarUsuario(usuarioDTO:any, idUsuario:string):Observable<any> {
43 | return this.http.put(this.url+"/usuarios/actualizarUsuario/"+idUsuario, usuarioDTO);
44 }
45
46 // ELIMINAR USUARIO (no retorna nada OK 204)
47 eliminarUsuario(id:string): Observable<any> {
48 | return this.http.post(this.url+"/usuarios/eliminarUsuario/"+id, "");
49 }

```

Fig. 4.3.35: Ejemplos con POST/PUT, para conectar Front con Back en el resto de operaciones.

g) Web final terminada: resultados de codificación.

Partimos de las siguientes premisas de codificación previa:

- Ya se ha creado y configurado completa el SDK de Firebase en el proyecto BackEnd (apartado 4.1.f).
- Ya se ha creado y configurado el SDK de Firebase en el proyecto FrontEnd. La conexión entre el proyecto FrontEnd y Firebase se ha establecido correctamente, pudiendo haber logrado ya realizar los procesos de LogIn y LogOut, iniciado por el FrontEnd contra Firebase (apartado 4.3.e).
- Ya se configuró también el tipo de login de “usuario/contraseña” (apartado 4.3.c, sección de seguridad en Firebase).
- Ya se configuraron las plantillas de los correos electrónicos enviados por Firebase, para la verificación de usuario y cambio de contraseña (apartado 4.3.e).
- Se ha terminado de desarrollar completamente el código definitivo del desarrollo Full Stack: tanto en Angular (FrontEnd) como en Java Spring Boot (BackEnd).
- Ya se ha conseguido realizar un CRUD completo desde Angular al BackEnd exitosamente (apartado 4.3.f).

Se siguieron 7 pasos para terminar la codificación final del proyecto:

Paso 1: Se creó en Firebase las Collections “Contenidos” y “Usuarios”. Así mismo, se crea un primer “documento” de ejemplo en cada colección (con la estructura que tendrán los nuevos documentos futuros).

Véase las Fig. 4.3.36 y Fig. 4.3.37 para más información.

The screenshot shows the Firebase console interface. On the left, there's a sidebar with '+ Iniciar colección' and a list of collections: 'Contenidos' (highlighted with a green box) and 'Usuarios'. The main area shows a collection named 'Contenidos' with a single document highlighted by a green box. The document ID is '1681079630'. The document data is displayed in a tree view:

- anho:** 2022
- descripcion:** "Después de dos años acechando por las calle... (string)"
- generosPertenece:** "accion,tendencia"
- idContenido:** 1681079630
- linkVideo:** "https://www.youtube.com/watch?v=plF8hzSqwgo"
- nombre:** "The Batman"
- tipoContenido:** "pelicula"

Fig. 4.3.36: Nueva Collection Contenidos, en el proyecto Firebase. Documento ejemplo.

The screenshot shows the Firebase console interface. On the left, there's a sidebar with '+ Iniciar colección' and a list of collections: 'Contenidos' and 'Usuarios' (highlighted with a green box). The main area shows a collection named 'Usuarios' with a single document highlighted by a green box. The document ID is 'ejemplo@ejemplo.com'. The document data is displayed in a tree view:

- activo:** true
- apellidos:** "Fernández"
- edad:** 18
- genero:** 1
- id:** "ejemplo@ejemplo.com"
- nombre:** "Pedro"
- telefono:** 123456789

Fig. 4.3.37: Nueva Collection Usuarios, en el proyecto Firebase. Documento ejemplo.

Paso 2: Definición del prototipo de todas las nuevas llamadas REST en Insomnia Core, que se consumirán por nuestro cliente web (tanto las llamadas REST de Contenidos como las de Usuarios). Ver las figuras 4.3.38 – 4.3.52.

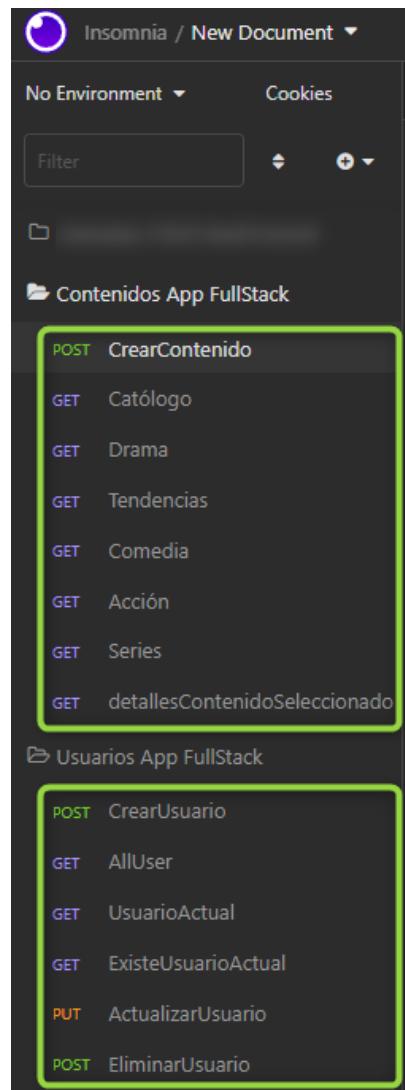


Fig. 4.3.38: Bloque general de llamadas REST, en Insomnia Core.

This screenshot shows a detailed view of a POST request. The URL is "localhost:8080/contenidos/nuevoContenido". The method is selected as "POST". The "JSON" tab is active, showing the following JSON payload:

```
1: {  
2:   "año": 1989,  
3:   "descripcion": "La trama de Dragon Ball Z se centra en la vida adulta de Son Goku, quien tendrá que defender la tierra de los numerosos villanos que amenazan con destruirla. Además, la serie trama de forma paralela la madurez de su hijo Gohan. La producción destaca por tener un tono más serio que sus predecesoras.",  
4:   "generosPertenece": "comedia,accion,drama",  
5:   "idContenido": 0,  
6:   "linkVideo": "https://www.youtube.com/watch?v=yir9PzzlCKs",  
7:   "nombre": "Dragon Ball Z",  
8:   "tipoContenido": "ficción"  
9: }
```

The "Send" button is highlighted in purple. To the right, the response is shown with a status of "200 OK", a duration of "7.89 s", and a size of "23 B". The "Preview" tab shows the JSON response: "1: { 2: \"result\": \"1684024766\" 3: }".

Fig. 4.3.39: Llamada 1 (CrearContenido), POST con datos I/O.

```

1 + [
2 +   {
3     "año": 2022,
4     "descripcion": "texto descripcion ejemplo",
5     "generosPertenece": "accion,tendencia",
6     "idContenido": 1681079630,
7     "linkVideo": "https://www.youtube.com/watch?v=p1F8hz5owgo",
8     "nombre": "The Batman",
9     "tipoContenido": "pelicula"
10    },
11   {
12     "año": 2006,
13     "descripcion": "texto descripcion ejemplo",
14     "generosPertenece": "drama",
15     "idContenido": 1683420707,
16     "linkVideo": "https://www.youtube.com/watch?v=BBmijmVg8tE",
17     "nombre": "Apocalypto",
18     "tipoContenido": "pelicula"
19   }
]

```

Fig. 4.3.40: Llamada 2 (Catálogo), GET con datos I/O.

```

1 + [
2 +   {
3     "año": 2006,
4     "descripcion": "texto descripcion ejemplo",
5     "generosPertenece": "drama",
6     "idContenido": 1683420707,
7     "linkVideo": "https://www.youtube.com/watch?v=BBmijmVg8tE",
8     "nombre": "Apocalypto",
9     "tipoContenido": "pelicula"
10    },
11   {
12     "año": 1972,
13     "descripcion": "texto descripcion ejemplo",
14     "generosPertenece": "drama",
15     "idContenido": 1683420864,
16     "linkVideo": "https://www.youtube.com/watch?v=vFkXN_KROMA",
17     "nombre": "El Padrino",
18     "tipoContenido": "pelicula"
19   }
]

```

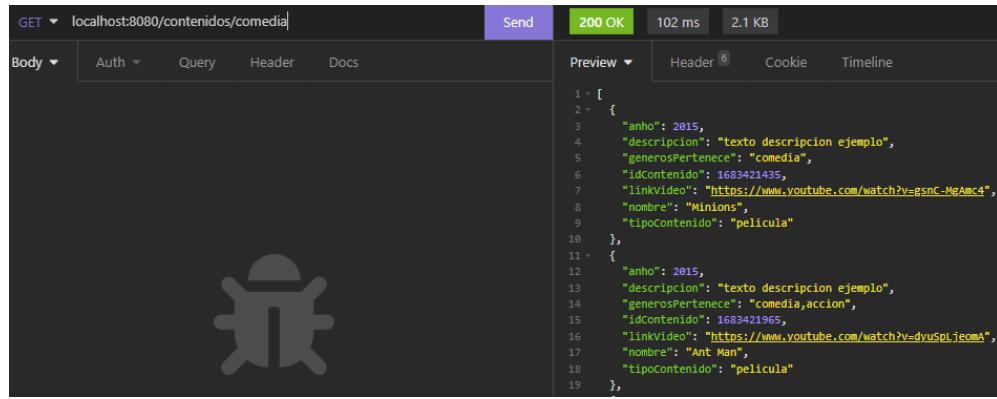
Fig. 4.3.41: Llamada 3 (Drama), GET con datos I/O.

```

1 + [
2 +   {
3     "año": 2022,
4     "descripcion": "texto descripcion ejemplo",
5     "generosPertenece": "accion,tendencia",
6     "idContenido": 1681079630,
7     "linkVideo": "https://www.youtube.com/watch?v=p1F8hz5owgo",
8     "nombre": "The Batman",
9     "tipoContenido": "pelicula"
10    },
11   {
12     "año": 2003,
13     "descripcion": "texto descripcion ejemplo",
14     "generosPertenece": "comedia,tendencia",
15     "idContenido": 1683422443,
16     "linkVideo": "https://www.youtube.com/watch?v=p2taikh4ey&list=PLC9Cy03L_13SwMxtAuyggulhhH9xuz8index=2",
17     "nombre": "Dos hombres y medio",
18     "tipoContenido": "serie"
19   }
]

```

Fig. 4.3.42: Llamada 4 (Tendencias), GET con datos I/O.

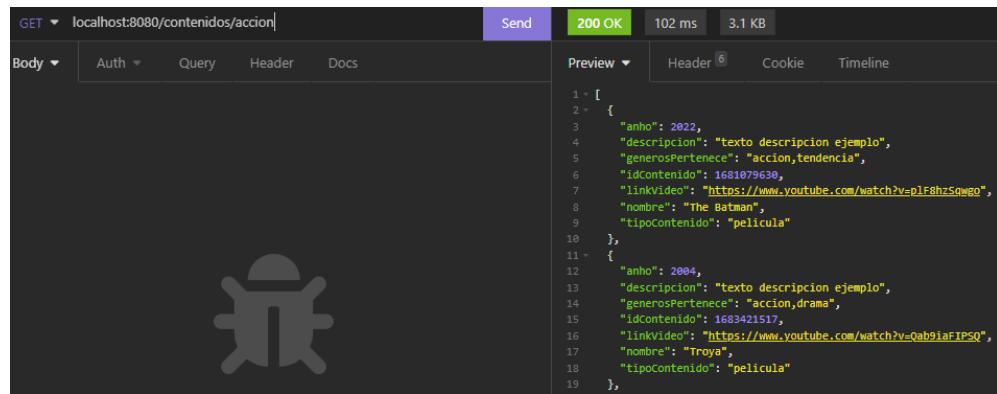


```

1 + [
2 +   {
3     "anho": 2015,
4     "descripcion": "texto descripcion ejemplo",
5     "generosPertenece": "comedia",
6     "idContenido": 1683421435,
7     "linkVideo": "https://www.youtube.com/watch?v=gsnC-MeAmc4",
8     "nombre": "Minions",
9     "tipoContenido": "pelicula"
10   },
11  {
12    "anho": 2015,
13    "descripcion": "texto descripcion ejemplo",
14    "generosPertenece": "comedia,accion",
15    "idContenido": 1683421965,
16    "linkVideo": "https://www.youtube.com/watch?v=dyuSpLjeomA",
17    "nombre": "Ant Man",
18    "tipoContenido": "pelicula"
19  },

```

Fig. 4.3.43: Llamada 5 (Comedia), GET con datos I/O.

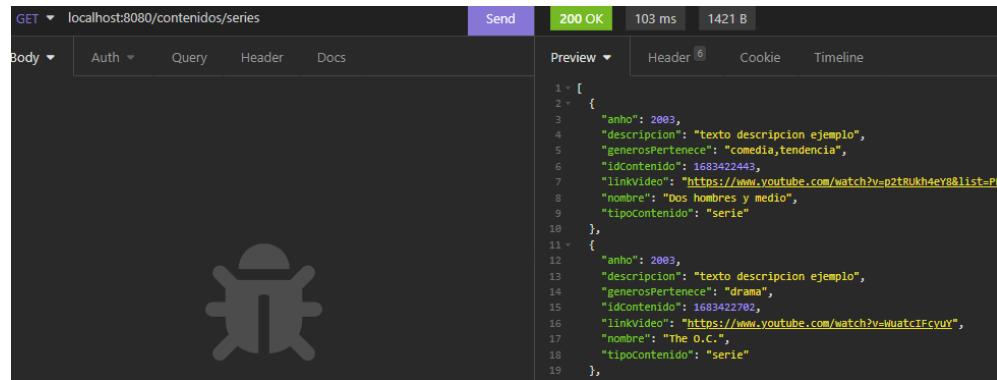


```

1 + [
2 +   {
3     "anho": 2022,
4     "descripcion": "texto descripcion ejemplo",
5     "generosPertenece": "accion,tendencia",
6     "idContenido": 1681079630,
7     "linkVideo": "https://www.youtube.com/watch?v=p1F8h-Sowgo",
8     "nombre": "The Batman",
9     "tipoContenido": "pelicula"
10   },
11  {
12    "anho": 2004,
13    "descripcion": "texto descripcion ejemplo",
14    "generosPertenece": "accion,drama",
15    "idContenido": 1683421517,
16    "linkVideo": "https://www.youtube.com/watch?v=Qab9iaFIPSO",
17    "nombre": "Troya",
18    "tipoContenido": "pelicula"
19  },

```

Fig. 4.3.44: Llamada 6 (Acción), GET con datos I/O.



```

1 + [
2 +   {
3     "anho": 2003,
4     "descripcion": "texto descripcion ejemplo",
5     "generosPertenece": "comedia,tendencia",
6     "idContenido": 1683422443,
7     "linkVideo": "https://www.youtube.com/watch?v=p2tRukh4ey8&list=PL",
8     "nombre": "Dos hombres y medio",
9     "tipoContenido": "serie"
10   },
11  {
12    "anho": 2003,
13    "descripcion": "texto descripcion ejemplo",
14    "generosPertenece": "drama",
15    "idContenido": 1683422702,
16    "linkVideo": "https://www.youtube.com/watch?v=wUatcIFcyuY",
17    "nombre": "The O.C.",
18    "tipoContenido": "serie"
19  },

```

Fig. 4.3.45: Llamada 7 (Series), GET con datos I/O.

```

1 + {
2   "anho": 2003,
3   "descripcion": "texto descripcion ejemplo",
4   "generosPertenece": "drama",
5   "idcontenido": 1683422702,
6   "linkvideo": "https://www.youtube.com/watch?v=MuatcIFcyuy",
7   "nombre": "The O.C.",
8   "tipocontenido": "serie"
9 }

```

Fig. 4.3.46: Llamada 8 (DetalleContenidoSeleccionado), GET con datos I/O.

```

1 + ejemplo@ejemplo.com

```

Fig. 4.3.47: Llamada 9 (CrearUsuario), POST con datos I/O.

```

1 + [
2   {
3     "id": "ejemplo@ejemplo.com",
4     "nombre": "Pedro",
5     "apellidos": "Fernández",
6     "edad": 18,
7     "activado": false,
8     "telefono": 123456789,
9     "genero": 1
10   },
11   {
12     "id": "██████████",
13     "nombre": null,
14     "apellidos": null,
15     "edad": null,
16     "activado": false,
17     "telefono": null,
18     "genero": 0
19   },

```

Fig. 4.3.48: Llamada 10 (AllUser), GET con datos I/O.

```

1 + {
2   "id": "ejemplo@ejemplo.com",
3   "nombre": null,
4   "apellidos": null,
5   "edad": null,
6   "activado": false,
7   "telefono": null,
8   "genero": 0
9 }

```

Fig. 4.3.49: Llamada 11 (UsuarioActual), GET con datos I/O.

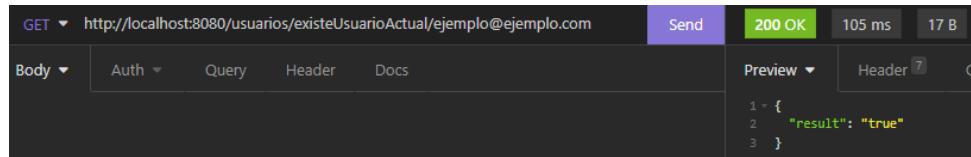


Fig. 4.3.50: Llamada 12 (ExisteUsuarioActual), GET con datos I/O.

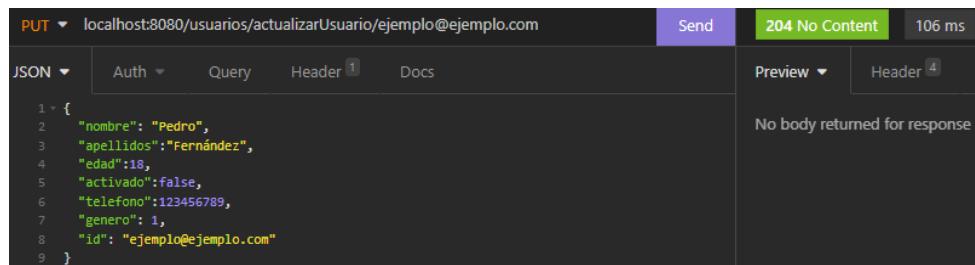


Fig. 4.3.51: Llamada 13 (ActualizarUsuario), PUT con datos I/O.

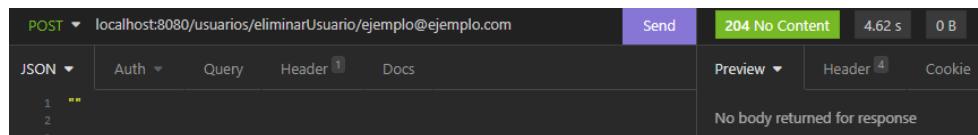


Fig. 4.3.52: Llamada 14 (EliminarUsuario), POST con datos I/O.

Paso 3: Comprobar que en base de datos desde Firebase aparecen los mismos resultados que devuelven las llamadas en InsomniaCore (durante creación, lectura, actualización y eliminación de documentos). Cada “Contenido” se llama por su el Timestamp asignado en el momento de creación, mientras que cada “Usuario” se llama por su nombre de usuario (será el propio correo electrónico).

Paso 4: Dar últimos retoques al código del proyecto BackEnd y finalizar su desarrollo. El código terminado del BackEnd será el siguiente (Fig. 4.3.53 – 4.3.65).



Fig. 4.3.53: Estructura de directorios, proyecto BackEnd.

```

1 package com.example.demo.controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.CrossOrigin;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import org.springframework.web.bind.annotation.PutMapping;
13 import org.springframework.web.bind.annotation.RequestBody;
14 import org.springframework.web.bind.annotation.RequestMapping;
15 import org.springframework.web.bind.annotation.RequestParam;
16 import org.springframework.web.bind.annotation.RestController;
17
18 import com.example.demo.dto.ContenidoDTO;
19 import com.example.demo.dto.UsuarioDTO;
20 import com.example.demo.mapper.ContenidoMapper;
21 import com.example.demo.mapper.UsuarioMapper;
22 import com.example.demo.model.Contenido;
23 import com.example.demo.model.ResponseString;
24 import com.example.demo.model.Usuario;
25 import com.example.demo.service.firebaseioContenidoService;
26 import com.example.demo.service.firebaseioUsuarioService;
27
28 @RestController
29 @RequestMapping("/contenidos")
30 @CrossOrigin(origins = "http://localhost:4200/")
31 public class ContenidosController {
32
33     @Autowired
34     private FirebaseContenidoService firebaseContenidoService;
35
36     @Autowired
37     private ContenidoMapper contenidoMapper;
38
39
40     // Llamada READ: lee todos los contenidos del catalogo, con llamada HTTP tipo GET.
41     @GetMapping(value = "/catalogo")
42     public ResponseEntity<List<Contenido>> mostrarTodosLosContenidosDelCatalogo () {
43
44         List<Contenido> contenidos = firebaseContenidoService.listarContenidosCatalogo();
45         contenidos = firebaseContenidoService.ordenarListaContenidos(contenidos);
46         ResponseEntity<Contenido> response = new ResponseEntity<Contenido>(contenidos, HttpStatus.OK);
47         return response;
48     }
49
50     @GetMapping(value = "/tendencias")
51     public ResponseEntity<List<Contenido>> mostrarTodosLosContenidosDeTendencias () {
52
53         List<Contenido> contenidos = firebaseContenidoService.listarContenidosTendencias();
54         contenidos = firebaseContenidoService.ordenarListaContenidos(contenidos);
55         ResponseEntity<Contenido> response = new ResponseEntity<Contenido>(contenidos, HttpStatus.OK);
56         return response;
57     }
58
59     @GetMapping(value = "/drama")
60     public ResponseEntity<List<Contenido>> mostrarTodosLosContenidosDeDrama () {
61
62         List<Contenido> contenidos = firebaseContenidoService.listarContenidosDrama();
63         contenidos = firebaseContenidoService.ordenarListaContenidos(contenidos);
64         ResponseEntity<Contenido> response = new ResponseEntity<Contenido>(contenidos, HttpStatus.OK);
65         return response;
66     }
67
68     @GetMapping(value = "/comedia")
69     public ResponseEntity<List<Contenido>> mostrarTodosLosContenidosDeComedia () {
70
71         List<Contenido> contenidos = firebaseContenidoService.listarContenidosComedia();
72         contenidos = firebaseContenidoService.ordenarListaContenidos(contenidos);
73         ResponseEntity<Contenido> response = new ResponseEntity<Contenido>(contenidos, HttpStatus.OK);
74         return response;
75     }
76
77     @GetMapping(value = "/accion")
78     public ResponseEntity<List<Contenido>> mostrarTodosLosContenidosDeAccion () {
79
80         List<Contenido> contenidos = firebaseContenidoService.listarContenidosAccion();
81         contenidos = firebaseContenidoService.ordenarListaContenidos(contenidos);
82         ResponseEntity<Contenido> response = new ResponseEntity<Contenido>(contenidos, HttpStatus.OK);
83         return response;
84     }
85
86     @GetMapping(value = "/series")
87     public ResponseEntity<List<Contenido>> mostrarTodosLosContenidosDeSeries () {
88
89         List<Contenido> contenidos = firebaseContenidoService.listarContenidosSeries();
90         contenidos = firebaseContenidoService.ordenarListaContenidos(contenidos);
91         ResponseEntity<Contenido> response = new ResponseEntity<Contenido>(contenidos, HttpStatus.OK);
92         return response;
93     }
94
95     @GetMapping(value = "/detalles/contenido/selected/{idContenidoSelected}")
96     public ResponseEntity<Contenido> mostrarDetallesContenidoSelected(@PathVariable long idContenidoSelected) {
97         Contenido contenido = firebaseContenidoService.conseguirDetallesContenidoSelected(idContenidoSelected);
98         return new ResponseEntity<Contenido>(contenido, HttpStatus.OK);
99     }
100
101     // Llamada CREATE: crea un contenido, con llamada HTTP tipo POST.
102     @PostMapping(value = "/crearcn")
103     public ResponseEntity<Contenido> crearNuevoContenido (@RequestBody ContenidoDTO contenidoDTO) {
104         Contenido contenido = this.contenidoMapper.comContenido(contenidoDTO);
105         ResponseString response = firebaseContenidoService.createContenido(contenido);
106         return new ResponseEntity<ResponseString>(response, HttpStatus.OK);
107     }
108
109 }

```

Fig. 4.3.54: Código fichero “ContenidosController.java”, proyecto BackEnd.

```

1 package com.example.demo.controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.CrossOrigin;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import org.springframework.web.bind.annotation.PutMapping;
13 import org.springframework.web.bind.annotation.RequestBody;
14 import org.springframework.web.bind.annotation.RequestMapping;
15 import org.springframework.web.bind.annotation.RequestParam;
16 import org.springframework.web.bind.annotation.RestController;
17
18 import com.example.demo.dto.UsuarioDTO;
19 import com.example.demo.mapper.UsuarioMapper;
20 import com.example.demo.model.ResponseString;
21 import com.example.demo.model.Usuario;
22 import com.example.demo.service.FirebaseUsuarioService;
23
24 @RestController
25 @RequestMapping("/usuarios")
26 @CrossOrigin(origins = "http://localhost:4200/")
27 public class UsuariosController {
28
29     @Autowired
30     private FirebaseUsuarioService firebaseUsuarioService;
31
32     @Autowired
33     private UsuarioMapper usuarioMapper; // Componente sin constructor
34
35     // Llamada CREATE: crea un usuario, con llamada HTTP tipo POST.
36     @PostMapping("creausuario/{idNuevoUsuario}")
37     public ResponseEntity<User> crearUsuario (@PathVariable String idNuevoUsuario, @RequestBody UsuarioDTO usuarioDTO){
38         Usuario usuario = this.usuarioMapper.comoUsuario(usuarioDTO);
39         String response = firebaseUsuarioService.crearUsuarioFirebase(usuario, idNuevoUsuario);
40         return new ResponseEntity(response, HttpStatus.OK);
41     }
42
43     // Llamada READ: lee todos los usuarios, con llamada HTTP tipo GET.
44     @GetMapping(value = "/allUsers")
45     public ResponseEntity<List<User>> mostrarTodosLosUsuarios (@RequestParam String ordenadosPor){
46         List<User> usuarios = firebaseUsuarioService.listarUsuariosFirebase();
47         usuarios = firebaseUsuarioService.ordenarListaUsuarios(usuarios, ordenadosPor);
48         ResponseEntity response = new ResponseEntity(usuarios, HttpStatus.OK);
49         return response;
50     }
51
52     // Llamada GET: devuelve obj ResponseString con "true"/"false" si existe o no el idUsuario
53     @GetMapping(value = "/existeUsuarioActual/{idUsuario}")
54     public ResponseEntity existeUsuarioActual (@PathVariable String idUsuario){
55         ResponseString response = firebaseUsuarioService.probarSiExisteUsuarioFirebase(idUsuario);
56         return new ResponseEntity(response, HttpStatus.OK);
57     }
58
59
60     @GetMapping(value = "/usuarioActual/{idUsuario} ")
61     public ResponseEntity<User> mostrarDetallesContenidoSeleccionado(@PathVariable String idUsuario){
62         User usuarioActual = firebaseUsuarioService.mostrarUsuarioActual(idUsuario);
63         return new ResponseEntity(usuarioActual, HttpStatus.OK);
64     }
65
66     // Llamada UPDATE: modifica un usuario con llamada HTTP tipo PUT.
67     @PutMapping("actualizarUsuario/{idUsuario}")
68     public ResponseEntity<void> modificarDetalleDeUsuario (@PathVariable String idUsuario,
69                 @RequestBody User usuarioDTO){
70         User usuario = this.usuarioMapper.comoUsuario(usuarioDTO);
71         Boolean b = firebaseUsuarioService.modificarUsuarioFirebase(idUsuario, usuario);
72         if (b != true){
73             return ResponseEntity.noContent().build();
74         } else {
75             ResponseEntity responseEntity = new ResponseEntity("Body: Imposible actualizar, el elemento no existe.", HttpStatus.BAD_REQUEST);
76             return responseEntity;
77         }
78     }
79
80     // Llamada DELETE: elimina un usuario con llamada HTTP tipo POST.
81     @PostMapping("eliminarUsuario/{id}")
82     public ResponseEntity<void> eliminarUsuario (@PathVariable String id){
83         Boolean b = firebaseUsuarioService.eliminarUsuarioPorIdFirebase(id);
84         if (b != null){
85             return ResponseEntity.noContent().build();
86         }
87     }
88     return ResponseEntity.notFound().build();
89 }

```

Fig. 4.3.55: Código fichero “UsuariosController.java”, proyecto BackEnd.

```

1 package com.example.demo.dto;
2 import java.io.Serializable;
3
4 public class ContenidoDTO implements Serializable{
5
6     private Long anho;
7     private String descripcion;
8     private String generosPertenece;
9     private Long idContenido;
10    private String linkVideo;
11    private String nombre;
12    private String tipoContenido;
13
14
15    public Long getAnho(){
16        return this.anho;
17    }
18
19    public void setAnho(Long anho){
20        this.anho= anho;
21    }
22
23    public String getDescripcion(){
24        return this.descripcion;
25    }
26
27    public void setDescripcion(String descripcion){
28        this.descripcion= descripcion;
29    }
30
31    public String getGenerosPertenece(){
32        return this.generosPertenece;
33    }
34
35    public void setGenerosPertenece(String generosPertenece){
36        this.generosPertenece= generosPertenece;
37    }
38
39    public Long getIdContenido(){
40        return this.idContenido;
41    }
42
43    public void setIdContenido(Long idContenido){
44        this.idContenido= idContenido;
45    }
46
47    public String getLinkVideo(){
48        return this.linkVideo;
49    }
50
51    public void setLinkVideo(String linkVideo){
52        this.linkVideo= linkVideo;
53    }
54
55    public String getNombre(){
56        return this.nombre;
57    }
58
59    public void setNombre(String nombre){
60        this.nombre= nombre;
61    }
62
63    public String getTipoContenido(){
64        return this.tipoContenido;
65    }
66
67    public void setTipoContenido(String tipoContenido){
68        this.tipoContenido= tipoContenido;
69    }
70}

```

Fig. 4.3.56: Código fichero “ContenidoDTO.java”, proyecto BackEnd.

```

1 package com.example.demo.dto;
2 import java.io.Serializable;
3
4 public class UsuarioDTO implements Serializable{
5     private String id;
6     private String nombre;
7     private String apellidos;
8     private Integer edad;
9     private boolean activado;
10    private long telefono;
11    private int genero;
12    private String aficion;
13
14    public String getId(){
15        return this.id;
16    }
17
18    public void setId(String id){
19        this.id= id;
20    }
21
22    public String getNombre(){
23        return this.nombre;
24    }
25
26    public void setNombre(String nombre){
27        this.nombre= nombre;
28    }
29
30    public String getApellidos(){
31        return this.apellidos;
32    }
33
34    public void setApellidos(String apellidos){
35        this.apellidos= apellidos;
36    }
37
38    public Integer getEdad(){
39        return this.edad;
40    }
41
42    public void setEdad(Integer edad){
43        this.edad=edad;
44    }
45
46    public boolean getActivado(){
47        return this.activado;
48    }
49
50    public void setActivado(boolean activado){
51        this.activado=activado;
52    }
53
54    public long getTelefono(){
55        return this.telefono;
56    }
57
58    public void setTelefono(Long telefono){
59        this.telefono=telefono;
60    }
61
62    public int getGenero(){
63        return this.genero;
64    }
65
66    public void setGenero(int genero){
67        this.genero=genero;
68    }
69
70    public String getAficion() {
71        return this.aficion;
72    }
73
74    public void setAficion(String aficion) {
75        this.aficion = aficion;
76    }
77 }

```

Fig. 4.3.57: Código fichero “UsuarioDTO.java”, proyecto BackEnd.

```

1 package com.example.demo.firebaseio;
2
3 import org.springframework.stereotype.Service;
4
5 import java.io.IOException;
6 import java.io.InputStream;
7
8 import java.annotation.PostConstruct;
9
10 import com.google.auth.oauth2.GoogleCredentials;
11 import com.google.cloud.firestore.Firestore;
12 import com.google.firebase.FirebaseApp;
13 import com.google.firebase.FirebaseOptions;
14 import com.google.firebase.cloud.FirestoreClient;
15
16
17 @Service
18 public class FirebaseInitializer {
19
20     @PostConstruct
21     private void initFirestore() throws IOException{
22
23         InputStream serviceAccount = getClass().getClassLoader().getResourceAsStream("private-key-firebase.json");
24
25         FirebaseOptions options = new FirebaseOptions.Builder()
26             .setCredentials(GoogleCredentials.fromStream(serviceAccount))
27             .setDatabaseUrl("https://fir-sbd01.firebaseio.com/")
28             .build();
29
30         if(FirebaseApp.getApps().isEmpty()){
31             FirebaseApp.initializeApp(options);
32         }
33
34     }
35
36     public Firestore getFirestore(){
37         return FirestoreClient.getFirestore();
38     }
39
40 }

```

Fig. 4.3.58: Código fichero “FirebaseInitializer.java”, proyecto BackEnd.

```

1 package com.example.demo.mapper;
2
3 import org.springframework.stereotype.Component;
4
5 import com.example.demo.dto.ContenidoDTO;
6 import com.example.demo.dto.UsuarioDTO;
7 import com.example.demo.model.Contenido;
8
9 @Component
10 public class ContenidoMapper{
11
12
13     public Contenido comoContenido(ContenidoDTO src) {
14
15         if(src == null){
16             return null;
17         }
18         Contenido contenido = new Contenido();
19         contenido.setAnho(src.getAnho());
20         contenido.setDescripcion(src.getDescripcion());
21         contenido.setGenerosPertenece(src.getGenerosPertenece());
22         contenido.setIdContenido(src.getIdContenido());
23         contenido.setLinkVideo(src.getLinkVideo());
24         contenido.setNombre(src.getNombre());
25         contenido.setTipoContenido(src.getTipoContenido());
26         return contenido;
27     }
28
29     public ContenidoDTO comoContenidoDTO(Contenido src) {
30
31         if(src == null){
32             return null;
33         }
34         ContenidoDTO contenidoDTO = new ContenidoDTO();
35         contenidoDTO.setAnho(src.getAnho());
36         contenidoDTO.setDescripcion(src.getDescripcion());
37         contenidoDTO.setGenerosPertenece(src.getGenerosPertenece());
38         contenidoDTO.setIdContenido(src.getIdContenido());
39         contenidoDTO.setLinkVideo(src.getLinkVideo());
40         contenidoDTO.setNombre(src.getNombre());
41         contenidoDTO.setTipoContenido(src.getTipoContenido());
42         return contenidoDTO;
43     }
}

```

Fig. 4.3.59: Código fichero “ContenidoMapper.java”, proyecto BackEnd.

```

1 package com.example.demo.mapper;
2
3 import org.springframework.stereotype.Component;
4
5 import com.example.demo.dto.UsuarioDTO;
6 import com.example.demo.model.Usuario;
7
8 @Component
9 public class UsuarioMapper{
10
11     public Usuario comoUsuario(UsuarioDTO src) {
12
13         if(src == null){
14             return null;
15         }
16         Usuario usuario = new Usuario();
17         usuario.setActivado(src.getActivado());
18         usuario.setApellidos(src.getApellidos());
19         usuario.setEdad(src.getEdad());
20         usuario.setGenero(src.getGenero());
21         usuario.setId(src.getId());
22         usuario.setNombre(src.getNombre());
23         usuario.setTelefono(src.getTelefono());
24         return usuario;
25     }
26
27     public UsuarioDTO comoUsuarioDTO(Usuario src) {
28
29         if(src == null){
30             return null;
31         }
32         UsuarioDTO usuarioDTO = new UsuarioDTO();
33         usuarioDTO.setActivado(src.getActivado());
34         usuarioDTO.setApellidos(src.getApellidos());
35         usuarioDTO.setEdad(src.getEdad());
36         usuarioDTO.setGenero(src.getGenero());
37         usuarioDTO.setId(src.getId());
38         usuarioDTO.setNombre(src.getNombre());
39         usuarioDTO.setTelefono(src.getTelefono());
40
41     }
}

```

Fig. 4.3.60: Código fichero “UsuarioMapper.java”, proyecto BackEnd.

```
1 package com.example.demo.model;
2
3 public class Contenido {
4
5     private Long anho;
6     private String descripcion;
7     private String generosPertenece;
8     private Long idContenido;
9     private String linkVideo;
10    private String nombre;
11    private String tipoContenido;
12
13    public Long getAnho(){
14        return this.anho;
15    }
16
17    public void setAnho(Long anho){
18        this.anho= anho;
19    }
20
21    public String getDescripcion(){
22        return this.descripcion;
23    }
24
25    public void setDescripcion(String descripcion){
26        this.descripcion= descripcion;
27    }
28
29    public String getGenerosPertenece(){
30        return this.generosPertenece;
31    }
32
33    public void setGenerosPertenece(String generosPertenece){
34        this.generosPertenece= generosPertenece;
35    }
36
37    public Long getIdContenido(){
38        return this.idContenido;
39    }
40
41    public void setIdContenido(Long idContenido){
42        this.idContenido= idContenido;
43    }
44
45    public String getLinkVideo(){
46        return this.linkVideo;
47    }
}
```

Fig. 4.3.61: Código parcial (inicial) del model “Contenido.java”, proyecto BackEnd.

```
1 package com.example.demo.model;
2
3 public class ResponseString {
4
5     private String result;
6
7     public String getResult(){
8         return this.result;
9     }
10
11    public void setResult(String result){
12        this.result= result;
13    }
14 }
```

Fig. 4.3.62: Código model “ResponseString.java”, proyecto BackEnd.

```

1 package com.example.demo.model;
2
3 public class Usuario {
4
5     private String id;
6     private String nombre;
7     private String apellidos;
8     private Integer edad;
9     private boolean activado;
10    private Long telefono;
11    private int genero;
12
13    public String getId(){
14        return this.id;
15    }
16
17    public void setId(String id){
18        this.id= id;
19    }
20
21    public String getNombre(){
22        return this.nombre;
23    }
24
25    public void setNombre(String nombre){
26        this.nombre= nombre;
27    }
28
29    public String getApellidos(){
30        return this.apellidos;
31    }
32
33    public void setApellidos(String apellidos){
34        this.apellidos= apellidos;
35    }
36
37    public Integer getEdad(){
38        return this.edad;
39    }
40
41    public void setEdad(Integer edad){
42        this.edad=edad;
43    }
44
45    public boolean getActivado(){
46        return this.activado;
47    }

```

Fig. 4.3.63: Código parcial (inicial) del model “Usuario.java”, proyecto BackEnd.

```

1 package com.example.demo.service;
2
3 import java.util.List;
4
5 import com.example.demo.model.Contenido;
6 import com.example.demo.model.ResponseString;
7 import com.example.demo.model.Usuario;
8
9 public interface FirebaseContenidoService {
10
11     List<Contenido> listarContenidosCatalogo();
12     List<Contenido> listarContenidosTendencias();
13     List<Contenido> listarContenidosDrama();
14     List<Contenido> listarContenidosComedia();
15     List<Contenido> listarContenidosAcción();
16     List<Contenido> listarContenidosSeries();
17     Contenido conseguirDetallesContenidoSeleccionado(Long idContenidoSeleccionado);
18     ResponseString crearContenido(Contenido contenido);
19     List<Contenido> ordenarListaContenidos (List<Contenido> contenidos);
20 }

```

Fig. 4.3.64: Código fichero “FirebaseContenidoService.java”, proyecto BackEnd.

```
1 package com.example.demo.service;
2
3 import java.util.List;
4
5 import com.example.demo.model.ResponseString;
6 import com.example.demo.model.Usuario;
7
8 public interface FirebaseUsuarioService {
9
10     List<Usuario> listarUsuariosFirebase();
11     ResponseString probarSiExisteUsuarioFirebase(String idUsuarioActual);
12     Usuario mostrarUsuarioActual(String idUsuario);
13     String crearUsuarioFirebase(Usuario usuario, String idUsuario);
14     Boolean modificarUsuarioFirebase(String id, Usuario usuario);
15     Boolean eliminarUsuarioPorIdFirebase(String id);
16     List<Usuario> ordenarListaUsuarios (List<Usuario> usuarios, String ordenadosPor);
17 }
```

Fig. 4.3.65: Código fichero “FirebaseUsuarioService.java”, proyecto BackEnd.

```

1 package com.example.demo.service;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.Comparator;
6 import java.util.HashMap;
7 import java.util.List;
8 import java.util.Map;
9
10 import com.example.demo.firebaseio.firebaseio_initializer;
11 import com.example.demo.modelo.Contenido;
12 import com.google.api.core.ApiFuture;
13 import com.google.cloud.firestore.CollectionReference;
14 import com.google.cloud.firestore.DocumentSnapshot;
15 import com.google.cloud.firestore.QuerySnapshot;
16 import com.google.cloud.firestore.WriteResult;
17 import org.springframework.beans.factory.annotation.Autowired;
18 import org.springframework.stereotype.Service;
19
20 @Service
21 public class FirebaseContenidoServiceImpl implements FirebaseContenidoService {
22     @Autowired
23     private FirebaseInitializer firebase;
24
25     @Override
26     public List<Contenido> listarContenidosCatalogo(){
27         List<Contenido> response = new ArrayList<>();
28         Contenido contenido;
29         ApiFuture<QuerySnapshot> querySnapshotApIfeature = obtenerColeccion().get();
30         try {
31             for(DocumentSnapshot doc : querySnapshotApIfeature.get().getDocuments()){
32                 contenido = doc.toObject(Contenido.class);
33                 try{
34                     contenido.setIdContenido( Long.valueOf(doc.getId()) );
35                 } catch (Exception e) {
36                     e.getMessage();
37                 }
38                 response.add(contenido);
39             }
40         } catch (Exception e) {
41             e.getMessage();
42         }
43         return response;
44     }
45     catch (Exception e) {
46         return null;
47     }
48 }
49
50     @Override
51     public List<Contenido> listarContenidosTendencias(){
52         List<Contenido> response = new ArrayList<>();
53         Contenido contenido;
54         ApiFuture<QuerySnapshot> querySnapshotApIfeature = obtenerColeccion().get();
55         try {
56             for(DocumentSnapshot doc : querySnapshotApIfeature.get().getDocuments()){
57                 contenido = doc.toObject(Contenido.class);
58                 String[] generosPertenece = Arrays.asList(contenido.getGenerosPertenece().split(regex));
59                 if(generosPertenece.contains("Tendencia")){
60                     try{
61                         contenido.setIdContenido( Long.valueOf(doc.getId()) );
62                     } catch (Exception e) {
63                         e.getMessage();
64                     }
65                     response.add(contenido);
66                 }
67             }
68         } catch (Exception e) {
69             e.getMessage();
70         }
71         return response;
72     }
73     catch (Exception e) {
74         return null;
75     }
76 }
77
78     @Override
79     public Contenido conseguirDetallesContenidoSeleccionado(Long idContenidoSeleccionado){
80         Contenido response = new Contenido();
81         Contenido contenido;
82         ApiFuture<QuerySnapshot> querySnapshotApIfeature = obtenerColeccion().get();
83         try {
84             for(DocumentSnapshot doc : querySnapshotApIfeature.get().getDocuments()){
85                 contenido = doc.toObject(Contenido.class);
86                 Long p= contenido.getIdContenido();
87                 if(p.longValue() == idContenidoSeleccionado.longValue()){
88                     return contenido;
89                 }
90             }
91         } catch (Exception e) {
92             e.getMessage();
93         }
94         return response;
95     }
96     catch (Exception e) {
97         return null;
98     }
99
100 // servicio Create
101 @Override
102 public void crearContenido(Contenido contenido) {
103     ApiFuture<WriteResult> documentosUsuarios = null;
104     long datetime = System.currentTimeMillis() / 1000;
105     documentosUsuarios = obtenerContenidosFirebase(contenido, datetime);
106     ApiFuture<String> resultado = feature = obtenerColeccion().document(String.valueOf(datetime)).create(documentosUsuarios);
107     Response<String> response = new Response<>();
108     try {
109         if(feature.getResult() != null){
110             response.setResult(datetime.toString());
111             return response;
112         }
113         response.setResult("resultado");
114         return response;
115     } catch (Exception e) {
116         response.setResult("resultado");
117         return response;
118     }
119 }
120
121 // metodo que ordena los Usuarios mostrados en operaciones GET HTTP, segun id, nombre o edad
122 @Override
123 public List<Contenido> ordenarListContenidos (List<Contenido> contenidos){
124     contenidos.sort(Comparator.comparing(Contenido::getIdContenido));
125     return contenidos;
126 }
127
128 private CollectionReference obtenerColección () {
129     return firebase.getFirestore().collection("Contenidos");
130 }
131
132 private Map<String, Object> obtenerContenidosFirebase(Contenido contenido, long idContenido){
133     Map<String, Object> documentosContenidosMap = new HashMap<>();
134     documentosContenidosMap.put("id", idContenido);
135     documentosContenidosMap.put("nombre", contenido.getNombre());
136     documentosContenidosMap.put("descripcion", contenido.getDescripcion());
137     documentosContenidosMap.put("generosPertenece", contenido.getGenerosPertenece());
138     documentosContenidosMap.put("linkVideo", contenido.getLinkVideo());
139     documentosContenidosMap.put("tipoContenido", contenido.getTipoContenido());
140     documentosContenidosMap.put("ultimoModificado", contenido.getUltimoModificado());
141     return documentosContenidosMap;
142 }
143
144 }
145
146 }
147

```

Fig. 4.3.66: Código parcial, fichero “FirebaseContenidoServiceImpl.java”, proyecto BackEnd.

```

1 package com.example.demo.service;
2
3 import java.util.ArrayList;
4 import java.util.Comparator;
5 import java.util.HashMap;
6 import java.util.List;
7 import java.util.Map;
8 import com.example.demo.firebaseio.FirebaseInitializer;
9 import com.example.demo.model.ResponseString;
10 import com.example.demo.model.Usuario;
11 import com.google.api.core.ApiFuture;
12 import com.google.cloud.firestore.CollectionReference;
13 import com.google.cloud.firestore.DocumentSnapshot;
14 import com.google.cloud.firestore.QuerySnapshot;
15 import com.google.cloud.firestore.WriteResult;
16 import com.google.rpc.context.AttributeContext.Response;
17 import org.apache.http.protocol.ResponseServer;
18 import org.springframework.beans.factory.annotation.Autowired;
19 import org.springframework.stereotype.Service;
20
21 @Service
22 public class FirebaseUsuarioServiceImpl implements FirebaseUsuarioService {
23     @Autowired
24     private FirebaseInitializer firebase;
25
26     // servicio READ
27     @Override
28     public List<Usuario> listarUsuariosFirebase() {
29         List<Usuario> response = new ArrayList<>();
30         Usuario usuario;
31         ApiFuture<QuerySnapshot> querySnapshotApiFeature = obtenerColeccion().get();
32         try {
33             for(DocumentSnapshot doc : querySnapshotApiFeature.get().getDocuments()){
34
35                 usuario = doc.toObject(Usuario.class);
36                 try{
37                     usuario.setId(doc.getId().toString());
38                     response.add(usuario);
39                 }
40                 catch(Exception e){
41                     e.getMessage();
42                 }
43             }
44             return response;
45         } catch (Exception e) {
46             return null;
47         }
48     }
49
50     @Override
51     public Usuario mostrarUsuarioActual(String idUsuario){
52         Usuario response = new Usuario();
53         Usuario usuario;
54         ApiFuture<QuerySnapshot> querySnapshotApiFeature = obtenerColeccion().get();
55         try {
56             for(DocumentSnapshot doc : querySnapshotApiFeature.get().getDocuments()){
57                 usuario = doc.toObject(Usuario.class);
58                 if(usuario.getId().equals(idUsuario)){
59                     response = usuario;
60                 }
61             }
62             return response;
63         } catch (Exception e) {
64             return null;
65         }
66     }
67
68     // servicio CREATE
69     @Override
70     public String crearUsuarioFirebase(Usuario usuario, String idUsuario) {
71         Map<String, Object> documentoUsuarios = new HashMap<>();
72         documentoUsuarios = obtenerUsuariosFirebase(usuario);
73         Long datetime = System.currentTimeMillis();
74         ApiFuture<WriteResult> feature = obtenerColeccion().document(String.valueOf(idUsuario)).create(documentoUsuarios);
75         try{
76             if(feature.get() !=null){
77                 return idUsuario;
78             }
79             return "-1";
80         } catch(Exception e){
81             return "-1";
82         }
83     }
84
85     // servicio UPDATE
86     @Override
87     public Boolean modificarUsuarioFirebase(String id, Usuario usuario) {
88         Map<String, Object> documentoUsuarios = new HashMap<>();
89         documentoUsuarios = obtenerUsuariosFirebase(usuario);
90         ApiFuture<WriteResult> feature = obtenerColeccion().document(id).set(documentoUsuarios);
91         try{
92             if(feature.get() !=null){
93                 return Boolean.TRUE;
94             }
95             return Boolean.FALSE;
96         } catch(Exception e){
97             return Boolean.FALSE;
98         }
99     }
100
101     // servicio DELETE
102     @Override
103     public Boolean eliminarUsuarioPorIdFirebase(String id) {
104         ApiFuture<WriteResult> feature = obtenerColeccion().document(id).delete();
105         try{
106             if(feature.get() !=null){
107                 return Boolean.TRUE;
108             }
109             return null;
110         } catch(Exception e){
111             return null;
112         }
113     }
114
115     // metodo interno 1: establece referencia al Collection Firestore
116     private CollectionReference obtenerColeccion (){
117         return firebase.getFirestore().collection("Usuarios");
118     }
119
120     // metodo interno 2: coloca el hashMap con el usuario entrante y devuelve el hashMap completado
121     private Map<String, Object> obtenerUsuariosFirebase(Usuario usuario){
122         Map<String, Object> documentosUsuarioMap = new HashMap<>();
123         documentosUsuarioMap.put(key:"nombre", usuario.getNombre());
124         documentosUsuarioMap.put(key:"apellidos", usuario.getApellidos());
125         documentosUsuarioMap.put(key:"edad", usuario.getEdad());
126         documentosUsuarioMap.put(key:"activada", usuario.getActivada());
127         documentosUsuarioMap.put(key:"telefono", usuario.getTelefono());
128         documentosUsuarioMap.put(key:"genero", usuario.getGenero());
129         documentosUsuarioMap.put(key:"id", usuario.getId());
130         return documentosUsuarioMap;
131     }
132
133 }

```

Fig. 4.3.67: Código parcial, fichero “FirebaseUsuarioServiceImpl.java”, proyecto BackEnd.

Paso 5: Terminar el desarrollo completo del proyecto FrontEnd (Angular). Se muestra a continuación la estructura general del proyecto FrontEnd definitivo (Fig. 4.3.68).

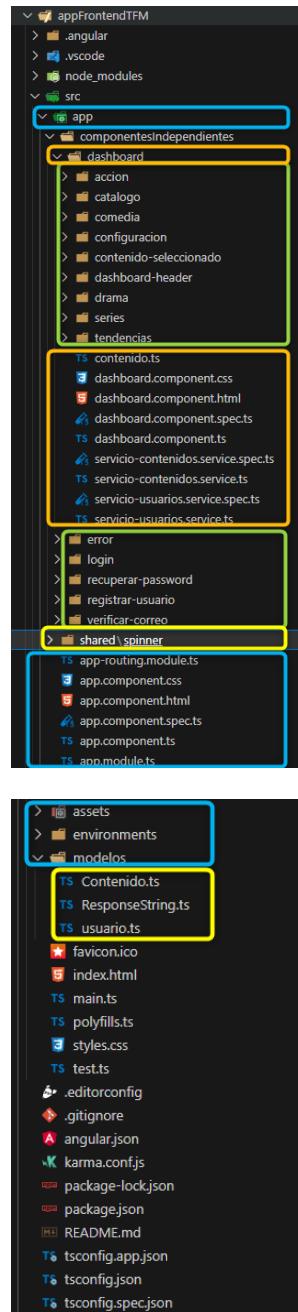


Fig. 4.3.68: Estructura de archivos global, proyecto Angular (FrontEnd).

Paso 6: Guardar, en el directorio ASSETS, la foto asociada a cada película o serie (la película que acabamos de crear en Firebase, dentro de la colección Contenido).

Importante: cada imagen de cartelera se llamará igual que el Timestamp con el que llamamos al contenido creado (id del contenido asociado a la imagen, igual al nombre de esa imagen).

Para conocer todas las imágenes del proyecto, véase Fig. 4.3.69.

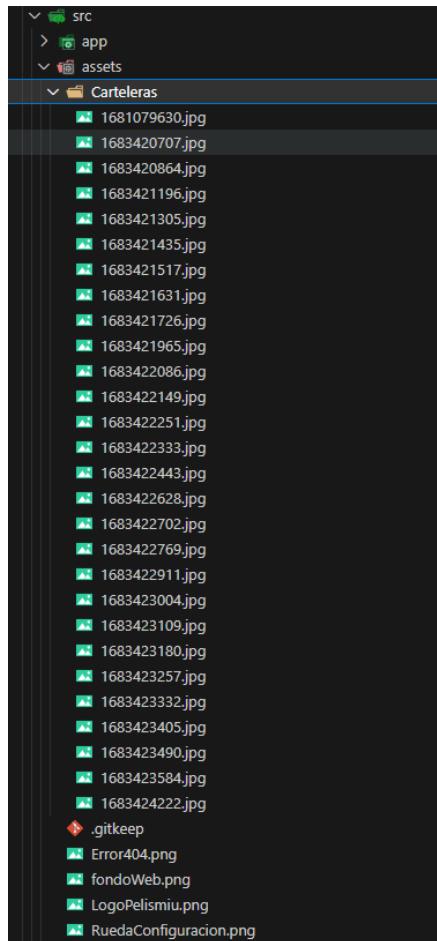


Fig. 4.3.69: imágenes de todo el proyecto, directorio ASSETS.

Paso 7: Probamos que funcionen correctamente las llamadas REST de tipo GET, correspondientes a la consulta de: “Catálogo”, “Tendencias”, “Drama”, “Comedia”, “Acción” y “Series”.

Como en el apartado 4.3.f se creó un CRUD completo de llamadas REST desde el cliente web, ahora replicar las llamadas adicionales que necesitemos para toda la web y según su tipo (GET, PUT o POST), probando igualmente que el resto de llamadas funcionan desde el cliente web.

h) Web final terminada: resultados funcionales.

Partimos de las siguientes premisas respecto a la funcionalidad previa:

- Ya se tiene funcionando correctamente el proyecto BackEnd (API REST).
- Ya se tendría arrancado el proyecto BackEnd.
- Ya se tendría arrancado el proyecto FrontEnd.
- Ya se tiene instalado un navegador de uso frecuente, a preferir “Fire Fox” o “Google Chrome”.

La exposición de la funcionalidad completa se dividirá en los siguientes pasos:

Paso 1: Para evitar que el BackEnd rechace las llamadas REST procedentes del BackEnd, siempre asegurarnos de tener configurada la siguiente etiqueta (en Sprint Boot), justamente sobre la interfaz de clase de todos los controllers en el proyecto BackEnd (fig. 4.3.48):

```
@CrossOrigin(origins = "http://localhost:4200/")
```

Paso 2: Poner la dirección base de localhost que llamará al cliente web:
<http://localhost:4200/>.

Nótese que se carga directamente la página de LogIn como primera página del sitio (<http://localhost:4200/login>), dando solo opción a 3 movimientos desde el exterior de la web:

- 1- Logearse par entrar al interior de la web (Fig. 4.3.70).
- 2- Registrar un nuevo usuario con el que logearse más adelante.
- 3- Recuperar la contraseña si la hemos olvidado.

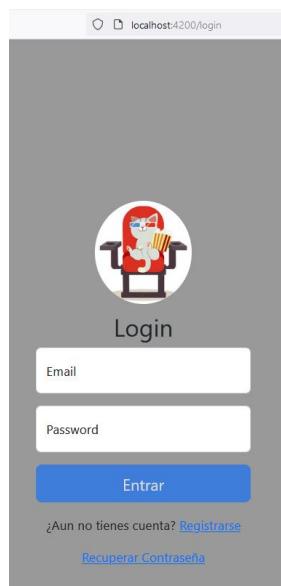


Fig. 4.3.70: Página de LogIn.

Paso 3: Registrar un nuevo usuario, que usaremos para acceder a la web PelisMiu. Para ello debemos clicar en el link “Registrarse” (de la imagen previa) y llenar todos los campos del formulario correctamente. Para ver la página de Registro, ver Fig. 4.3.71.



Fig. 4.3.71: Página de Registro.

Paso 4: Al clicar en “Registrarme” nos mandan un correo electrónico para verificar la cuenta, mientras se no redirige al LogIn de nuevo. El correo de verificación sería algo como el de la Fig. 4.3.72.

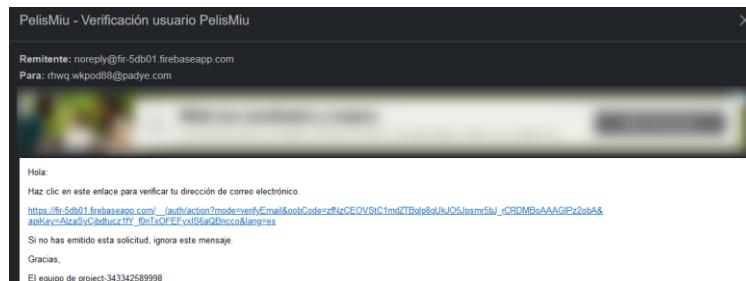


Fig. 4.3.72: Correo de verificación de la cuenta.

Paso 5: Intentando acceder al interior de la web sin verificar el correo facilitado accederemos a una web donde nos obligará a verificar el correo recibido antes de logearnos. Véase Fig. 4.2.73.

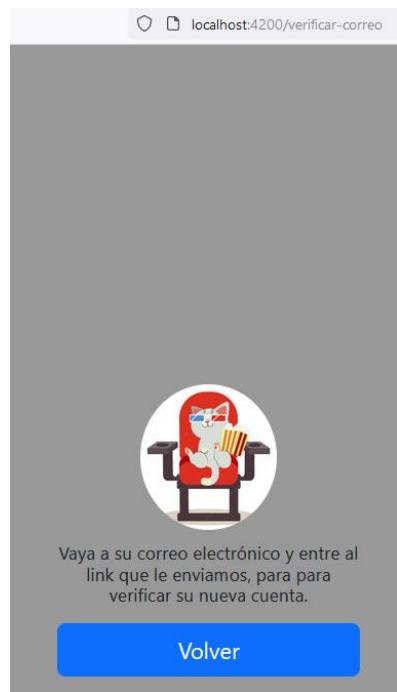


Fig. 4.3.73: página de Verificación de Correo.

Paso 6: Una vez volvamos a la página de LogIn y verificamos el correo recibido al crear la cuenta, intentar logearse de nuevo en la web. Esta vez podremos acceder a todas las secciones de contenidos, la configuración y el detalle del contenido seleccionado.

La primera página visualizada tras registrarnos siempre será la de “Catálogo”, donde veremos todos los contenidos de la web ordenados por orden inverso de llegada a la web (véase Fig. 4.3.74 para ver parte de los contenidos de la web).

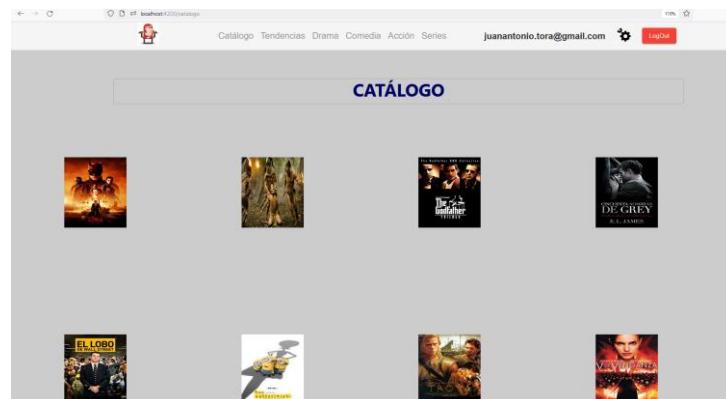


Fig. 4.3.74: página “Catálogo”.

Naveguemos por las 5 secciones de contenidos restantes, disponibles en la cabecera superior de la web: Tendencias, Drama, Comedia, Acción y Series.

Véase las imágenes comprendidas entre Fig. 4.3.75 – Fig. 4.3.79.



Fig. 4.3.75: página de “Tendencias”.

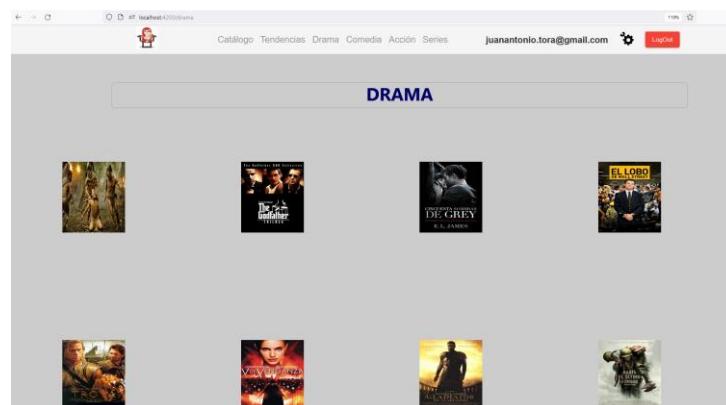


Fig. 4.3.76: página parcial de “Drama”.

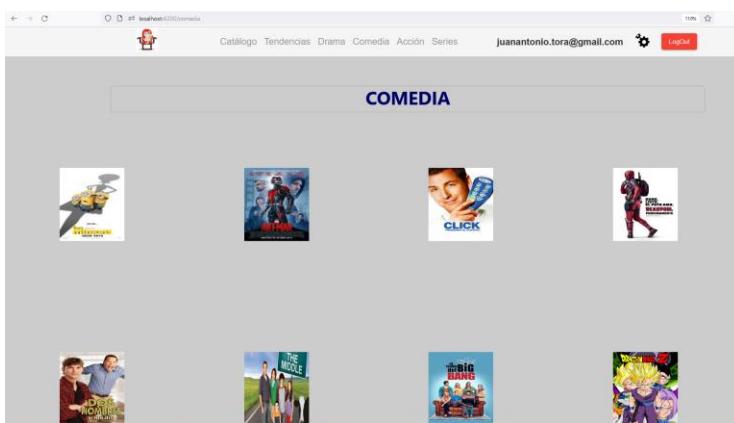


Fig. 4.3.77: página de “Comedia”.

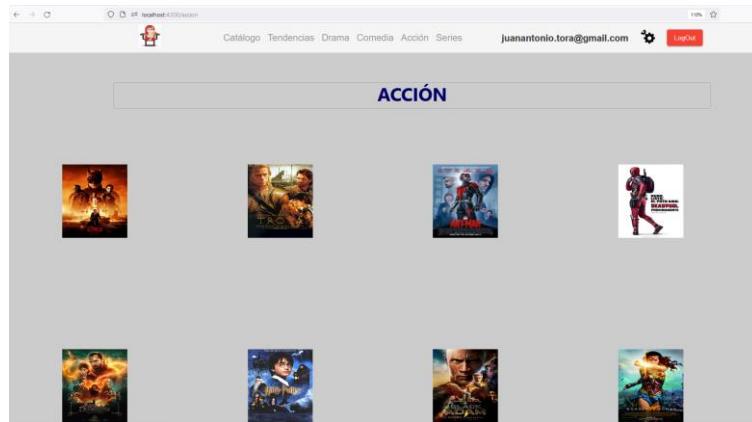


Fig. 4.3.78: página parcial de “Acción”.

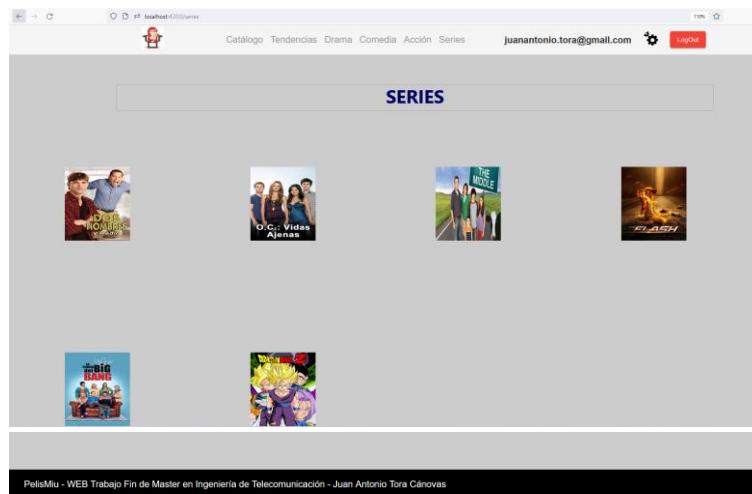


Fig. 4.3.79: página de “Series”.

Paso 7: Ir a la página de ContenidoSeleccionado, donde podremos visualizar los atributos de un contenido seleccionado (ver Fig. 4.3.80). Entre esos atributos de contenidos podemos mencionar los siguientes:

- ❖ Año de emisión.
- ❖ Nombre del contenido.
- ❖ Contenido (vídeo Youtube).

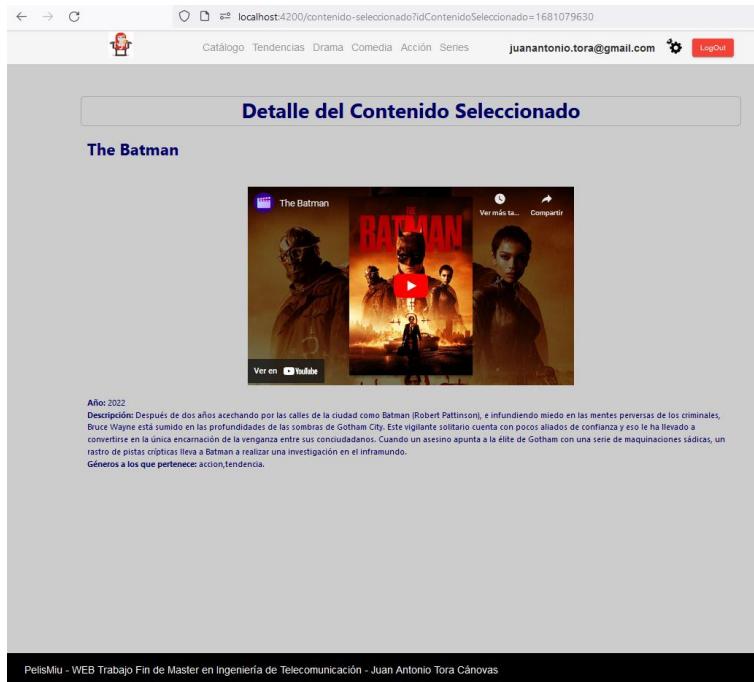


Fig. 4.3.80: página de “Contenido Seleccionado”.

Paso 8: Ir a la página de Configuración (ícono de rueda que encontrará arriba a la derecha, al lado de LogOut), donde podremos ver un formulario con las propiedades de usuario.

Entre todas las propiedades visualizadas aquí, no se podrán modificar ni el “usuario” ni el valor de “activo” (solo se podrán modificar por los propietarios de la web).

Si es la primera vez que cargas la página de Configuración, aparecerán todos los campos vacíos (véase la Fig. 4.3.81).



Fig. 4.3.81: página Configuración la primera vez que es cargada.

Paso 9: Desde la página de Configuración, probaremos el botón de “Actualizar Usuario”:

- Dar valor a los campos del formulario (son string, excepto tlfon y edad que serán numéricos). Sin clicar el botón de Actualizar Usuario ir a página Catálogo y luego volver a la Configuración. Apreciará que tenemos la misma página que vemos en la anterior Fig. 4.3.81 (no se han guardado los campos modificados).
- Sin embargo, si modificamos los campos del formulario como intentamos antes, pulsamos el botón Actualizar Usuario, nos vamos a página Catálogo y volvemos a Configuración, ahora los valores sí aparecerán modificados (se han guardado al clicar Actualizar Usuario). Ver resultados actualizados, en Fig. 4.3.82.

The screenshot shows a web application window titled 'CONFIGURACIÓN'. At the top, there is a navigation bar with links to 'Catálogo', 'Tendencias', 'Drama', 'Comedia', 'Acción', 'Series', and a user email 'juanantonio.tora@gmail.com'. On the right side of the header is a 'Logout' button. Below the header, the main content area has a title 'CONFIGURACIÓN' in bold. It contains a form with the following fields and values:

- Usuario: juanantonio.tora@gmail.com
- Nombre: Gerar
- Apellidos: Gómez
- Edad: 18
- Teléfono: 123456789
- Género:
 - Masculino
 - Femenino
 - Otro
- Activo: false

At the bottom of the form are two buttons: 'Actualizar usuario' (highlighted in blue) and 'Eliminar cuenta'.

Fig. 4.3.82: Propiedades de usuario actualizadas, página Configuración.

Paso 10: Probaremos el botón de LogOut, desde cualquier página, que estará siempre colocado en la parte superior derecha. Nos sacará de la web interna al cabo de 1 segundo, y nos quitará el acceso a los contenidos de la zona interna de la web. En caso de intentar acceder a una URL interna siempre redirige al LogIn.

Paso 11: Desde el exterior de la web y en la pantalla de Login, clicar el enlace que tenemos ahí de “Recuperar Contraseña”. En la página “Recuperar Contraseña”, introducir nuestro correo electrónico y enviar una solicitud de recuperación de contraseña.

Al enviar la solicitud de recuperación de contraseña, la aplicación enviará un correo de electrónico que llevará un link de cambio de contraseña. Entra ahora a ese correo electrónico que has recibido y cambia la contraseña en la dirección indicada dentro del correo.

Véase para más información la Fig. 4.3.83.



Fig. 4.3.83: Pantalla de “Recuperar Contraseña” y Correo recibido para cambiar password.

Paso 12: Desde la pantalla de login, entrar en la web nuevamente. Ahora probaremos qué ocurre cuando ponemos una URL incorrecta en el navegador.

En caso de URL incorrecta nos redirige a una página de Error, mostrando una visualización agradable al usuario. Véase un ejemplo en la Fig. 4.3.84.

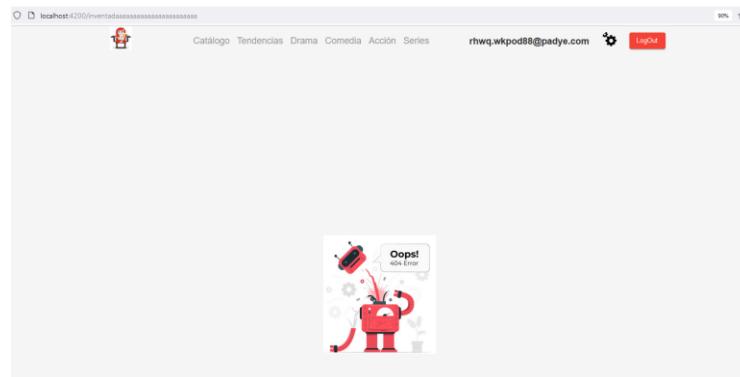


Fig. 4.3.84: página de error, cuando una URL no es válida.

Paso 13: por último, si queremos eliminar la cuenta de usuario de la plataforma, ir a la página de Configuración y clicar el botón “Eliminar Cuenta”.

Esta acción eliminará el acceso a la plataforma, pero no el registro: en el futuro no podrá accederse a la web con el mismo usuario (desde el logIn), ni volver a registrarse como nuevo usuario, salvo que se solicite reactivar el usuario a los administradores de la web.

5. Conclusiones finales.

Durante el tiempo que ha llevado la redacción e investigación de este trabajo, la realización de este proyecto ha contribuido a mejorar profesionalmente en competencias transversales, además de formarme en tecnologías nuevas con Angular.

Alguna de las competencias transversales adquiridas son los conocimientos en DevOp, metodologías Ágiles, SCRUM, etc (ver ANEXO VI).

Los apartados 2 y 3, aunque pueden resultar tediosos, plasman el hecho de que cualquier programador, con un mínimo de experiencia, podrá crear una aplicación casi gratuita profesionalmente, y desde cero.

El apartado 4 fue el más costoso de realizar, pero tengo la esperanza de que la distribución en 2 tutoriales back/front y culminando en otro FullStack, puedan dar al lector de este trabajo una mayor comprensión global de todo el proceso. De hecho, se colocó primeramente el tutorial backend porque el frontend no se suele desarrollar hasta poder consumir las llamadas del back.

No se pudo cubrir la parte de API REST con JWT o BearerToken, porque se quedaba mucho más extenso este trabajo, pero quedo contento con el resultado global.

En los 2 siguientes apartados (y últimos) podrá obtener más información de estudio y documentación sobre algunos temas tratados en el trabajo.

Si se desea descargar el código terminado de los apartados 4.1, 4.2 y 4.3, puede hacerlo en el siguiente repositorio público de GitHub:

<https://github.com/juanantoniotora/codigoTrabajoFinDeMasterEnIngTelecomunicaciones>

Por último, se dejan 4 líneas de continuación a partir de este trabajo:

- 1- Asegurar canales públicos con: SSH, 2FA, cifrado de archivos y seguridad en BBDD Firebase.
- 2- Despliegues de la aplicación web a entornos productivos: ver compatibilidad con Java 11, Maven 2.7 y Tomcat v10.1.X. Más info. en ANEXO VII.
- 3- Uso de OpenShift, GitHubActions, Docker y Kubernetes: ANEXO V.
- 4- Gestión de versiones y de código fuente, con buenas prácticas: ANEXO III y ANEXO IV.

6. Anexos

·Anexo I: metodología CI/CD.

En ingeniería del software, CD/CI se definen como:

“[...] refiere a las prácticas combinadas de integración continua y entrega continua. En el contexto de comunicación corporativa, CI/CD también puede referirse al proceso global de diseño corporativo e identidad corporativa.” (Fuente: Wikipedia).

Diríjase a la Fig. 6.1 para ver un ejemplo de la arquitectura CI/CD.



Fig. 6.1: Imagen de web oficial RedHat sobre arquitectura CI/CD.

·Anexo II: seguridad.

Criterios mínimos de seguridad en desarrollo web:

- 1) Control de acceso a los datos, lo más estricto posible.
- 2) Realizar copias de seguridad (de respaldo).
- 3) Utilizar contraseñas seguras (entre servicios y en dispositivos).
- 4) Trabajar en la nube, siempre que nos sea posible.
- 5) Proteger correo electrónico corporativo y personal eficazmente.
- 6) Opcionalmente, usar protectores de aplicaciones web.

Fundamentos de la tecnología JWT:

Video técnico informativo sobre JWT del canal Youtube “CodelyTV”:

<https://www.youtube.com/watch?v=3o4vElkiRgE&t=619s>

Estructura visual de un JWT decodificado en texto claro (Fig. AII.1):

The screenshot shows the jwt.is website interface. At the top, there's a navigation bar with links for Debugger, Libraries, Introduction, Ask, and a Crafted by auth0 logo. Below the navigation, a yellow warning bar states: "Warning: JWTs are credentials, which can grant access to resources. Be careful where you paste them! We do not record tokens, all validation and debugging is done on the client side." The main area is divided into two sections: "Encoded" on the left and "Decoded" on the right.

Encoded: This section contains the raw JWT string:

```
eyJhbGciOiJSUzIiNiIsImtpZCI6ImM4MjNkMWEVlMzk1LCJ9eXAiOiJKV1QiLCJpY3M10iJodHbWczovL3Ny3VvYzXRVa2VuLmdvb2dsZS5jb2BvZmlyLTvKjAxIiwiYXVkJoiZmlyLTvKjAxIiwiuX2luX3Byb3ZpZGVyIjoiG0Fzc3dvcnQifx0.r12aVs2yh5FicquvlBiuPk1HqjXh2fdgcPghdmBjeIfAEhx0N0duUTk56wEuvDytX3iXY_kFpQEwz2Q0._KE-
```

Decoded: This section shows the decoded token structure:

```
HEADER: ALGORITHM & TOKEN TYPE
{
  "alg": "RS256",
  "kid": "█████████████████████████████████████",
  "typ": "JWT"
}

PAYLOAD: DATA
{
  "iat": "https://securetoken.google.com/fir-████████",
  "aud": "fir-████████",
  "auth_time": "████████",
  "user_id": "████████",
  "sub": "████████",
  "iss": "████████",
  "exp": "████████",
  "email": "████████",
  "email_verified": true,
  "firebase": {
    "identities": {
      "email": [
        "████████"
      ]
    },
    "sign_in_provider": "████████"
  }
}

VERIFY SIGNATURE
RSASHA256(
base64urlEncode(header) + "." +
base64urlEncode(payload),
Public Key in PEM, PKCS #1, X.509 Certificate, or JWK string format

Private Key in PKCS #8, PKCS #1, or JWK string format. The key never leaves your browser
)
```

Fig. AII.1: Visualizando datos de un JWT en texto claro.

Seguridad en el BackEnd:

Vídeo sobre autenticación con Spring Boot y Angular:

<https://www.youtube.com/watch?v=yOxxA3DpgnU>

Video para montar API REST con JWT authentication:

<https://www.youtube.com/watch?v=0WYer34XjjM>

Tutorial online sobre colocación de “interceptores en Angular”:

<https://codigoencasa.com/angular-y-jwt/>

Video de autenticación con Spring Boot y Angular, para preparar Back con JWT:

https://www.youtube.com/watch?v=_p-Odh3MZJc

Seguridad en Firebase: reglas de acceso y documentación:

Documentación oficial de Firebase, para configurar las reglas de acceso:

<https://firebase.google.com/docs/rules/basics?hl=es-419>

Ejemplo visual de reglas mínimas necesarias, para el acceso a Firebase (Fig. AII.2):



```
Cloud Firestore  Realtime Database  Cloud Storage
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Fig. AII.2: Reglas de acceso mínimas en Firebase.

:Anexo III: gestión del código fuente.

Es altamente recomendable usar habitualmente un gestor de código fuente, como Git (Fig. 6.2) y GitHub (Fig. 6.3).

Respecto a GitHub, es recomendable crear un nuevo repositorio (público o privado). Después clonar su contenido en tu máquina local dentro, de la carpeta elegida. Ejemplo del comando git a continuación:

git clone [URLRepositorioRemoto]

Sincronización del repositorio remoto con repositorio local: con Git:

- Al comienzo de cada jornada laboral, o antes de realizar cualquier Push a remoto: hacer operación PULL desde remoto para actualizar código local:

git pull [direccionRepositorioRemoto]

- Cuando decida subir un código commiteado a remoto, siempre que la aplicación esté funcionando sin errores:

git push [direccionRepositorioRemoto]

El repositorio GitHub podrá ser un grupo de trabajo donde puedan subir commits varias personas. Éste podrá ser de carácter privado o público.



Fig. 6.2: logo de la aplicación Git.



Fig. 6.3: logo de la aplicación GitHub.

• Anexo IV: buenas prácticas.

- Al comienzo de cada jornada, descargar los cambios más recientes del repositorio remoto (fetch+pull).
- Cuando vayamos a subir código nuevo al repositorio remoto:
 - Asegurarnos del correcto funcionamiento de la aplicación (realizar pruebas).
 - Después, realizar operaciones “*Fetch + Pull*”.
 - Por último, realizar operación “*Push*”.
- Cuando se decida subir de versiones un software, siga los siguientes criterios, partiendo de la versión inicial 1.0.0:
 - Cada vez que se corrija un error detectado en producción, cuando se redesploque en producción el código corregido, la versión subirá a la 1.0.1.
 - Si se detecta un error en el producto que aun no llegó a producción, el error será corregido sin actualizar versión (sigue siendo la versión 1.0.0).
 - Cada vez que desplieguen cambios en producción, los cuales correspondan a mejoras de funcionalidad, se incrementará la versión intermedia. Ejemplo: de la versión 1.0.0 pasaremos a la versión 1.1.0.
 - Cuando se realice un cambio en el código tal que cree incompatibilidades con las versiones anteriores, cambiar la versión mayor. En este caso pasaríamos la versión desde la 1.0.0 a la 2.0.0.
- A la hora de administrar un repositorio git o GitHub, puede realizar una gestión rápida sin mucho control, o bien una gestión más compleja que maximice el control. Según el nivel técnico del usuario:
 - Opción para desarrolladores Juniors (Fig. 6.4):
Gestionar repositorios Git y GitHub de forma sencilla.

Instalar Git + GitHub + SourceTree (o Fork).

Tanto SourceTree como Fork, proporcionan una interfaz gráfica, y son herramientas clientes de Git.



Fig. 6.4: Clientes de Git gratuitos: SourceTree y Fork.

- Opciones para desarrolladores seniors (Fig. 6.5):
 - Gestión de repositorios de forma más compleja, pero con mayor control del proceso.
 - Instalar Git + terminal de comandos.
 - Realización de commits, pull, push, fetch, merge, etc a través de comandos en terminal Git nativo.
 - Con la instalación de Git, el terminal Git viene incluido.



Fig. 6.5: Logo del gestor de versiones Git.

•Anexo V: DevOp.

- Significado del acrónimo inglés “DevOp”: Development Operations.
- ¿Qué es DevOp?
 - Definición 1:

“[...]se refiere a una metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de sistemas en las tecnologías de la información (IT)”.

Fuente: web www.paradigmadigital.com.
 - Definición 2:

“[...] No existe una única herramienta de DevOps, sino un conjunto de herramientas o cadenas de herramientas de DevOps esenciales para los ingenieros de DevOps, los desarrolladores, los operadores y otros miembros del equipo ”.

Fuente: web de www.kinsta.com .

- Errores de concepto sobre DevOp:
 - Pensar que DevOp es solo una herramienta.
 - No es lo mismo “cultura DevOp”, que “filosofía DevOp”.
- Mejoras clave que aporta las herramientas DevOp:
 - Acelera el desarrollo y la entrega de los proyectos.
 - Reduce las fricciones entre los miembros del proyecto.
 - Aumenta la probabilidad de terminación del proyecto con éxito, con mayor satisfacción final por parte del cliente.
- Ventajas de usar DevOp para una organización:
 - Alcance de metas empresariales más rápidamente.
 - Ventaja competitiva en el mercado.
 - Atención más rápida al cliente, incluso mejorando los resultados.
 - Acelera el proceso productivo y la entrega al cliente final.
 - Velocidad sin comprometer la estabilidad y la fiabilidad del sistema: busca el punto medio entre “entornos muy cambiantes que necesitan cambios muy rápidos” y “entornos productivos sólidos sin que sean demasiado rígidos”.
 - Escalabilidad.
 - Seguridad.
 - Mayor satisfacción del cliente.
- Algunas metodologías anteriores a DevOp son:
 - Enterprise Systems Management (ESM, surge aprox. año 2000).
 - Desarrollo Ágil.

- Tabla de herramientas DevOp (Fig. 6.6):



Fig. 6.6: Tabla de Iconos de los programas DevOp más usados.

•Anexo VI: Gestión de proyectos ágiles.

Cuando no es suficiente con la metodología DevOp para integrar el desarrollo rápido con la perspectiva de negocio, se presentan herramientas que nos ayuden con la lógica de negocio (ver desde Fig. 6.7 hasta la Fig. 6.10):

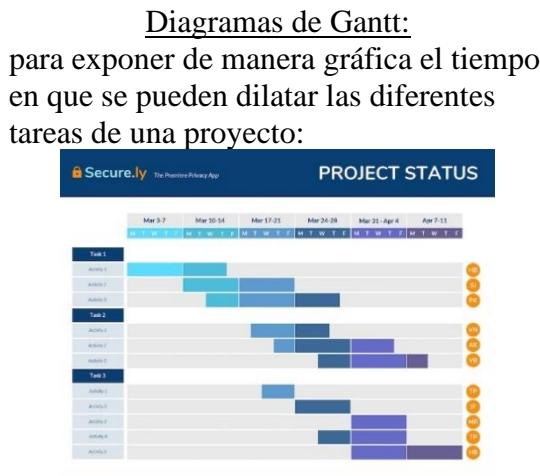


Fig. 6.7: Imagen propiedad de “es.venngage.com”, diagrama Gantt.

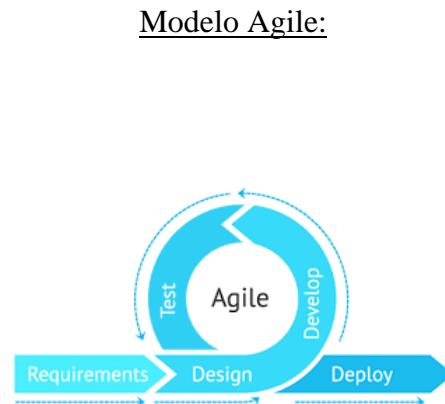


Fig. 6.8: Imagen propiedad de www.shikshaglobe.com, ciclo metodología Agile.

Modelo Canvas:
“*es una plantilla de gestión estratégica para el desarrollo de nuevos modelos de negocio o documentar los ya existentes*”
(citado de Wikipedia).



Fig. 6.9: Imagen representativa del lienzo del modelo Canvas.

Trello:
para administrar proyectos con interfaz web:



Fig. 6.10: Herramienta Trello.

•Anexo VII: Servidores Tomcat y compiladores Java en producción.

- De Java 6 a Java 8 existen cambios muy relevantes dentro de la propia funcionalidad de programas diseñados en Java.
- En relación a las “versiones de servidores Tomcat” también debe tenerse en cuenta, porque Java 11 solo puede ejecutarse en servidores Tomcat de versión 10.1.X o superior (ver Fig. 6.13).

Apache Tomcat Versions							
Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	Authentication (JASPI) Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
6.0	3.1	5.0	2.1	3.0	10.1.x	10.1.0-M16 (beta)	11 and later
5.0	3.0	4.0	2.0	2.0	10.0.x	10.0.22	8 and later
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.64	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.81	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.53 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x (archived)	7.0.109 (archived)	6 and later (7 and later for WebSocket)
2.5	2.1	2.1	N/A	N/A	6.0.x (archived)	6.0.53 (archived)	5 and later
2.4	2.0	N/A	N/A	N/A	5.5.x (archived)	5.5.36 (archived)	1.4 and later
2.3	1.2	N/A	N/A	N/A	4.1.x (archived)	4.1.40 (archived)	1.3 and later
2.2	1.1	N/A	N/A	N/A	3.3.x (archived)	3.3.2 (archived)	1.1 and later

Fig. 6.13: Especificaciones técnicas, compatibilidades de versiones Tomcat - Java.

7. Bibliografía y webgrafía

Documentación sobre Spring Framework & Java 11:

1) Spring, sitio oficial:

<https://spring.io/>

2) Inversión de control con Spring (tutorial):

<https://blog.auriboxtraining.com/java/introduccion-la-ioc/>

3) Descarga del JDK de Java 11, versión OpenJDK:

<https://adoptopenjdk.net/>

Documentación sobre Servidores Web y de Aplicaciones:

1) Comparativa entre servidores:

<https://www.iteramos.com/pregunta/5731/cual-es-la-diferencia-entre-tomcat-y-jboss-y-glassfish>

2) Descarga de servidores Apache:

<https://tomcat.apache.org/download-80.cgi>

3) Servidores TOMCAT:

<https://sites.google.com/site/tecceilpii/home/0---temas-de-investigacion/tomcat>

4) Todo sobre los Servidores Web Embebidos:

<https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto.webserver>

5) Configurar un Tomcat embebido en un Spring Boot:

<https://www.adictosaltrabajo.com/2017/08/01/configurar-el-tomcat-embebido-de-spring-boot/>

6) Tabla de compatibilidad entre versión de Java (JRE) soportada por cada versión de Apache Tomcat:

<https://tomcat.apache.org/whichversion.html>

7) Tutorial acerca de cómo conectar API REST desarrollada en Java, mediante cliente FrontEnd desarrollado en Angular:

<https://codingpotions.com/angular-servicios-llamadas-http>

Comparativas funcionales entre diversos lenguajes de programación:

1) Ventajas y desventajas de JavaScript puro:

<https://www.nextu.com/blog/conoce-las-ventajas-y-desventajas-de-javascript/>

2) Comparativa JavaScript vs jQuery:

<https://openwebinars.net/blog/diferencias-javascript-jquery/>

3) Comparativa JavaScript vs Angular:

<https://www.educba.com/javascript-vs-angularjs/>

4) Comparativa de Angular JS vs jQuery:

<https://eladrodriguez.gitbooks.io/angularjs/content/1-introduccion-angularjs/1-3-angularjs-jquery.html>

5) Comparativas de AngularJS vs React js (parte 1):

<https://www.freecodecamp.org/espanol/news/angular-vs-react-cual-elegir-para-su-aplicacion/>

6) Comparativas de AngularJS vs React js (parte 2):

<https://www.toptal.com/front-end/angular-vs-react-cual-es-mejor-para-el-desarrollo-web>

Servicio HTTP:

1) Descarga del cliente HTTP “Insomnia Core”:

<https://insomnia.rest/download>

Documentación sobre IDE's:

1) Qué es un IDE:

https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

2) Documentación del IDE Visual Studio Code:

<https://www.xatakawindows.com/aplicaciones-windows/microsoft-actualiza-visual-studio-code-nuevas-combinaciones-teclas-mejoras-terminal>

Documentación de herramientas y filosofía DevOp:

1) Explicación de la filosofía DevOp y presentación de herramientas DevOp:
<https://kinsta.com/es/blog/herramientas-devops/>

2) Artículos explicativo de qué es y qué no es DevOp:

<https://www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops/>

3) Presentación del “ingeniero DevOp”, qué es, funciones y futuro de este perfil:

<https://openwebinars.net/amp/blog/que-es-un-ingeniero-devops-y-que-funciones-tiene/>

4) Estándar de subida de las versiones de desarrollo de software:

<https://ed.team/blog/como-se-deciden-las-versiones-del-software>

Documentación sobre arquitecturas y metodología CI/CD:

1) Artículo de RedHat sobre CI/CD:

<https://www.redhat.com/es/topics/devops/what-is-ci-cd>

Documentación y blogs de ciberseguridad:

1) Medidas de ciberseguridad básicas para cualquier empresa:

<https://www.datos101.com/blog/las-9-medidas-de-seguridad-informatica-basicas/>

2) Sitio oficial de INCIBE (Instituto Nacional de Ciberseguridad):

<https://www.incibe.es/protege-tu-empresa/blog>

3) Blog oficial de Chema Alonso (divulgador ciberseguridad):

<https://www.elladodelmal.com/>

4) Divulgadores más relevantes en ciberseguridad:

- **Chema Alonso.**
- **Marcelo Vázquez.**

Documentación de Firebase:

- 1) Lista de documentación:

<https://console.firebaseio.google.com/u/0/project/fir-demo-tfm.firebaseio/data/~2F?hl=es>

- 2) Tutorial concreto, usado en apartado 4.2.f) para configurar SDK de Firebase en Spring Boot (el proyecto backend Java):

<https://firebase.google.com/docs/admin/setup#windows>

Tutoriales de programación desde Youtube:

- 1) Diseño del Login iniciado desde Angular:

<https://www.youtube.com/watch?v=l808gGh9RTU>

- 2) Asegurando la API REST con Bearer Token (JWT):

<https://www.youtube.com/watch?v=0WYer34XjjM>

- 3) Inyectar un video de Youtube en página HTML:

<https://www.youtube.com/watch?v=8pCzrNP2K1E&feature=youtu.be>

Bibliografía y Libros:

- 1) **JAVA, The Complete Reference – Tenth Edition**

(Ed. Mc Graw Hill; Auth. Herbert Schildt).

- 2) **JavaScript, The Definitive Guide**

(Ed. O'REILLY; Auth. David Flanagan).

- 3) **Comenzando Angular con TypeScript**

(Ed. ENI; Auth: Jullen Caliendo).

- 4) **El libro del Hacker**

(Ed. ANAYA; Auth: María Ángeles Caballero y Diego Cilleros Serrano)