Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Interfaz Web de Intérprete de Comandos de MATLAB

Autor: Alfonso Lerma Trevilla

Tutor: José Ramón Cerquides Bueno

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Interfaz Web de Intérprete de Comandos de MATLAB

Autor:
Alfonso Lerma Trevilla

Tutor:
José Ramón Cerquides Bueno
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Interfaz Web de Intérprete de Comandos de MATLAB

Autor: Alfonso Lerma Trevilla

Tutor: José Ramón Cerquides Bueno

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

Agradecimientos

En primer lugar, he de agradecer a Ramón Cerquides la confianza y paciencia depositadas a lo largo del periodo de incertidumbre con respecto al rumbo del proyecto. Por último, es justo agradecer también a toda la comunidad de personas e instituciones que contribuyen al desarrollo del software libre, ya que, además de este proyecto, el mundo en el que vivimos se sostiene gracias a ellas.

Resumen

En un principio, el presente trabajo de fin de grado estaba basado en un programa de MATLAB que implementaba un sistema de reconocimiento de huella digital de audio, o audio fingerprint [1] [2]. El programa era capaz de identificar, usando el micrófono del ordenador, pistas de audio que hubieran sido previamente almacenadas en la memoria de programa, simulando una base de datos. Prácticamente se trataba de la misma funcionalidad que el software comercial Shazam, de hecho, el artículo con el cual comenzó la investigación para este trabajo es el que presentaba dicha tecnología [3].

La necesidad de facilitar el uso del programa y dar usabilidad a una pieza de software en la que se había empleado cierto esfuerzo, motivó la búsqueda de una manera de implementar una interfaz gráfica. Estudiando a fondo las herramientas que brinda MATLAB para convertir uno o varios scripts en una aplicación, se llegó a compilar como ejecutable para Windows, resultando en un programa que, aun funcionando correctamente apenas imprimía algo de información en la terminal, y la experiencia de uso era poco satisfactoria. El mal rendimiento de dicha aplicación sirvió de estímulo para seguir probando otros caminos, en concreto, organizar el código en un MATLAB Live Script, de forma que se pudiera visualizar un desglose de las etapas del procesamiento para la extracción de fingerprints, incluyendo representaciones gráficas. El análisis de las peculiaridades de los live scripts inspiró una inquietud que permitió tomar la decisión que más sentido pudo aportar al proyecto: salir de MATLAB.

Salir de MATLAB se sentía como una buena idea tanto como una obsesión. En primer lugar, técnicamente hablando, aun pareciendo buena idea, carecía de significado. Un motor de MATLAB, o MATLAB Runtime, ¿puede ser invocado desde cualquier "lugar" de un sistema operativo? ¿Es necesario tener abierta la aplicación MATLAB? ¿Se puede usar en sí mismo el lenguaje MATLAB desde cualquier otro entorno de programación? En otras palabras, ¿existe algo parecido a una API (Application Programming Interface) para comunicarse con el interior de MATLAB desde una aplicación externa?

El entorno integrado de programación (IDE) Visual Studio Code, en adelante VSCode, de licencia libre, que es utilizado ampliamente en todo el mundo por su versatilidad, cuenta con una gran cantidad de extensiones, y resulta que algunas de ellas, brindan soporte para MATLAB. En concreto, inspeccionando la extensión "MATLAB Interactive Terminal" [4] que no es más que un fragmento de código de Python, se halló la pieza que dio sentido a este rompecabezas: MATLAB Engine API para Python. Este motor es simplemente un paquete de Python que, tal y como su nombre indica, permite ejecutar funciones de MATLAB, y es uno de los requisitos para poder usar correctamente la extensión de VSCode.

Una vez que fue posible ejecutar con éxito funciones propias de MATLAB desde un IDE externo, incluyendo el sistema de audio fingerprint, se llegó a un punto de inflexión en el desarrollo del proyecto [5]. Estando en un intérprete de comandos de Python, y conociendo las potentes herramientas o frameworks (entornos de trabajo) que existen para programar aplicaciones con dicho lenguaje, la ocurrencia de diseñar y desarrollar una interfaz web para acceder a MATLAB era prácticamente obvia. Aun existiendo MATLAB Online, que pagando una licencia ya ofrece exactamente eso, la motivación de hacerlo con tecnología libre y moderna fue suficiente para considerarlo algo útil.

De modo que, finalmente, el presente proyecto ofrece un prototipo de interfaz web para comunicarse con un motor de MATLAB. El sistema cuenta con la estructura típica de *frontend* y *backend* de una aplicación web, pudiéndose considerar como backend la combinación de la aplicación Python que hace uso del MATLAB Engine API, y el motor en sí. Dicho backend se ha implementado usando FastAPI, un framework moderno y de alto rendimiento para desarrollar APIs en Python [6]. Para la parte de la interfaz de usuario o frontend, por comodidad y por existir algo de experiencia profesional previa, se ha optado por usar Angular, una de las plataformas de desarrollo de aplicaciones web más usadas y con más soporte hoy en día. Dicha interfaz de usuario será la que permita disfrutar de una experiencia más parecida al uso típico de la aplicación MATLAB, ya que cuenta con el intérprete de comandos y la visualización de figuras.

Abstract

Initially, this project was based on a MATLAB program that implemented an audio fingerprinting system. The code was able to identify previously stored audio files using the computer's built-in microphone, just the way comercial software Shazam does. In fact, the article that triggered this projects' research process is the one that introduced that technology in 2001.

The quite poor experience that piece of software offered, motivated the research of implementing a user interface. At one point, after going deep into the available tools MATLAB and MathWorks provide, it was possible to compile the whole set of scripts into one single Windows executable file, but even working properly as a console application, it merely prompted some text, and the user experience was nonexistent. Furthermore, the bad performance encouraged other research paths, for example setting up a MATLAB Live Script, so a step-by-step breakdown of the program could be analized and even visualized. Exploring and the main particularities of the aforementioned Live Script allowed to make the most helpful decision so far: stepping out of MATLAB.

Stepping out of MATLAB felt as a good idea the same way it felt like some sort of obsession. First of all, technically speaking, while being a good plan, it was meaningless. Can a MATLAB engine or MATLAB Runtime be invoked within "anywhere" in an operating system? Is it required to open MATLAB desktop window? Can MATLAB language be used itself from any other environment? In other words, is there such thing as an API (Application Programming Interface) that allows to communicate with MATLAB within an external application?

The open source integrated development environment (IDE) Visual Studio Code, or VSCode, which is globally spread beacuse of its versatility, has a wide variety of extensions to add support for different languages, and luckily some of them provided tools related to MATLAB. More precisely, going deep into "MATLAB Interactive Terminal", which is just a short piece of Python code, it was found the main character of this puzzle: MATLAB Engine API for Pyhton. This engine is just a Python package that, as its name suggests, allows to run MATLAB functions.

Once it was posible to run MATLAB functions succesfully from VSCode, including the audio fingerprinting system, a turning point was reached. Being placed in a Python command line interface, and, taking a look at the powerful open source state-of-the-art tools available to build applications with that language, the idea of designing and developing a web interface to access MATLAB was essentially obvious. Even with the existence of MATLAB Online, which offers something similar, the fact of doing it with modern and open source technology was enough for it to be considered something.

So, finally, this project delivers a prototype for a web application which interfaces with a MATLAB engine. The system is structured as a typical frontend-backend web stack, considering as backend the combination of the Pyhton application that makes use of the MATLAB Engine API, and the engine itself. This backend was implemented using FastAPI, a high performance framework designed to build APIs using Python language. For the graphical user interface or frontend, for the sake of simplicity and not having to learn yet another new technology, Angular, a widely supported web development platform, was chosen. This user interface would take it closer to the classic MATLAB application use, providing command-line interpreter functionality and the visualization of figures.

Índice

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Establecer la comunicación entre dos dispositivos o entidades podría considerarse como la esencia de las tecnologías que se estudian en nuestro ámbito. En el mundo del software, prácticamente, a lo que se dedica la mayoría de las personas que están involucradas es a escribir una pieza de código que sirva de conexión o traducción entre una entidad que procesa la información de una determinada manera, y otra entidad que lo hace de otra. Cuando un sistema concreto no ofrece suficientes funcionalidades o presenta un cierto grado de complejidad para hacer de él un uso coherente y cotidiano, la forma natural de caracterizarlo es creando capas de abstracción, es decir, una vez que se conocen las funciones básicas de un sistema, la combinación adecuada de éstas puede (y debe) dar lugar a nuevas utilidades tan fundamentales y valiosas como las originales. Un sistema compuesto por un entorno de computacion numérica y un interprete de comandos como MATLAB, recibe a lo largo de su existencia constantes mejoras y actualizaciones, muchas de ellas relacionadas con la naturaleza de los datos que se desean computar, con la visualización de los datos, exportación de programas y aplicaciones completas, despliegue de aplicaciones web con su propio servidor, y muchas más. El problema está en que todas esas valiosas herramientas son software propietario, es decir, para poder disfrutar de ellas como usuario particular, es necesario un notorio desembolso o pertenecer a una comunidad educativa concreta que cuente con un convenio con MathWorks.

1.1 Objetivos

El objetivo del proyecto podría considerarse el diseño y prototipado de una aplicación web para comunicarse con un motor de MATLAB, disfrutando de una experiencia similar al uso de la aplicación convencional, pero con herramientas open source. La singular característica de este sistema es que varios usuarios, o literalmente varias ventanas del navegador, pueden tener su propia sesión de MATLAB en sus dispositivos con una sola licencia instalada en un servidor remoto. En este contexto, el término sesión debe ser definido: se considera una sesión como la combinación de una instancia de MATLAB ejecutándose (un proceso *matlab.exe*) y la parte del frontend que haga una referencia a ese proceso en segundo plano. En otras palabras, cada usuario dispondrá de sus propias variables, figuras e intérprete de comandos aislados de los demás.

En cuanto al despliegue o puesta en producción de la aplicación completa, queda fuera del alcance del proyecto, ya que el prototipo será desplegado en un entorno local Windows para servir de prueba de concepto. Sería ideal hacer uso de Docker, una tecnología que permite desplegar servicios web en contenedores aislados, similar a una máquina virtual, asegurando que el entorno en el que una aplicación se ejecuta es siempre el mismo, tiene siempre la misma configuración, y tiene visibilidad desde y hacia todos los servicios que interactúan. Cabe mencionar que, con herramientas no oficiales, existe la posibilidad de instalar MATLAB en una máquina Linux, por lo que el despliegue del sistema sería una tarea más segura y precisa.

Dado que MATLAB ya ofrece la posibilidad de exportar el workspace a un archivo que después se puede importar en una sesión en blanco, queda también fuera del alcance del proyecto la implementación de un sistema de persistencia de información, como por ejemplo una base de datos.

1.2 Estructura del documento

Capitulo 2: Arquitectura del Sistema

- **2.1 Pila tecnológica**
- **2.2 Arquitectura de los servicios y aplicaciones**

Capítulo 3: Diseño e Implementación

- **3.1 Backend**
- **3.2 Frontend**

Capítulo 4: Despliegue y Acceso al Servicio

- **4.1 Despliegue**
- **4.2 Acceso al Servicio**

Capítulo 5: Conclusiones

- **5.1 Carencias**
- **5.1 Valoración del Autor**

Capítulo 6: Anexo

- **6.1 Código fuente**

2 ARQUITECTURA DEL SISTEMA

En este capítulo se define la estructura de la aplicación web completa, también conocido como pila tecnológica. En ella se puede ver las distintas tecnologías de software que se están usando, la forma concreta de interconectarlas, y el mapa lógico-físico de las entidades o dispositivos que albergan esos servicios.

2.1 Pila tecnológica

Como se ha mencionado anteriormente, el sistema está compuesto por tres aplicaciones: la interfaz gráfica, la API que expone las funcionalidades del motor de MATLAB, y el motor en sí como proceso en segundo plano.

2.1.1 Frontend

2.1.1.1 Contexto

En lo que respecta a este proyecto, la elección de la tecnología para crear la interfaz gráfica de la aplicación no es algo crítico. Hoy en día se da por hecho que una aplicación web integrable y escalable en un entorno mínimamente profesional estará escrita en JavaScript, haciendo uso de alguno de los frameworks que están disponibles y que cuentan con soporte y constante actualización por parte de la comunidad y de las instituciones que los desarrollan. De entre ellos, los que hoy en día destacan y son conocidos por cualquier individuo mínimamente relacionado con este ámbito son React y Angular. La principal diferencia entre ellos, además de la subjetiva pugna de popularidad que sufre cualquier tecnología tras pasar por el uso de la comunidad del software, es que React aporta un framework para desarrollo de interfaces de usuario, mientras que Angular consiste en una plataforma completa de desarrollo de software. En otros términos, React, aunque tecnológicamente presenta muchas ventajas en cuanto a rendimiento y popularidad, no deja de ser una librería, que requiere de otras librerías a su vez para poder formar parte de una solución software completa. Esto aporta una libertad de implementación del código que Angular no ofrece, al menos a priori. En React, la manera por defecto que tienen de interactuar los elementos de la interfaz y la lógica de los componentes es unidireccional, es decir, la lógica determina el estado de la interfaz. Sin embargo, en Angular, si se modifica el valor de una propiedad en la vista o interfaz, también se ve afectado el modelo de datos. Esta forma de fluir la información también provoca que Angular tenga que actualizar todo el árbol de elementos de la interfaz cuando hay un cambio, mientras que React tan solo actualiza aquel elemento que haya sufrido una modificación con respecto al estado anterior. Hablando del soporte para multiplataforma, existe un framework conocido como React Native, o nativo, que desde hace años es una acertada opción para el desarrollo de aplicaciones móviles. Su rendimiento es próximo a una aplicación nativa, o desarrollada expresamente para el hardware en el que va a ejecutarse. Angular, sin embargo, no cuenta con un framework tan sólido sobre el que desarrollar software multiplataforma, aunque sí algunas opciones como NativeScript o Ionic. El hecho de que Angular esté escrito en TypeScript, un lenguaje orientado a objetos, y que cuente con las herramientas necesarias para crear una aplicación compleja sin necesidad de librerías externas, fueron los factores concretos que facilitaron la decisión. Para la realización de un prototipo o prueba de concepto sin nociones sólidas sobre desarrollo de frontend, es probable que React supusiera un sobreesfuerzo, ya que permite un grado más alto de personalización, y por tanto requiere mayor destreza y experiencia.

2.1.1.2 Descripción

La elección de Angular como plataforma sobre la que desarrollar la interfaz de usuario se debe a la facilidad de implementación y buen rendimiento que ofrece esta herramienta. Angular es una plataforma de desarrollo escrita en TypeScript, un superset sintáctico de JavaScript orientado a objetos creado por Microsoft. JavaScript es un lenguaje compilado just-in-time con tipos dinámicos, que es interpretado directamente por el navegador. TypeScript encapsula dicho lenguaje y otras funcionalidades como el tipado estricto, y cuando se ejecuta, antes es transpilado a JavaScript. Angular, como plataforma, incluye un framework basado en componentes, una colección de librerías ya integradas, y una suite de herramientas de desarrollo con las que escribir, compilar, testear y desplegar la aplicación. En el ámbito del desarrollo de interfaces gráficas en la web, un componente es un bloque fundamental de la aplicación. Está compuesto por una clase TypeScript, una plantilla HTML y estilos, prácticamente los pilares básicos de una aplicación web clásica. En el archivo HTML se describe qué elementos gráficos se van a renderizar en una determinada pantalla o pestaña, mientras que el archivo TypeScript, que es en el que se define la lógica del componente, es el que rige el comportamiento y el contenido de los elementos que se renderizan, así como lo que sucede cuando se disparan los eventos que conciernen a la aplicación. Al igual que en la mayoría de los lenguajes y frameworks de desarrollo web, las entidades que se definen están pensadas para ser reutilizadas, de forma que el código se vaya creando de la forma más escalable y generalizable posible. Cabe destacar que Angular incluye en su librería base el soporte para el enrutado entre pantallas o vistas, la gestión y validación de formularios, y un robusto cliente HTTP con el que implementar funcionalidades de comunicación cliente-servidor.

El módulo o librería de Angular responsable de realizar las peticiones al backend es HttpClientModule, en concreto, el HttpClient es quien se encarga de las peticiones, y está disponible para importarlo como clase allá donde se necesite. [7] La peculiaridad de ésta y de otras tecnologías de frontend, es que las transacciones hacia un servicio externo se realizan mediante el concepto de Observable y Subscriber. El Observable es un tipo de dato que se usa para la gestión asíncrona de eventos, y se puede "observar" subscribiendolo a un Subscriber. Un Subscriber recibe o consume el cambio de estado de un Observable y la gestión del estado del Observable se realiza en los callbacks del Subscriber. Una función callback es un fragmento de código que se ejecuta únicamante cuando se cumple un evento concreto de un Subscriber. [8] A la hora de hacer una petición al backend, para el propósito de este proyecto, solo se usará la función HttpClient.get(), que devuelve un Observable, y se puede obtener la respuesta de la petición con el método subscribe(). A lo largo de esta aplicación, se usarán los callbacks result y error, que contendrán el cuerpo de la respuesta de la petición, sea exitosa o errónea, respectivamente.

En la descripción de la implementación de la aplicación frontend se profundizará en la utilidad de este concepto, por ahora, es suficiente comprender que la gestión de las transacciones entre cliente-servidor la controla el módulo HttpClientModule.

Los elementos gráficos que renderiza Angular deben crearse a mano o bien ser importados desde librerías de componentes. De entre las más conocidas hoy en día, como MaterialUI, desarrollada por Google, o PrimeNG, desarrollada por PrimeTek, se elige la segunda por simplicidad. Para importar un componente, basta con añadir el módulo en el fichero AppModule, como se describirá en secciones posteriores del documento.

Para la instalación de las herramientas para desarrollar la aplicación frontend, es necesario incluir la herramienta más importante para cualquier sistema basado en JavaScript hoy en día, Node.js. Se trata de un motor de ejecución orientado a eventos asíncronos, escrito en JavaScript, y diseñado para construir sistemas escalables de red, entre ellos aplicaciones web o incluso frameworks. Junto con Node.js, es necesario contar con NPM, Node Packet Manager, la herramienta con la que podemos instalar cualquier dependencia, librería, o framework que pertenezca a Node. En el caso de una máquina Windows y para el propósito de esta aplicación, bastará con el instalador, que actualiza las variables de entorno para usar NPM y Node desde cualquier lugar del sistema operativo [9].

Tras la instalación de Node, es necesario instalar Angular, que se consigue con el comando:

```
npm install -g @angular/cli
```

2.1.2 MATLAB Engine

2.1.2.1 Contexto

Existen diversas opciones a la hora de integrar MATLAB en un intérprete de comandos o entorno de desarrollo de otro lenguaje. Obviamente, primero es necesario contar con la aplicación MATLAB instalada y con una licencia activa. MathWorks ofrece la posibilidad de acceder a la API de MATLAB desde entornos como C++, Java, Python, C, Fortan y algunos más. [10] Como se menciona en el resumen, la elección del motor de MATLAB para Python fue una acertada casualidad, ya que era un requisito para poder usar la extensión de VSCode que permitía acceso a un intérprete de comandos de MATLAB. [4]

2.1.2.2 Descripción

La máquina que albergue la aplicación debe contar con una instalación de Python3.8, aunque dependerá de la versión de MATLAB instalada. [11] Al igual que con Node.js, el instalador de Pyhton3.8 para Windows será la opción adecuada para las necesidades del proyecto. Además, viene añadida en la instalación el gestor de paquetes de Python, PIP. Similar a NPM en el caso de Node, PIP permite añadir librerías y dependencias adicionales que no se distribuyen con Pyhton por defecto. Una vez se complete ese paso, basta con lanzar un script de Python que viene por defecto con la instalación de MATLAB. En la ruta del programa, que aun pudiendo variar según el sistema, tendrá esta apariencia: `"C:\Program Files\MATLAB\R2018a\extern\engines\python"`, se ejecuta el comando:

```
python setup.py install
```

Tras esto, ya será posible importar el motor de MATLAB en un script de Python y comenzar a interactuar con el intérprete de comandos. [12]

Existen algunas limitaciones con respecto a las funcionalidades que permite el motor de MATLAB para Python, entre ellas [13] :

- El motor no puede inicializar MATLAB si ambos se encuentran en máquinas distintas. Es por ello por lo que la API debe ser contenida en el entorno que cuente con la instalación de MATLAB
- No es posible utilizar palabras clave de Python como argumentos de entrada a funciones de MATLAB. En caso de coincidencia, como por ejemplo el comando `print`, la lógica interna se decanta por el comando de MATLAB.
- El tamaño de los arrays que se transfieren entre Python y MATLAB está limitado a 2 gigabytes.
- Una estructura de datos que se contiene a sí misma como valor, es decir, una estructura recursiva, no puede ser utilizada como argumento de entrada a funciones de MATLAB, o ser declaradas en el workspace.

2.1.3 API

2.1.3.1 Contexto

La labor de una interfaz de programación de aplicación es hacer las funcionalidades de un sistema accesibles desde otro tipo de software. Desde hace dos décadas existe un modelo de arquitectura web que se implementa sobre el concepto de API, y que rige la naturaleza de la comunicación cliente-servidor. Esta tecnología se conoce como REST: Representational State Transfer, y el paradigma consiste en que el servidor concede al cliente el acceso a recursos mediante peticiones HTTP. [14] Estos recursos se sirven en formatos de hipertexto tales como XML, JSON, o incluso texto plano. El formato de intercambio de datos que ha prosperado en el mundo del desarrollo web es el JSON, o JavaScript Object Notation, cuyo fundamento es que usa el lenguaje humano para almacenar y transmitir texto, que al fin y al cabo resulta algo obvio. Es por ello por lo que se categoriza como autodescriptivo, y es totalmente independiente del lenguaje en el que se use, ya que es en esencia una estructura de datos universal. [15] Se define como una estructura nombre/valor, que relaciona el nombre de un atributo con su valor. Por ejemplo, un elemento JSON de nombre ejemplo con un atributo edad cuyo valor sea 10, obviando la sintaxis concreta del lenguaje en el que se implemente, se podría definir como:

```
ejemplo = {"edad": 10, "nombre": "Alfonso"};
```

Y el acceso a los atributos del objeto JSON sería algo parecido a:

```
ejemplo["edad"]
ejemplo["nombre"]
```

El concepto matemático de función, que no es más que un proceso o relación que asocia un conjunto de elementos biunívocamente con otro conjunto de elementos, tiene algo que ver en esta estructura de datos, a diferencia que no es posible que exista un valor para el cual no haya un nombre, mientras que un nombre puede relacionarse con un valor vacío, que aun vacío, es distinto de nada.

Volviendo a la descripción de la arquitectura REST, los requisitos para que una API se considere como *RESTful* son:

- Una arquitectura compuesta por clientes, servidores y recursos, comunicándose mediante peticiones HTTP.
- Comunicación cliente-servidor no orientada a estado, de forma que no se almacena información entre peticiones, que serán siempre independientes e inconexas.
- La posibilidad de guardar datos en un caché que simplifique interacciones cliente-servidor.
- Una interfaz uniforme que reciba y devuelva datos de forma consistente, con recursos biunívocamente identificables e instrucciones autodescriptivas para que el cliente conozca el estado y resultado de cada petición.
- Un sistema jerárquico que estructure y compartimente las distintas capas del servidor, tales como seguridad, acceso a datos o la propia lógica de negocio, que se suele denominar al código interno de la aplicación, o, en otras palabras, lo que "hace".

Los tipos de métodos HTTP que se utilizan en cualquier arquitectura basada en REST son:

- GET: Ofrece acceso de solo lectura a un recurso.
- POST: Permite crear un nuevo recurso.
- PUT: Se usa para actualizar un recurso existente, aunque también puede servir para crear uno nuevo en caso de no existir.
- DELETE: Como su nombre indica, sirve para eliminar un recurso.

Una función que pertenece a una API y que está configurada o diseñada para atender una petición concreta, se denomina endpoint. Un endpoint consiste en una URL mediante la cual se direcciona un recurso dentro de un servicio o aplicación. Se trata de un concepto análogo al de un archivo dentro de un directorio, es la ruta que debe seguir el navegador para dar con el recurso que representa la información solicitada.

Existen varias maneras de enviar un parámetro como argumento de una llamada a un endpoint, cada una reservada para un uso, aunque a la hora de implementarlo no existe ninguna restricción, tan solo la que recomienda el paradigma. Cuando se desea especificar un recurso a través de su identificador en el sistema, ese dato se añade como variable de ruta, con la estructura:

```
GET dirección:puerto/nombreRecurso/id
```

Un ejemplo para acceder al usuario cuyo id es 5 sería:

```
GET localhost:8000/usuario/5
```

Si lo que se desea no es acceder a un recurso a través de su identificador, sino que se desea filtrar un recurso a través de uno de sus atributos, se utilizan parámetros de query, cuya sintaxis es:

```
GET dirección:puerto/nombreRecurso?atributo=valor
```

De esta forma, podría obtenerse el o los usuarios cuya edad sea igual a 10 mediante la petición:

```
GET localhost:8000/usuario?edad=10
```

En general, se definen como parámetros de ruta aquellos identificadores que no van a cambiar a lo largo del periodo de vida del sistema, y como parámetros de query aquellos atributos variables que, según la finalidad del endpoint, darán un significado u otro al resultado obtenido. [16]

Existen bastantes opciones a la hora de desarrollar un backend o una API web, más si no se cuenta una preferencia o restricción tecnológica de peso. En el entorno profesional, se siguen utilizando tecnologías como PHP, .NET, Python y Java. En el ámbito de las aplicaciones de infraestructura pública, tales como los servicios de la Consejería de Salud de la Junta de Andalucía o muchas instituciones educativas, sigue predominando Java, el que ha sido el lenguaje muliplataforma por excelencia. En concreto, Spring es el framework de Java que más se ha popularizado en el ámbito del desarrollo web. Spring aporta todas las funcionalidades necesarias para construir una aplicación web de backend, desde el núcleo del lenguaje Java hasta la compatibilidad con distintas bases de datos o tecnologías añadidas. En los últimos años se ha podido apreciar como Python y JavaScript han visto incrementados su uso, soporte y popularidad con respecto a Java. Las modernas y cómodas herramientas de alto rendimiento que existen hoy en día para desarrollar no tienen ni punto de comparación con la complejidad y delicadeza de un sistema hecho sobre Java. El soporte de la comunidad y lo extendido que están Java y Spring Framework se debe a su antigüedad y a haberse instaurado como lenguaje de infraestructura en muchos países, pero no porque sea un lenguaje eficiente a la hora de funcionar en una máquina. Python, sin embargo, se ha convertido en una potente herramienta todoterreno, que poco a poco se ha ido instaurando de forma sólida en muy diversos ámbitos de la ingeniería del software, tales como el procesado de datos, sistemas de aprendizaje automático, y aplicaciones web. Teniendo en cuenta que el núcleo de este proyecto es un motor de MATLAB para Python, la decisión de buscar un framework de desarrollo web basado en Python fue casi trivial. Cualitativamente hablando, el grado de complejidad que suponía el hecho de implementar sobre una aplicación escrita en Python una API web en un lenguaje distinto fue suficientemente ahuyentador.

2.1.3.2 Descripción

La inclinación por FastAPI como interfaz entre Pyhton y la web se basó, en parte, en una importante característica de este framework: el soporte para implementar código de ejecución asíncrona, algo fundamental para una aplicación que actuará de servidor para un numero arbitrario de clientes. La asincronía consiste en la ejecución de un fragmento concreto de código de forma paralela al hilo principal de la aplicación. Este proceso asíncrono notificará al hilo principal cuando haya finalizado, permitiendo así un control exhaustivo sobre el flujo del programa. Hasta hace poco, el estándar para implementar una interfaz entre un servidor web y una API en Python era WSGI, Web Server Gateway Interface, que operaba en modo síncrono, pero su sucesor, ASGI, Asynchronous Server Gateway Interface, aporta la misma funcionalidad, además de la gestión asíncrona de peticiones [17] [18] [19] [20].En concreto, tal y como recomienda la documentación de FastAPI, se usará Uvicorn, una implementación de un servidor ASGI. Este paquete de Pyhton es fácilmente instalable y es el que servirá para ejecutar y desplegar la aplicación backend en un servidor local. Otra ventaja de FastAPI es la facilidad de implementación, dado que es una tecnología open source y actual, cuenta con herramientas de alto rendimiento sumamente cómodas para crear un proyecto en blanco tecleando pocos comandos y comenzar a desarrollar al instante. [21]

Como ya debe haber una versión de Pyhton3.8 o superior en el sistema operativo, la instalación de FastAPI y Uvicorn, gracias al gestor de paquetes PIP, se resume a la ejecución de dos comandos:

```
pip install fastapi
pip install uvicorn
```

En este punto ya se podrá importar la librería de FastAPI desde cualquier script de Pyhton, con lo que concluye la instalación de las herramientas que constituyen la aplicación completa.

La instalación de FastAPI, entre otras tecnologías, incluye Swagger UI, que se trata de una herramienta que permite visualizar e interactuar con una API sin necesidad de existir un frontend u otra lógica adicional implementada. Esto es útil para hacer peticiones reales al backend desde un navegador y llevar a cabo el desarrollo del código de forma coherente y compartimentada.

2.2 Arquitectura de los servicios y aplicaciones

Para el propósito de este proyecto, las tres aplicaciones que componen el sistema se ejecutarán en la misma máquina, y el frontend y la API están configuradas para que arranque cada una en un servidor local con la misma dirección de red, diferenciándose por el número de puerto. La dirección de red de dicho dispositivo, al ser la misma en el caso del frontend y del backend, se trata de la dirección de la máquina propia, que, en cualquier sistema, tiene el alias `localhost`, que por defecto se traduce a `127.0.0.1` [22]

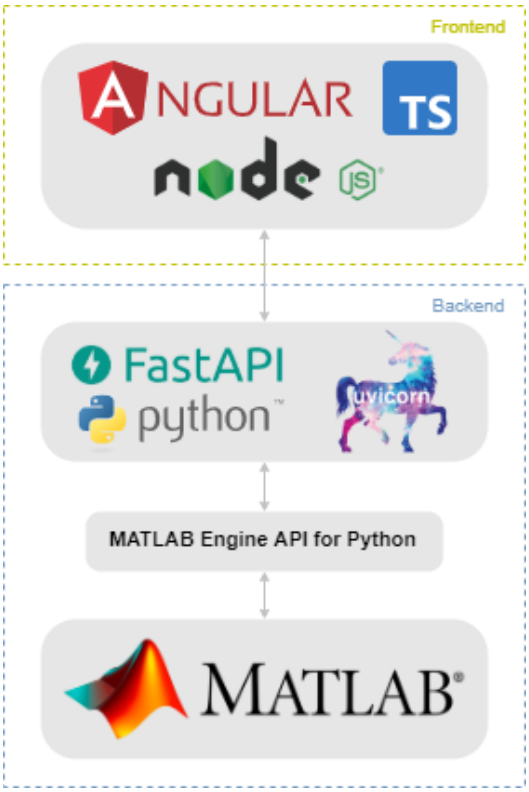

Figura 1: Arquitectura del sistema

3 DISEÑO E IMPLEMENTACIÓN

La principal ventaja de haberse decantado por el motor de MATLAB para Python es que el código de la extensión de VSCode previamente mencionada, ya proporciona un punto de partida sólido sobre el que comenzar a desarrollar el backend. Una vez que es posible realizar peticiones a la API, implementar la relación entre una sesión de frontend con un proceso de MATLAB, y recibir con éxito la respuesta de los comandos del intérprete diferenciados según sesión, se construye el frontend, que será la aplicación que solicite datos del backend para representarlos en la interfaz de usuario.

3.1 Backend

3.1.1 MATLAB Engine API for Python

Lo primordial a la hora de diseñar una interfaz entre sistemas con naturaleza aparentemente distinta, es decidir de qué funcionalidades se va a prescindir. Esto ayuda a establecer un límite en el grado de generalización que se va a alcanzar con el sistema. De entre las funciones del motor de MATLAB, se usan las siguientes:

- `start_matlab()`: Esta función inicia un proceso de MATLAB en segundo plano y si lo hace con éxito devuelve un objeto de tipo `MatlabEngine`. Para arrancar otro motor en un proceso independiente, basta con invocar la función y asignar el resultado a otra variable distinta, y así sucesivamente. En otras palabras, existirán tantos procesos de MATLAB como instancias de `MatlabEngine` se creen en la aplicación. Los argumentos que toma no son relevantes para su uso en esta aplicación, por lo que se usará sin argumentos de entrada.
- `quit()`: Finaliza el proceso de MATLAB y borra el contenido de la variable que almacenaba la instancia correspondiente de `MatlabEngine`.
- `eval(command,nargout,stdout,stderr)`: Esta es la misma función del lenguaje MATLAB que permite evaluar una expresión. La variable `command` contiene una cadena de texto equivalente a la ruta de un script, o al nombre de uno o varios comandos, separados por punto y coma tal y como se escribirían en la interfaz de comandos clásica de MATLAB. El argumento `nargout` debe ser siempre igual al número de argumentos de salida que se esperan, y `stdout`, `stderr` deben ser dos variables del tipo `StringIO()`, una clase de Python con la que gestionar flujos de texto, en este caso, la salida estándar y la salida de error.

Como se ha mencionado anteriormente, las funciones descritas anteriormente deben ser invocadas en un ámbito de la aplicación desde el que sea posible crear una o varias instancias del `MatlabEngine`, por ello, tal y como estaba estructurado el código de la extensión "MATLAB Interactive Terminal", se recoge todo en la clase `MatlabInterface` en el fichero `matlab_interface.py`. Esta clase es la que contiene en su constructor la llamada a `start_matlab()`, de modo que instanciarla equivale a iniciar un proceso de MATLAB en segundo plano.

`MatlabInterface` es quien importa el paquete `matlab.engine`, y se compone de métodos que a su vez encapsulan las llamadas a los métodos de `MatlabEngine`, mediante los cuales es más sencilla y ordenada la gestión de excepciones y la limpieza del código:

- `__init__()`: Se trata del constructor de la clase, en el que tiene lugar la asignación del resultado de `start_matlab()`, lo que significa que el ámbito del programa en el que invoque a `MatlabInterface()` será el adecuado para albergar el objeto que represente la sesión. Esto se verá con claridad en el apartado dedicado a la API, la entidad que expondrá en un servidor web todas estas funciones.
- `stop()`: Hace uso de la función `quit()` del `MatlabEngine`.
- `run_command()`: Hace uso de la función `eval()` del `MatlabEngine` y devuelve una cadena de texto con la salida estándar del comando o script que se haya invocado.

3.1.2 FastAPI

Esta API REST cuenta con los métodos necesarios para iniciar, parar o reiniciar una sesión de MATLAB, ejecutar comandos, y otras funcionalidades para gestionar las sesiones y formatear el cuerpo de las respuestas. Todos los métodos HTTP son de tipo GET, ya que está fuera de las necesidades de esta API el crear, modificar o eliminar recursos concretos. En su lugar, la labor de esta interfaz se reduce prácticamente a declarar de forma dinámica tantas instancias de `MatlabInterface` como se soliciten, y a ejecutar comandos de MATLAB dentro del proceso que le corresponda a cada instancia.

Por simplicidad y por insuficiente potencia de computación en mi máquina local, se limita el número máximo de sesiones a tres, ya que aumentar la escala del programa no aporta nada al prototipo o prueba de concepto que se desea implementar, y es preferible que la ejecución de la aplicación no sufra inconvenientes que puedan estar fuera del alcance del proyecto.

3.1.2.1 Estructura de archivos

Se detallan a continuación los distintos ficheros que componen el backend, excluyendo el ya conocido `matlab_interface.py`:

- `main.py`: Se trata del fichero principal de la aplicación, en el que se importan, entre otros, los paquetes `FastAPI` y `MatlabInterface`. En este ámbito se gestiona la instancia de la clase `FastAPI()`, se inicializa la variable `sessions` que almacenará las distintas sesiones de MATLAB, y se definen los métodos de la API, que se describirán en el siguiente apartado.
- `session.py`: Fichero que contiene la clase `Session`, que será la que encapsule la relación entre la sesión de frontend y el proceso de MATLAB. La variable `sessions` es un array de elementos de tipo `Session`. Este objeto tiene como atributos:
  - `sid` (Session ID): Sirve para identificar la sesión con un valor numérico.
  - `pid` (Process ID): Al igual que el `sid`, identifica esta vez el proceso de MATLAB, y su valor se obtiene mediante el comando no documentado de MATLAB [23] :

    ```
    feature('getpid')
    ```

    Esta acción se realiza con `run_command()` justo tras inicializar con éxito el proceso de MATLAB, y devuelve el PID de la propia instancia que se está ejecutando dicho comando.
  - `matlab`: Variable de tipo `MatlabInterface` que contiene el descriptor de la instancia de dicha clase. Es la que se usa para llamar a las funciones del motor de MATLAB. Hasta que este atributo no tiene un valor, el objeto Session en cuestión no está asociado a ningún proceso de MATLAB. Cuando se para el motor de MATLAB de una sesión en concreto, tanto este atributo como el atributo pid se vacían.

La clase `Session`, además, cuenta con su constructor, que toma como argumento de entrada el `sid`, y con una función de utilidad que transforma a JSON los atributos `sid` y `pid` de la sesión. Dado que `matlab` no es una variable numérica sino un objeto, serializarlo para enviar su valor como texto no es una tarea trivial, y como no aporta información, no se incluye en la transformación a JSON.

```python
class Session:
    sid = None
    matlab: MatlabInterface
    pid = None

    def __init__(self, sid):
        self.sid = sid

    def toJSON(self):
        return {"sid": self.sid, "pid": self.pid}
```

Figura 2: Definición de la clase Session en el fichero session.py

- `task.py`: Contiene la clase `Task`, que se utiliza para albergar la información sobre un proceso de Windows. Se compone de los atributos `imageName`, `pid`, `cpuTime` y `windowName`.
- `utils.py`: La clase `Utils`, que consta de utilidades que facilitan el trabajo, tales como:
  - `taskList()`: Lanza el siguiente comando de Windows:

    ```
    tasklist /FI "imagename eq matlab.exe" /v /fo csv /nh
    ```

    y devuelve la información de todas las instancias de MATLAB ejecutándose actualmente en el sistema. [24] Posteriormente, se parsean usando la clase `Task` para así poder convertirlas una estructura JSON.
  - `printS(title,sessions)`: Función que imprime por salida estándar el mensaje que se le pase en la variable `title` y el array contenido en la variable `sessions`.
- `figures.m`: Se trata de un script de MATLAB que se ejecuta después de cualquier comando, y sirve para codificar las figuras en Base64 y construir un array de elementos JSON compuestos por un atributo `"id"` y un atributo `"figure"`. El valor de `"id"` es el descriptor de la figura de MATLAB, y el valor de `"figure"` es la cadena de texto representando la imagen en Base64. Gracias a este sencillo script es posible representar gráficamente en el frontend las imágenes que sólo se podrían visualizar en el backend.
- `base64encode.m`: Como su nombre indica, sirve como utilidad para la codificación en Base64, y utiliza una función de un paquete de Java para realizar la operación:

  ```
  org.apache.commons.codec.binary.Base64.encodeBase64Chunked()
  ```

3.1.2.2 Métodos de la API

Como se puede observar en el apartado anterior, el backend realiza algunas funciones complementarias a los endpoints de la API, que son los que están accesibles para el frontend. No son estrictamente necesarias para el funcionamiento esencial del sistema, pero aportan un grado de control y de precisión similar al que tendría la aplicación en su versión de producción. A continuación, se detallan las funciones que componen la interfaz web para MATLAB y posteriormente los endpoints que hacen uso de esas funciones:

- `getSession(sid)`: A partir de un identificador de sesión, esta función busca en el array

`sessions` y devuelve el objeto `Session` correspondiente.

- `getSessions()`: Esta función devuelve el contenido de la variable `sessions` convertido a JSON array, encapsulado en un objeto JSON con un atributo "`sessions`".
- `newSession()`: Este método itera el array `sessions` y selecciona el primer elemento del array que esté disponible, es decir, aquel cuyo atributo `pid` sea `None`, el equivalente a `null` en Pyhton. Una vez completado con éxito, se inserta el objeto `Session` actualizado en el array `sessions`. Devuelve la representación JSON de la sesión recién iniciada, encapsulado en un objeto JSON con un atributo "`result`" y otro atributo "`session`". El valor de "`result`" es un mensaje de información:

  `'New Matlab process with PID='+s.pid`

- Al final de dicho mensaje se concatena con el operador '+' el `pid` de la sesión recién iniciada. En caso de no existir ninguna sesión disponible en la variable `sessions`, se devuelve en el atributo "`result`" un mensaje informando de dicha situación.
- `startMatlab(sid)`: El comportamiento de este método es análogo a `newSession()`, solo que, en lugar de inicializar la primera sesión disponible, se inicializa aquella que indique el argumento `sid`, obteniéndose con la función `getSession(sid)`.
- `taskList()`: Esta función devuelve el resultado de la función de utilidad con el mismo nombre, descrita anteriormente.
- `stopMatlab(sid, restart)`: Es el método que hace uso de la función `stop()` de `MatlabInterface`. Sirve para parar el motor de MATLAB de la sesión indicada mediante el parámetro `sid`. Si el argumento opcional `restart` tiene un valor lógico alto, la respuesta es la de la función `startMatlab(sid)`, en caso contrario, se devuelve un mensaje confimando la parada de la instancia de MATLAB:

  `'Session '+str(sid)+' stopped'`

- `run(sid, commands)`: En primer lugar, usando `getSession(sid)` se obtiene la sesión indicada por el argumento `sid`, y si existe y está disponible, se llama a la función `run_command()` del atributo `matlab`, que alberga la `MatlabInterface` de dicha sesión. El parámetro `commands` contiene uno o varios comandos de MATLAB separados por coma o punto y coma, tal y como se ejecutarían en el intérprete de comandos original o se programarían en un script. Tras la ejecución exitosa de los comandos, se ejecuta el script `figures.m` para obtener las figuras que se hubieran podido representar. Esta función devuelve un objeto JSON que encapsula los atributos "`result`" y "`figures`". El valor del atributo "`result`" es la salida estándar que imprimen los comandos ejecutados, y el valor de "`figures`" es el array de objetos figura con la estructura descrita anteriormente.

3.1.2.3 Endpoints

La estructura de la API a nivel de endpoints es similar a la de las funciones, con la diferencia de que los endpoints están compuesto de un método HTTP, una ruta, y los parámetros, mientras que las funciones no dejan de ser el concepto clásico de función, fragmentos de código que toman valores como entrada y que devuelven valores de salida. Los endpoints, reciben un mensaje HTTP y devuelven otro mensaje HTTP, en el que se encapsula el estado de la petición y el cuerpo del mensaje de respuesta, en caso de haberlo. A continuación, se describen los distintos endpoints que componen la API, una labor para la que está precisamente diseñada la librería Swagger:

- `/sessions`: Llama a la función `getSessions()`.
- `/taskList`: Llama a la función `taskList()`.
- `/newSession`: Llama a la función `newSession()`

- `/startMatlab?sid={sid}`: Recibe como parámetro de query un número entero que identifica a la sesión que se desea arrancar, en este caso representado como `{sid}`. Con ese argumento, llama a la función `startMatlab(sid)`.
- `/stopMatlab?sid={sid}&restart={restart}`: Recibe como parámetros de query un número entero y un parámetro booleano que por defecto tiene un valor lógico bajo, representado como `{restart}`. Con esos argumentos, llama a la función `stopMatlab(sid, restart)`.
- `/run?sid={sid}&commands={commands}`: Recibe el identificador de la sesión y una cadena que contiene los comandos a ejecutar, y llama a la función `run(sid,commands)`.

Como detalle de nomenclatura, cabe mencionar que en el ámbito del backend, se ha hecho referencia al concepto de sesión y se han definido clases y ficheros con ese nombre, mostrando algo de consistencia. Sin embargo, en el frontend, se usa el término "workspace" para referirse a la sesión, por la familiaridad con el concepto clásico de la interfaz original de la aplicación MATLAB.

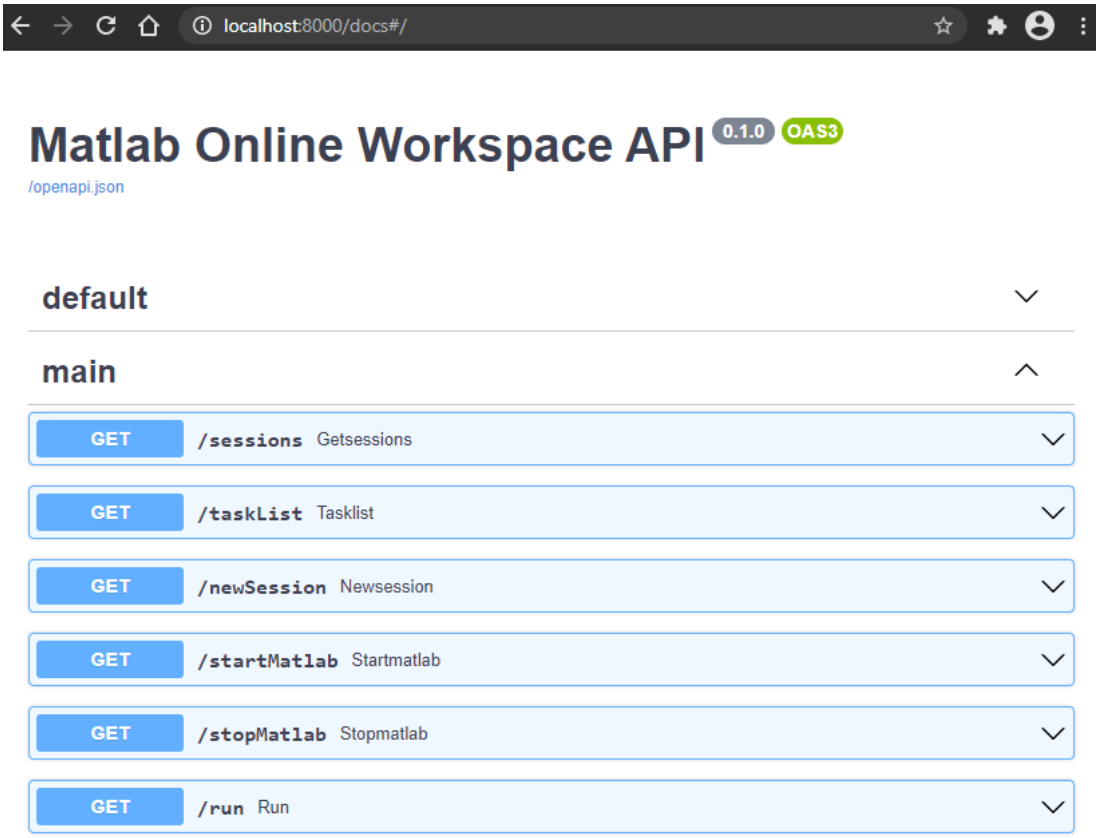

Figura 3: Vista principal de la herramienta Swagger UI, que permite visualizar y realizar peticiones a los métodos de la API de la aplicación.

En la Figura 4 se puede apreciar la utilidad de la herramienta Swagger a la hora de explorar una API. Se muestra el equivalente al comando `curl` para hacer esa petición manualmente, así como la URL que identifica al endpoint.

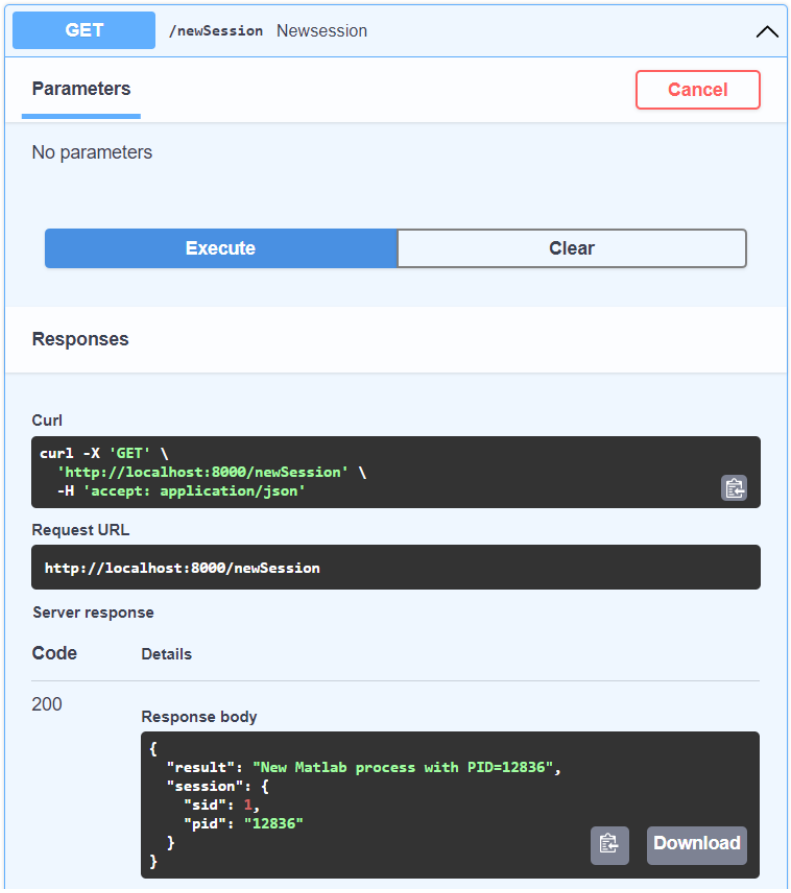

Figura 4: Vista desplegada de un endpoint concreto en la herramienta Swagger UI.

3.2 Frontend

3.2.1 Estructura de archivos

La plataforma Angular, genera automáticamente una estructura de archivos concreta al crear el proyecto con el comando:

```
ng new <nombre de la aplicación>
```

Este entorno de trabajo contiene, además del directorio `/src` con el código fuente de la aplicación y el directorio `/node_modules` con los paquetes de NPM que se están usando, una serie de archivos de configuración [25]:

- `.editorconfig`: Configuración para el editor de código.
- `.gitignore`: Especifica qué archivos se van a ignorar en caso de usar tecnología de control de versiones.
- `README.md`: Archivo introductorio de documentación para la aplicación.
- `angular.json`: Configuración para otros proyectos de angular en el mismo entorno de trabajo.
- `package.json`: Configuración de los paquetes de NPM para todos los proyectos del entorno de trabajo.

- `package-lock.json`: Configuración de las versiones de los paquetes de NPM instalados en el directorio `/node_modules`.
- `tsconfig.json`: Configuración para el lenguaje TypeScript en los proyectos del entorno de trabajo.
- `tsconfig.app.json`: Configuración para el lenguaje TypeScript para la aplicación específica.
- `tsconfig.spec.json`: Configuración para los tests de la aplicación.
- `.browserslistrc`: Configuración para asociar versiones de navegadores web con versiones de elementos del frontend, para asegurar compatibilidad.
- `karma.conf.js`: Configuración relacionada con los tests.

Dentro del directorio `/src` se encuentra el código fuente, los datos y los ficheros necesarios para la aplicación, tales como iconos, fuentes, imágenes, etc. En primer lugar, se detallan los ficheros generados por Angular a la hora de generar el proyecto base, y en el siguiente apartado se describirán en profundidad los ficheros con el código propio de la aplicación, tras haber concluido el desarrollo. Esto es relevante para poder trazar la delgada línea que permite entender al mismo tiempo el valor que tiene lo que aporta esta plataforma de desarrollo y el valor de la solución software desarrollada en este proyecto.

- `/app`: Contiene los componentes mediante los cuales se define la lógica de la aplicación. Por defecto, el código mínimo para tener una aplicación funcional se genera en el componente raíz `AppComponent`, distribuido en los ficheros:
  - `app.component.ts`: Define la lógica del componente `AppComponent`. El resto de componentes que constituyen las pestañas de la aplicación son elementos anidados a éste.
  - `app.component.html`: Define la plantilla HTML asociada al componente `AppComponent`.
  - `app.component.scss`: Define la hoja de estilo.
  - `app.component.spec.ts`: Define un test unitario para el componente raíz.
  - `app.module.ts`: Se trata del módulo raíz que engloba al resto de módulos que componen la aplicación.
- `/assets`: Contiene imágenes y otros archivos relevantes que no son de código ni configuración.
- `/environments`: Contiene información acerca de los distintos entornos en los que pretende configurar el despliegue de la aplicación. Aquí es donde se definen variables de entorno tales como la dirección de la API, algunas constantes u otra información relevante al entorno concreto en el que se ejecuta el frontend.
- `favicon.ico`: El icono que aparecerá en la miniatura de la pestaña del navegador.
- `index.html`: La página HTML que se entrega a un navegador cuando visita la dirección en la que está desplegada la aplicación. Por defecto, ya está configurada para que renderice el componente `AppComponent`.
- `main.ts`: Este es el punto de entrada a la aplicación, donde se compila y empaqueta el módulo `AppModule` para ejecutarlo en el navegador.
- `polyfills.ts`: Contiene scripts de polyfill, es decir, adaptación de funcionalidades para navegadores demasiado antiguos para soportarlas.
- `styles.scss`: Contiene los archivos que proveen estilo a la aplicación.
- `test.ts`: Este es el punto de entrada a los tests de la aplicación.

La apariencia de la aplicación es el primer paso para visualizar los componentes de Angular que forman parte de la vista, y descomponer la plantilla HTML del componente `AppComponent`, que es sobre el cual se sustenta el frontend. Este frontend consta de una sola pantalla o vista, en la que se distinguen los cuatro elementos de la interfaz: el menú de navegación, el área de notificaciones, la consola, y la zona en la que se visualizan figuras:

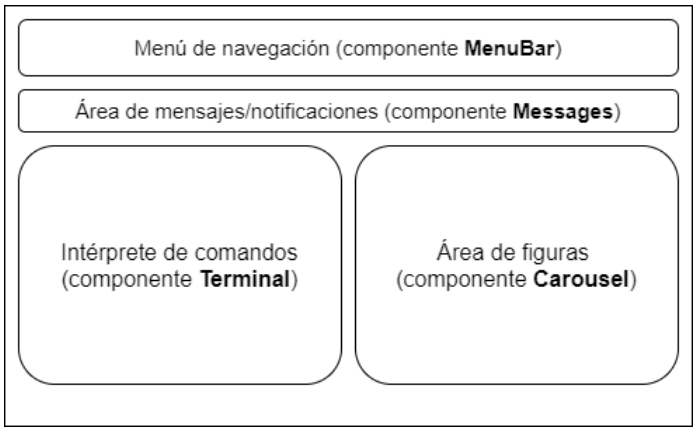

Figura 7: Esquema de la disposición de los componentes de la interfaz

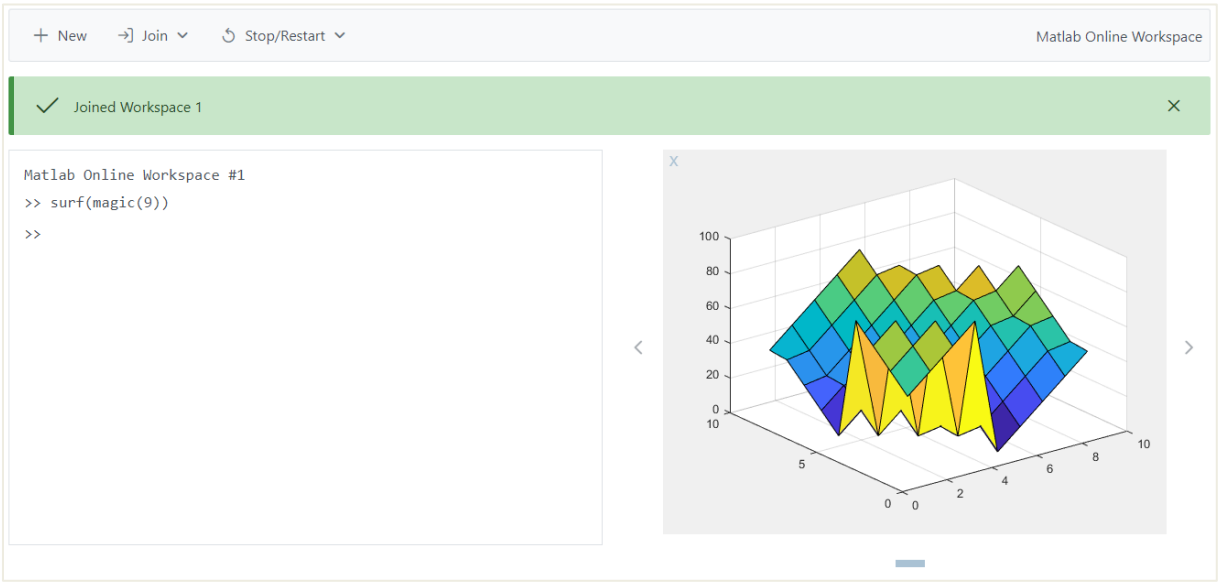

Figura 6: Interfaz gráfica de la aplicación

En este apartado se describe el contenido y las relaciones entre los ficheros del directorio `/app`, que son los que al fin y al cabo constituyen la lógica "palpable" de la aplicación, y más tarde se describirán a nivel gráfico y a nivel lógico los componentes que constituyen `AppComponent`:

- `app.component.html`: Plantilla HTML que describe qué elementos se renderizarán en la pantalla, dependiendo de los atributos especiales de Angular que gobiernan el comportamiento de algunas de las características de los componentes, definidos en `app.component.ts`. Cada uno de los cuatro elementos que componen la interfaz es un componente importado de PrimeNG: `MenuBar`, `Messages`, `Terminal` y `Carousel`. A lo largo de la descripción del frontend, la renderización y los atributos de estos componentes son los que invervienen en el funcionamiento de la aplicación, por lo que serán referidos y analizados de forma transversal.

- `app.component.ts`: Se trata del fichero principal del componente raíz `AppComponent`. En él se definen, declaran y asignan las variables que representarán los atributos que determinarán el comportamiento de los cuatro componentes. Todos los eventos y acciones que permite realizar la interfaz tienen como punto de partida este fichero. Desde aquí se invoca al servicio `MatlabService`, y se actualiza el valor de los atributos concretos de los componentes que dan funcionalidad a la interfaz. En lugar de describir las funciones definidas en este fichero, es preciso conocer primero algunos detalles sobre el uso concreto de los componentes de la interfaz, y tener una visión general del flujo de la información y de los eventos que tienen lugar con la interacción del usuario. Por ello, se irá describiendo la lógica del programa a medida que sea necesario entender la acción que se está describiendo.
- `app.component.scss`: Hoja de estilo para los elementos de `app.component.html`. Desde aquí se especifican los atributos de estilo para ajustar la apariencia, tales como la fuente utilizada en el componente `Terminal`, o las dimensiones y márgenes de los elementos.
- `app.module.ts`: Este fichero contiene todos los módulos necesarios para la aplicación: `BrowserModule`, `HttpClientModule`, `MenubarModule`, `MessagesModule`, `TerminalModule`, y `CarouselModule`.
- `models.ts`: Fichero que alberga las clases que representan las entidades protagonistas del frontend:
  - `MatlabSession`: Clase análoga a la clase `Session` del backend, contiene dos atributos de tipo numérico: `pid`, y `sid`
  - `Figure`: Clase análoga al atributo `"figure"` que contiene el elemento JSON generado por el script `figures.m` del backend. Contiene un atributo `id` de tipo numérico y un atributo `base64` de tipo cadena, o también conocido como `string`.
  - `MatlabResponse`: Esta clase, gracias a la flexibilidad de TypeScript, modela de forma generalizada todos los objetos JSON de respuesta que devuelve la API. Es decir, el parseo de la respuesta de un endpoint mediante la clase `MatlabResponse` dará como resultado un objeto que únicamente tiene los atributos que incluya ese objeto JSON de respuesta. Por ejemplo, si el JSON sólo contiene `"result"`, al parsearlo con esta clase, se obtendrá un objeto que sólo tendrá un atributo `result`, y los demás atributos definidos para la clase simplemente se ignoran. Sus atributos son:
    - `result`: Cadena de texto que devuelve la API como mensaje informativo.
    - `figures`: Un array de objetos de tipo `Figure`.
    - `session`: Un objeto de tipo `MatlabSession`.
    - `sessions`: Un array de objetos de tipo `MatlabSession`.
- `matlab.service.ts`: Se trata del fichero que encapsula la entidad encargada de dar uso al `HttpClient` y realizar las peticiones a API: `MatlabService`. Aquí se define la URL del servidor que despliega Uvicorn en el backend, así como las funciones análogas a las descritas en el apartado de la API, tales como:
  - `getSessions()`: Realiza la petición GET al endpoint `/sessions`.
  - `newWorkspace()`: Realiza la petición al endpoint `/newSession`
  - `runCommand(sid, commands)`: Recibe como parámetro el sid y la cadena con los comandos, y llama al endpoint `/run`.
  - `stopMatlab(sid, restart)`: Llama al endpoint `/stopMatlab` con argumentos de entrada análogos.

- `getFigures(sid)`: Llama a la función `runCommand(sid, 'figures')`, para así ejecutar el script de MATLAB del backend que devuelve las figuras abiertas en el workspace de la sesión indicada por el argumento `sid`.

Todas las funciones de `MatlabService` devuelven un `Observable` que encapsula un `MatlabResponse`.

3.2.2 Descripción y funcionamiento de los componentes

A continuación, se detallan las acciones o eventos que desencadena la interacción del usuario con los elementos de la interfaz, así como el flujo completo de información desde que se dispara un evento hasta que se procesa y se renderiza el resultado. Como puede apreciarse, la plantilla HTML que constituye la interfaz gráfica es sumamente sencilla, y se resume a los cuatro componentes con sus propiedades.

```
<p-menubar [model]="menuItems"> Matlab Online Workspace </p-menubar>
<p-messages [(value)]="msgs"></p-messages>
<div class="grid">
  <div class="col-6">
    <p-terminal
      *ngIf="this.displayTerminal"
      welcomeMessage="Matlab Online Workspace #{{ this.session.sid }}"
      prompt="{{ this.prompt }}"
      autofocus
    >
    </p-terminal>
  </div>
  <div class="col-6">
    <p-carousel
      *ngIf="this.displayFigures"
      [value]="matlabResponse.figures"
      [numVisible]="1"
      [numScroll]="1"
    >
      <ng-template let-figure pTemplate="item">
        <div class="grid">
          <div class="col-1">
            <button
              pButton
              class='p-button-text'
              (click)="closeFigure(figure.id)">X</button>
          </div>
          <div class="col-10">
            <img src="{{ figure.base64 }}" />
          </div>
        </div>
      </ng-template>
    </p-carousel>
  </div>
</div>
```

Figura 8: Plantilla HTML de la aplicación.

3.2.2.1 Menubar

El menú superior de navegación recibe un solo parámetro `model`, mediante el cual se define la estructura de los botones y acciones que desencadenan los principales eventos relacionados con la creación de sesiones y la apertura de sesiones existentes. [26] La variable `menuItems` es un array de objetos de tipo `MenuItem`, que a su vez tiene los atributos:

- `label`: Cadena de texto que da nombre al `MenuItem`
- `icon`: Icono que se visualiza dentro del botón
- `items`: Array de objetos `MenuItem` anidados al `MenuItem` en cuestión.
- `command`: Función o estructura funcional que se ejecuta cuando se pulsa sobre el `MenuItem`.

La variable `menuItems` se asigna a la propiedad `model` del `Menubar`, y da como resultado un menú con tres nodos, dos de ellos con un nivel de jerarquía anidado:

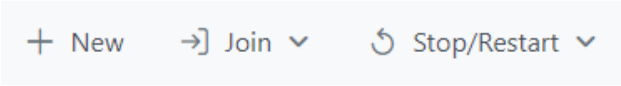

Figura 10: Menú superior de navegación

```
this.menuItems = [
  {
    label: 'New',
    icon: 'pi pi-fw pi-plus',
    command: () => this.newWorkspace(),
  },
  {
    label: 'Join',
    icon: 'pi pi-fw pi-sign-in',
    items: this.menuItemSessions,
  },

  {
    label: 'Stop/Restart',
    icon: 'pi pi-fw pi-replay',
    items: [
      {
        label: 'Delete',
        icon: 'pi pi-fw pi-trash',
        command: () => this.stopSession(this.session),
      },
      {
        label: 'Restart',
        icon: 'pi pi-fw pi-refresh',
        command: () => this.restartSession(this.session),
      },
    ],
  },
];
```

Figura 10: Estructura anidada que da funcionalidad al componente Menubar.

Las funcionalidades del menú de navegación son:

- New: Llama a la función `newWorkspace()` de `app.component.ts`, que a su vez llama a la función con el mismo nombre de `matlab.service.ts`. Con esta acción se crea un nuevo workspace con una nueva sesión de MATLAB, y si la petición se resuelve con éxito, se muestra el componente `Terminal` con el intérprete de comandos del workspace recién creado. Tras esta acción, queda asignada o actualizada la variable `session`, que contiene la información del workspace actual.
- Join: Esta opción contiene en su atributo `items` otro array de `MenuItems`, cada uno representando un workspace existente al que el usuario puede unirse. Este array se define en la variable `menuItemSessions`, que se actualiza tras crear una nueva sesión o tras cerrarla. Cada elemento de `menuItemSessions` es un `MenuItem` cuyo atributo `command` invoca a la función `joinWorkspace()`. Esta función, obtiene las sesiones actualmente activas, actualiza la variable `session`, y llama a la función `getFigures()` del `MatlabService`, para actualizar en el frontend las figuras abiertas del workspace en cuestión.
- Stop/Restart: Esta opción despliega otros dos `MenuItems` que sirven para parar o reiniciar el workspace actual, uno con el `label` Close y como `command` la función `closeWorkspace()`, y otro con `label` Restart y como `command` la función `restartWorkspace()`. Ambas funciones se comportan de forma similar, solo que `restartWorkspace()` hace uso del argumento `restart` del endpoint `/stopMatlab`, asignándole un valor lógico alto para así reiniciar la sesión.

La visibilidad de los distintos elementos del menú de navegación está sujeta a condiciones, es decir, si no hay ninguna sesión activa, los botones Join y Stop/Restart no aparecen, ya que carecen de sentido. De forma similar, si existen sesiones activas pero el usuario no se ha unido a ninguna, el botón Stop/Restart permanece oculto. La estructura menuItems debe actualizarse y volver a asignarse a la propiedad model del Menubar cada vez que se cree o elimine un workspace.

3.2.2.2 Messages

Este componente sirve para notificar al usuario del estado o resultado de las acciones que se están desencadenando. [27] La estructura es simple, cada mensaje se encapsula en un elemento con dos atributos:

- `severity`: Toma los valores `'success'`, `'info'`, `'warn'`, `'error'`, que modifican el color y el icono de la notificación. Respectivamente, los colores son verde, azul, amarillo y rojo.
- `detail`: Cadena de texto que constituye el mensaje en sí.

El funcionamiento es tan simple como la estructura, basta con asignar a la propiedad `value` un array de objetos `Message`, llamado `msgs`, y cada vez que se desee mostrar un nuevo mensaje, se inserta al final del array con el método `push()` de JavaScript. En esta aplicación solo se va a mostrar un mensaje al mismo tiempo, por lo que antes de insertar un nuevo elemento, se vacía.

```
this.msgs.push({
  severity: 'info',
  detail: 'No Matlab workspaces running!',
});
```

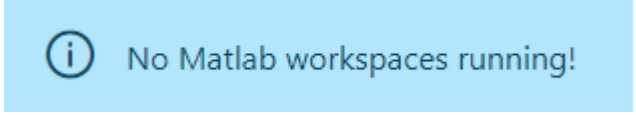

Figura 11: Inserción de un elemento en el array msgs (arriba), y resultado renderizado en la interfaz (abajo).

3.2.2.3 Terminal

Aquí es donde tiene lugar el envío de comandos y visualización del resultado, con una apariencia y dinámica similar al intérprete original de comandos de MATLAB, incluyendo los caracteres “>>” para el prompt. [28] Las propiedades de `Terminal` que van a usarse son :

- `welcomeMessage`: Cadena de texto que se imprime dentro del área de la consola la primera vez que se inicializa un workspace, a modo de mensaje de bienvenida. El mensaje incluye el valor del atributo `sid` de la variable `session`, para así identificar en qué workspace está el usuario según el mensaje. El valor de la variable se escapa envolviendo la variable con doble llave: `{{this.session.sid}}`.
- `prompt`: Cadena de texto que representa el símbolo propio del intérprete de comandos. Se le asigna la variable `prompt` de `app.component.ts`.
- `autofocus`: Valor booleano que configura el componente para que siempre tenga el foco del cursor, es decir, al pulsar una tecla, ese texto se interprete como input en la terminal. Es equivalente a hacer clic con el ratón en el área de la terminal.
- *`ngIf`: Se trata de una propiedad que puede aplicarse a cualquier elemento HTML de angular, y sirve para determinar si se va a renderizar o no. Si esta propiedad toma un valor lógico alto, se muestra. En este caso se le asigna la variable `displayTerminal`, cuyo valor se actualiza en situaciones como la inicialización del componente raíz `AppComponent` o cuando el usuario cambia a un workspace existente o nuevo. En general, si la variable `session` está vacía, el componente `Terminal` no se renderiza, ya que no tiene sentido permitir la entrada de comandos si no hay asociada una sesión de MATLAB que los pueda procesar.

```
constructor(
  private terminalService: TerminalService,
  private matlabService: MatlabService,
) {
  this.terminalService.commandHandler.subscribe((command) => {
    this.prompt = "<<";
    this.matlabService.runCommand(this.session.sid, command).subscribe(
      (result) => {
        this.matlabResponse = new MatlabResponse();
        this.matlabResponse = result as MatlabResponse;
        this.terminalService.sendResponse(this.matlabResponse.result);
        if(command === 'clc'){
          this.clearConsole();
        }
        this.plotFigures();
        this.prompt = ">>";
      },
      (error) => {
        console.log(error);
        this.msgs = [];
        this.msgs.push({
          severity: 'error',
          summary: '',
          detail: error,
        });
      }
    );
  });
}
```

Figura 12: Constructor de AppComponent y definición del procesado de un comando.

Este componente tiene una forma concreta de implementarse, ya que debe recibir el texto introducido por el usuario, procesar el comando e imprimir el resultado a partir de la siguiente línea de la consola. Para ello se hace uso del TerminalService, que está diseñado para gestionar el procesado de comandos y se define en el constructor de `AppComponent`. Con el atributo `commandHandler` de tipo `Observable` y usando el método `subscribe()` de ese `Observable`, es posible enviar el comando a la API, parsear la `MatlabResponse`, y enviar la respuesta a la terminal con un método del `TerminalService`. [primeng terminal] Analizando en la Figura 12 el código correspondiente al constructor del componente raíz, se muestra paso a paso cómo funciona el procesado de comandos.

Cada vez que se escribe un comando y se pulsa Intro, se dispara el `commandHandler` y cuando termina de procesar la cadena de texto que el usuario acaba de introducir, entra en el método `subscribe()`. Lo primero que ocurre es que se imprime el prompt pero con los caracteres opuestos, de forma que intuitivamente se entienda que no se debe escribir nada nuevo hasta haberse procesado el actual comando. A continuación, se hace la petición al backend y se procesa la respuesta en el método `subscribe()` del `commandHandler`, es decir, hasta que la petición no se ha completado con éxito, el flujo del programa no llega a la estructura funcional cuyo argumento de entrada es la variable `result`. Si la petición se resuelve con algún error, se ejecutará el fragmento correspondiente a la variable `error`, que contendrá el cuerpo de respuesta. Dentro del callback de éxito, se inicializa vacía la variable `matlabResponse` y se usa para parsear el contenido de la variable `result`. Tras esto, se llama a la función `sendResponse()` del TerminalService, que toma como argumento de entrada la cadena de texto que representa la salida estándar del comando recién ejecutado en el motor de MATLAB del backend. Posteriormente, debido a que el intérprete de comandos del backend no controla el componente `Terminal` del frontend, es necesario implementar manualmente el vaciado de la consola al introducir el comando `clc` de MATLAB. Esto se realiza en la función `clearConsole()`, atendiendo a la nomenclatura de estilos del componente `Terminal`. La clase HTML a la que pertenece el elemento dentro del cual se renderiza el contenido es `p-terminal-content`. La interfaz de JavaScript `Document`, que con una estructura jerárquica representa todos los elementos que hay renderizados en una determinada página, tiene el método `getElementByClassName()`, que toma como argumento de entrada una cadena de texto representando el nombre de la clase. [29] De esta forma se accede al atributo `innerHTML` del `p-terminal-content`, que consiste en el texto que tiene dentro el elemento con esa clase. Finalmente, se invoca a la función `plotFigures()`, que será descrita en detalle en el siguiente apartado, pero simplemente comprueba que el atributo `figures` de la variable `MatlabResponse` no esté vacío, para así mostrar el componente `Carousel` con las figuras. Para que el usuario sepa que se ha terminado de procesar un determinado comando, se vuelve a imprimir el prompt original.

3.2.2.4 Carousel

Este es el componente que se encarga de albergar las figuras que se abrirían en ventanas independientes en la aplicación MATLAB. Está diseñado para mostrar cualquier tipo de elemento, pero en este caso se usa como visor de imágenes [30]. Las propiedades que intervienen en la implementación del componente son:

- *`ngIf`: Propiedad que determina la visibilidad del componente, en este caso, se usa la variable `displayFigures`, que se pone a nivel alto cada vez que el atributo figures de la variable `matlabResponse` tenga una longitud mayor que cero.
- `value`: Representa la colección de objetos que se van a mostrar en el `Carousel`. Se le asigna directamente el atributo `figures` de la variable `matlabResponse`, que, convenientemente actualizada, debe contener en todo momento las figuras abiertas en el workspace en cuestión.
- `numVisible`: Esta propiedad determina el número de elementos que son visibles al mismo tiempo en el `Carousel`. Por mantener la simplicidad de la interfaz y por aprovechar el espacio reservado para el componente, solo se muestra una figura a la vez.

- `numScroll`: Con esta propiedad se define el número de figuras que se desplazarán al pulsar sobre los botones de desplazamiento del `Carousel`. De forma análoga a la propiedad anterior, se mantiene este valor a uno, para navegar una a una por las figuras en visualización.

Como se ha descrito anteriormente, lo que el backend envía en la respuesta de la llamada a `/runCommand` es la codificación en Base64 de la imagen, encapsulada en un objeto `Figure`, con atributos `id` y `base64`. Para renderizar una imagen en HTML, basta con crear un elemento `<img>` y asignar a su propiedad `src` la ruta de la imagen. La ruta de la imagen puede definirse de varias maneras, pero para esta aplicación se ha optado por una URL de datos, que se construye de la siguiente forma [31]:

```
'{base64}'
```

Donde `{base64}` representa la codificación en Base64 de la imagen.

La variable `displayFigures`, que controla la visibilidad de las figuras, se pone a nivel alto únicamente dentro de la función `plotFigures()`, y con la condición de que, tras ejecutarse un comando o cambiar de workspace, el array de figuras de la variable `matlabResponse` tenga al menos un elemento.

Con respecto al contenido del elemento `Carousel` en la plantilla HTML, cabe mencionar algunos detalles, entre ellos el uso del elemento de Angular `ng-template`, con el que se define un elemento HTML que no se renderiza por defecto, sino sujeto a alguna condición. Uno de los principales usos de `ng-template` es albergar contenido que será manipulado por directivas de control como `*ngIf`. [32] En este caso, se utiliza para renderizar cada elemento del array `matlabResponse.figures`, que, en caso de estar vacío, no habrá ningun elemento `ng-template` que se visualice. Por defecto, la propiedad que permite asignar un nombre al iterador de la colección es la palabra clave `let`, seguida de un guión y el nombre con el que se desee identificar a cada elemento del array. De esta forma, simplemente con la propiedad `let-figure`, se tiene un objeto `Figure` del que se puede extraer los atributos `id` y `base64`. Dentro del `ng-template`, hay un elemento `<img>` y un `<button>`, que son los que simulan la ventana de figura de MATLAB. A la etiqueta `<img>` se le asigna la ruta de datos de la imagen, y a la propiedad `click` del `<button>` se le asigna la función `closeFigure()`, que toma como argumento de entrada el `id` de una figura, que se genera en el backend a partir de su descriptor en el entorno gráfico del motor de MATLAB. Esta función simplemente llama al endpoint `/run`, y ejecuta en el backend el comando `close` seguido del identificador de la figura. Como ocurre tras ejecutar cualquier otro comando, la respuesta `matlabResponse` contiene el array de figuras actualizado, y se actualizará en el frontend, resultando así en el cierre de la figura.

4 DESPLIEGUE Y ACCESO AL SERVICIO

El despliegue de este sistema, gracias a la elección de las herramientas y tecnologías, resulta ser una tarea sencilla, al menos en lo que respecta al despliegue de frontend y backend en la misma máquina. En un entorno de producción o preproducción, sería indispensable dedicar un profundo análisis y un diseño minucioso, entre otras cosas, a la gestión de la visibilidad de red entre los distintos servicios y aplicaciones. También es importante diseñar un proceso aproximadamente automatizado para instalar las herramientas necesarias para ejecutar la aplicación, como Node, Angular, FastAPI, Pyhton, Uvicorn, y la menos trivial, MATLAB. Cada elemento de la aplicación debe estar deplegado en una subred con un alias para su tupla dirección/puerto, las tablas de encaminamiento y configuracion de las interfaces debe ser coherente con la situación de cada máquina, etc.

Para el caso de uso que concierne a este prototipo, basta con seguir el estándar propuesto por los frameworks que engloban el proceso de despliegue de cada componente de la aplicación: Node para el frontend y Uvicorn para el backend.

4.1 Despliegue

En el caso del backend, para desplegarlo basta con utilizar el comando de Uvicorn que inicia el servidor local en la URL http://localhost:8000.

```
uvicorn main:app
```

El comando toma como argumento el nombre del fichero principal de la API, main, seguido de dos puntos, y el nombre de la variable que alberga la instancia de la clase FastAPI(), en este caso, app. En el caso del frontend, el despliegue con el comando

```
ng serve
```

inicia por defecto un servidor local, que está ejecutando la aplicación a la que se accede mediante la URL http://localhost:4200. También es posible realizar la acción de despliegue con el comando:

```
npm start
```

Éste dependerá de la configuración existente en el fichero package.json, en el que, entre otras cosas, se definen los comandos que se deben ejecutar cuando se lance la orden start de NPM. Finalmente, si está bien configurado, el comando que se ejecuta es ng serve [33].

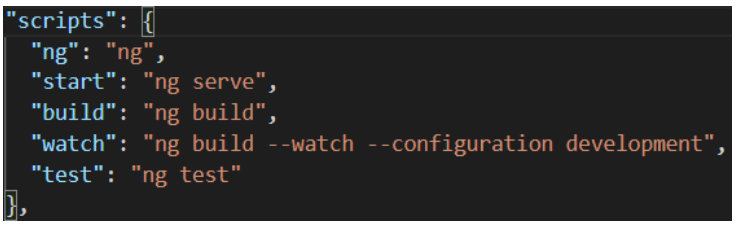

Figura 13: Fragmento del fichero package.json.

```
D:\Dropbox\tfg\front>npm start

> front@0.0.0 start
> ng serve

⠋ Generating browser application bundles (phase: setup)...[HPM] Proxy created: undefined  ->  http://localhost:8000
[HPM] Subscribed to http-proxy events:  [ 'error', 'close' ]
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |      Size
vendor.js             | vendor        |   3.03 MB
styles.css, styles.js | styles        | 814.37 kB
polyfills.js          | polyfills     | 508.73 kB
main.js               | main          |  29.08 kB
runtime.js            | runtime       |   6.57 kB

                      | Initial Total |   4.35 MB

Build at: 2021-08-30T15:39:49.748Z - Hash: 330515edcabf8f4f7da5 - Time: 8857ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


✓ Compiled successfully.
```

Figura 14: Resultado de ejecución del comando de arranque del frontend

```
D:\Dropbox\tfg\fastapi>uvicorn main:app
INFO:     Started server process [16100]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Figura 15: Resultado de la ejecución del comando de arranque del backend

4.2 Acceso al servicio

Si se accede a la URL localhost:4200 en el navegador, lo primero que el usuario se encuentra es el menú de navegación y la notificación correspondiente para informar de que no existen sesiones de MATLAB activas. Al pulsar el botón New, se inicializa un nuevo workspace, y cuando finaliza el proceso se hace visible el intérprete de comandos:

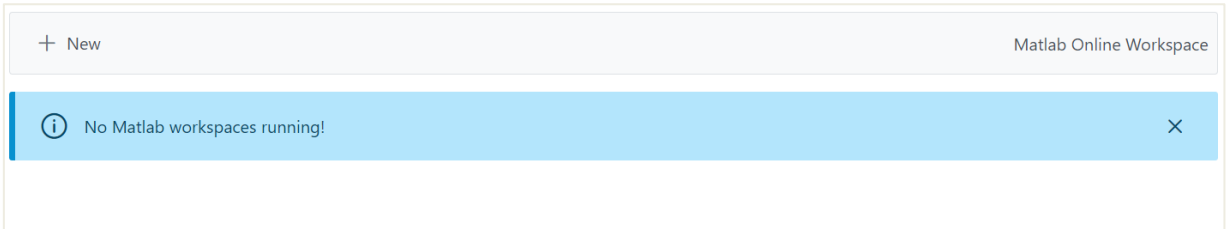

Figura 16: Pantalla principal de la interfaz

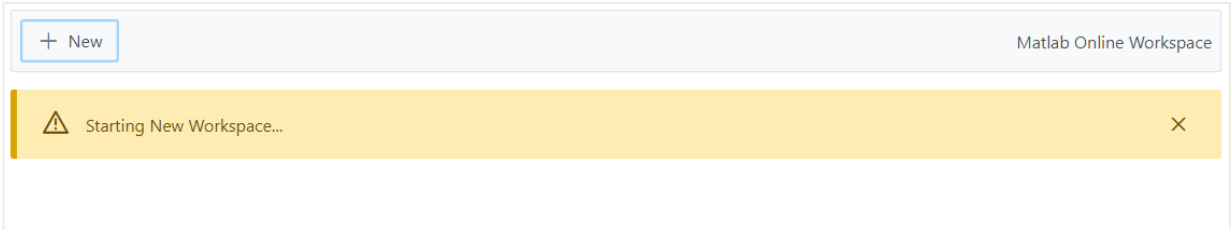

Figura 17: Notificación correspondiente a la inicialización de un nuevo workspace

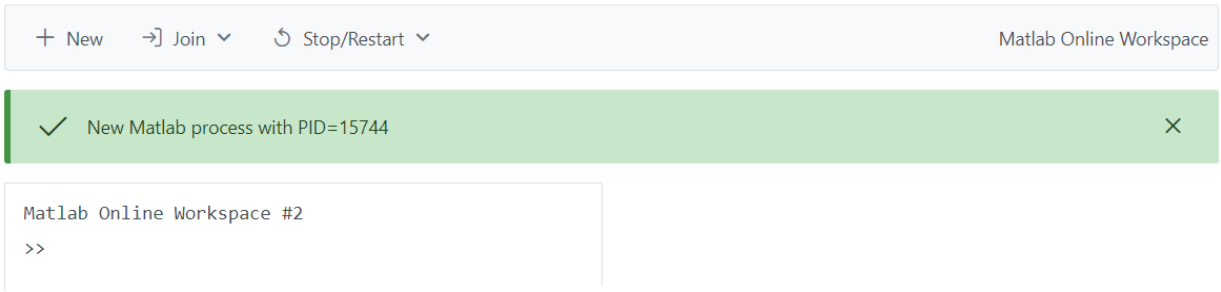

Figura 18: Resultado tras unirse a un nuevo workspace

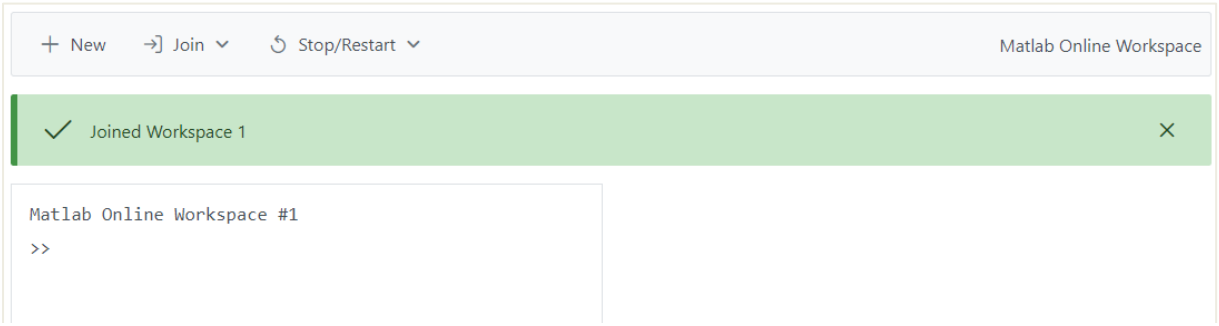

Figura 19: Resultado tras unirse a un workspace

Si habiendo creado un workspace se recarga la pagina, desaparece el intérprete de comandos, pero está visible el elemento de navegación Join, que permite unirse a un workspace activo. Una vez situado en el intérprete de comandos, es posible ejecutar funciones propias de MATLAB y visualizar la respuesta o la figura correspondiente. Por ejemplo, como se muestra en la Figura 20, viendo el resultado del comando pwd se determina que el directorio raíz del motor de MATLAB es el directorio en el que se encuentra el código fuente del backend.

```
Matlab Online Workspace #2
>> pwd
 'D:\Dropbox\tfg\fastapi'
```

Figura 20: Resultado del comando '**p**rint **w**orking **d**irectory'

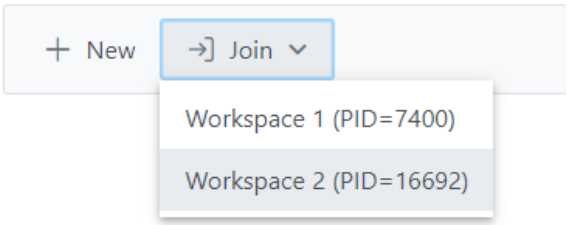

Figura 22: Opciones del menú de navegación para unirse a sesiones abiertas

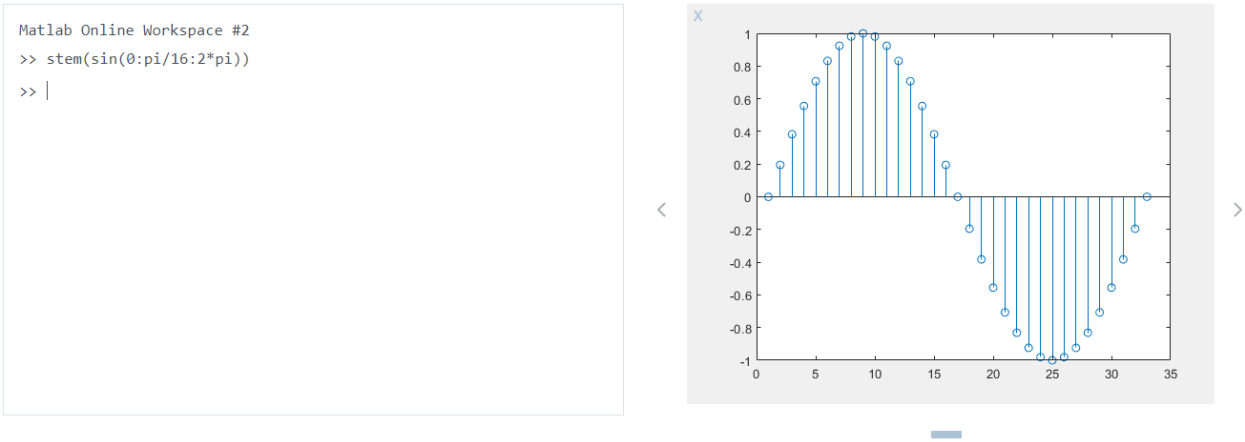

Figura 21: Ejemplo de comando que desencadena apertura de figura

5 CONCLUSIONES

El desarrollo de este proyecto partió del propósito de diseñar un sistema que permitiera a uno o varios usuarios disfrutar de las características principales del intérprete de comandos de MATLAB, bajo una sola licencia oficial registrada. Esto puede significar, en el entorno académico, una mejora con respecto a la accesibilidad de cualquier alumno o interesado a este software propietario, y otra contribución más para el ámbito del software libre y de las herramientas que ayudan a que la ciencia pertenezca a todo el mundo.

5.1 Carencias

Además de algunos detalles con respecto al despliegue de la aplicación web como sistema distribuido, existen algunos aspectos relevantes de las funcionalidades de la aplicación MATLAB y su intérprete de comandos que han quedado fuera del alcance del proyecto, bien por su complejidad de implementación o por ser inviable debido a las restricciones que conlleva la comunicación HTTP entre dos aplicaciones. Este proyecto pretende ser una prueba de concepto para desencadenar posibles líneas futuras de investigación.

La funcionalidad que más brilla por su ausencia en la interfaz es el editor de scripts, algo bastante cómodo teniendo en cuenta la posibilidad de depurar el código y usar la aplicación como un IDE. Sin embargo, agudizando un poco el ingenio, el usuario puede darse cuenta de que no hay nada que le impida generar un archivo de texto con la extensión de MATLAB, cuyo contenido sea el script que se desea ejecutar:

```
Matlab Online Workspace #1
>> ls
 . README.md main.py task.py .. __pycache__ matlab-fastapi.code-workspace utils.py
 .git base64encode.m matlab_interface.py .gitignore figures.m session.py
>> file = fopen('ejemploScript.m','w');
>> ls
 . README.md figures.m session.py .. __pycache__ main.py task.py .git
 base64encode.m matlab-fastapi.code-workspace utils.py .gitignore ejemploScript.m
 matlab_interface.py
>> fprintf(file, "close all; clc, clear; im=ones(256); im(64:172,64:172)=0.5;
imshow(im)");
>> fclose(file);
>> type ejemploScript.m
 close all; clc, clear; im=ones(256); im(64:172,64:172)=0.5; imshow(im)
>>
```

Figura 23: Solución alternativa para guardar en un fichero de texto una secuencia de comandos, a modo de script.

En la Figura 22 puede observarse cómo haciendo uso de las funciones de MATLAB fopen, fprintf y fclose, se crea un archivo que puede ser ejecutado como script. El comando type se usa para imprimir por salida estándar el contenido de un fichero. La Figura 23 muestra la ejecución de dicho script, que no ha sido excesivamente cómodo de redactar. El resultado es aproximadamente el mismo que escribir directamente los comandos en una línea del intérprete separados por punto y coma, es decir, el argumento que se le pasa a la función fprintf.

Figura 24: Resultado de la ejecución de un script generado desde la línea de comandos

Otra de las características del intérprete de comandos original es, como en cualquier consola o terminal, la posibilidad para navegar fácilmente entre los últimos comandos que se han ejecutado. Dado que el componente Terminal de Angular no es un intérprete de comandos real, ese tipo de detalles hay que implementarlos manualmente, algo parecido a lo que ocurrió con el comando clc.

5.2 Valoración del autor

Este proyecto ha supuesto una gran inversión de tiempo e ingenio, ya que su propósito real se ha mantenido algo difuso durante una buena parte de la etapa de investigación y análisis de artículos y publicaciones. Es más, la mayor parte de la investigación estuvo centrada en las tecnologías de audio fingerprint y en el desarrollo y optimización del programa de MATLAB que simulaba el funcionamiento del software comercial Shazam. Gracias a la iniciativa de *salir de MATLAB*, el proyecto acabó dando el giro oportuno. En un principio, el hecho de haber encontrado un propósito tan distinto al original supuso una sensación de incertidumbre, pero dio lugar al escenario perfecto para poner a prueba la perspicacia y visión transversal en el ámbito del software.

Las herramientas utilizadas en el desarrollo de la aplicación son las mismas que se utilizan en cualquier entorno relacionado con el desarrollo web, y adquirir experiencia con ellas siempre va a ser provechoso. De hecho, siendo herramientas relativamente nuevas, si han adquirido tal popularidad y soporte de la comunidad es precisamente por la simplicidad con la que están diseñadas para usarse. Usando con sabiduría los motores de búsqueda, con nociones básicas de desarrollo web y siendo capaz de seguir manuales de instrucciones, este proyecto ha sido posible sin necesidad de una gran experiencia en ninguno de los ámbitos que intervienen.

Teniendo en cuenta el aprecio hacia MATLAB tras tantos años estudiando y disfrutando de sus peculiaridades, el hecho de haber desarrollado una interfaz web sencilla y con tecnología punta de código libre parece una legítima forma de finalizar los estudios y contribuir a la comunidad.

6 ANEXO

6.1 Código fuente

El control de versiones se ha realizado alojando el código en GitHub. En ambos casos, se pueden visualizar en orden cronológico todos los cambios que ha sufrido el código.

6.1.1 Frontend

Página principal del repositorio: https://github.com/nilodude/mow-ui

Historial de cambios: https://github.com/nilodude/mow-ui/commits/master

Se adjuntan al documento los ficheros principales, los del directorio `/app`:

`app.component.html`:

```
<p-menubar [model]="menuItems"> Matlab Online Workspace </p-menubar>
<p-messages [(value)]="msgs"></p-messages>
<div class="grid">
  <div class="col-6">
    <p-terminal
      *ngIf="this.displayTerminal"
      welcomeMessage="Matlab Online Workspace #{{ this.session.sid }}"
      prompt="{{ this.prompt }}"
      autofocus
    >
    </p-terminal>
  </div>
  <div class="col-6">
    <p-carousel
      *ngIf="this.displayFigures"
      [value]="matlabResponse.figures"
      [numVisible]="1"
      [numScroll]="1"
    >
      <ng-template let-figure pTemplate="item">
        <div class="grid">
          <div class="col-1">
            <button
              pButton
              class='p-button-text'
              (click)="closeFigure(figure.id)">X</button>
          </div>
          <div class="col-10">
            <img src="{{ figure.base64 }}" />
          </div>
        </div>
      </ng-template>
    </p-carousel>
  </div>
</div>
```

app.component.scss:

```
:host ::ng-deep .p-terminal {
  font-family: monospace;
  height: 27rem;
  font-size: 130%;
}

:host ::ng-deep .p-terminal-content  {
  font-family: monospace;
  font-size: 100%;
  padding: 10px 0px 0px 0px;
}

:host ::ng-deep .p-terminal-prompt  {
  font-family: monospace;
  font-size: 100%;
  padding: 10px 10px 10px 0px;
}

:host ::ng-deep .p-terminal-response {
  font-family: monospace;
  font-size: 100%;
  padding: 7px 10px;
}

:host ::ng-deep .p-terminal-input {
  font-family: monospace;
  font-size: 100%;
  padding: 0px 10px;
}

:host ::ng-deep .p-carousel .p-carousel-indicators .p-carousel-indicator.p-highlight
button {
  background-color: #aac2d4;
  border-color: transparent;
}

.p-button {
  vertical-align: top;
  position: relative;
}

.p-button.p-button-text:enabled:hover {
  background: rgba(33, 150, 243, 0.04);
  color: #5b99cc;
  border-color: transparent;
}
.p-button.p-button-text {
  background-color: transparent;
  color: #aac2d4;
  border-color: transparent;
}
:host ::ng-deep .p-menubar-root-list>.p-menuitem-active>p-menubarsub>.p-submenu-list {
  display: block;
  width: max-content;
}

.col-1 {
  flex: 0 0 auto;
  padding: 0;
  width: 0.3333%;
}
```

app.component.ts:

```
import { Component } from '@angular/core';
import { TerminalService } from 'primeng/terminal';
import { Subscription } from 'rxjs';
import { MatlabService } from './matlab.service';
import { Figure, MatlabResponse, MatlabSession } from './models';
import { MenuItem, Message } from 'primeng/api';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss'],
  providers: [TerminalService],
})
export class AppComponent {
  title = 'front';
  subscription: Subscription = new Subscription();
  session: MatlabSession = new MatlabSession();
  sessions: MatlabSession[];
  matlabResponse: MatlabResponse;
  menuItems: MenuItem[];
  element: HTMLElement;
  msgs: Message[];
  menuItemSessions: MenuItem[];
  displayTerminal: boolean;
  displayFigures: boolean;
  figures: Figure[];
  prompt: string;

  constructor(
    private terminalService: TerminalService,
    private matlabService: MatlabService,
  ) {
    this.terminalService.commandHandler.subscribe((command) => {
      this.prompt = "<<";
      this.matlabService.runCommand(this.session.sid, command).subscribe(
        (result) => {
          this.matlabResponse = new MatlabResponse();
          this.matlabResponse = result as MatlabResponse;
          this.terminalService.sendResponse(this.matlabResponse.result);
          if(command === 'clc'){
            this.clearConsole();
          }
          this.plotFigures();
          this.prompt = ">>";
        },
        (error) => {
          console.log(error);
          this.msgs = [];
          this.msgs.push({
            severity: 'error',
            detail: error,
          });
        }
      );
    });
  }

  ngOnInit() {
    this.displayTerminal = false;
    this.msgs = [];
    this.element = document.getElementById('termi') as HTMLElement;
    this.getSessions();
    this.figures = [];
    this.prompt = ">>";
  }

  getMenuItems(): MenuItem[] {
```

```
  this.menuItems = [];

  this.menuItems.push( {
    label: 'New',
    icon: 'pi pi-fw pi-plus',
    command: () => this.newWorkspace(),
  });
  if(this.menuItemSessions.length > 0){
    this.menuItems.push({
      label: 'Join',
      icon: 'pi pi-fw pi-sign-in',
      items: this.menuItemSessions,
    });
  }
  if(this.session.pid){
    this.menuItems.push({
      label: 'Stop/Restart',
      icon: 'pi pi-fw pi-replay',
      items: [
        {
          label: 'Close',
          icon: 'pi pi-fw pi-trash',
          command: () => this.closeWorkspace(this.session),
        },
        {
          label: 'Restart',
          icon: 'pi pi-fw pi-refresh',
          command: () => this.restartWorkspace(this.session),
        },
      ]
    });
  }
  return this.menuItems;
}

ngOnDestroy() {
  if (this.subscription) {
    this.subscription.unsubscribe();
  }
}

newWorkspace(): void {
  this.displayFigures= false;
  this.displayTerminal = false;
  this.msgs = [];
  this.msgs.push({
    severity: 'warn',
    detail: 'Starting New Workspace...',
  });
  this.matlabService.newWorkspace().subscribe(
    (result) => {
      this.session = new MatlabSession();
      this.session = result.session as MatlabSession;
      sessionStorage.setItem('currentSession', JSON.stringify(this.session));
      this.msgs = [];
      this.msgs.push({
        severity: 'success',
        detail: result.result,
      });
      this.displayTerminal = true;
      this.getSessions();
    },
    (error) => {
      console.log(error);
      this.msgs.push({ severity: 'error', detail: error });
    }
  );
}
```

```
getSessions() {
  this.menuItemSessions = [];
  this.matlabService.getSessions().subscribe(
    (result) => {
      this.sessions = result.sessions as MatlabSession[];
      const runningSessions = this.sessions.filter((each) => each.pid);
      if (runningSessions.length === 0) {
        this.msgs = [];
        this.msgs.push({
          severity: 'info',
          detail: 'No Matlab workspaces running!',
        });
      } else {
        runningSessions.forEach((session) => {
          this.menuItemSessions.push({
            label: 'Workspace ' + session.sid + ' (PID=' + session.pid+')',
            command: () => this.joinWorkspace(session),
          });
        });
      }
      this.getMenuItems();
    },
    (error) => {
      console.log(error);
    }
  );
}

joinWorkspace(session: MatlabSession) {
  this.displayTerminal = false;
  this.displayFigures= false;
  this.session = session;
  this.msgs = [];
  this.msgs.push({
    severity: 'success',
    detail: 'Joined Workspace ' + session.sid,
  });

  this.clearConsole();
  this.matlabService.getFigures(session.sid).subscribe(
    (result) => {
      this.matlabResponse = result as MatlabResponse;
      this.getMenuItems();
      this.displayTerminal = true;
      this.plotFigures();
    },
    (error) => {
      console.log(error);
    }
  );
}

closeWorkspace(session: MatlabSession) {
  this.displayTerminal = false;
  this.displayFigures = false;
  this.msgs = [];
  this.msgs.push({
    severity: 'success',
    detail: 'Closing Workspace...' + session.sid,
  });
  this.matlabService.stopMatlab(session.sid, false).subscribe(
    (result) => {
      this.msgs = [];
      this.msgs.push({
        severity: 'success',
        detail: result.result,
      });
      this.session = new MatlabSession();
      this.getSessions();
```

```
      },
      (error) => {
        console.log(error);
        this.msgs.push({ severity: 'error', summary: '', detail: error });
      }
    );
  }

  restartWorkspace(session: MatlabSession) {
    this.displayTerminal = false;
    this.msgs = [];
    this.msgs.push({
      severity: 'success',
      detail: 'Restarting Workspace...' + session.sid,
    });

    this.matlabService.stopMatlab(session.sid, true).subscribe(
      (result) => {
        this.session = result.session as MatlabSession;
        this.msgs = [];
        this.msgs.push({
          severity: 'success',
          detail: result.result,
        });

        this.getSessions();
        this.displayTerminal = true;
      },
      (error) => {
        console.log(error);
        this.msgs.push({ severity: 'error', detail: error.error });
      }
    );
  }

  plotFigures(): void {
    this.figures=[];
    this.displayFigures = false;
    if (this.matlabResponse.figures.length > 0) {
      this.displayFigures = true;
    }
  }

  closeFigure(id: number): void {
    this.figures = this.figures.filter((figure) => figure.id !== id);
    this.matlabResponse.figures = this.matlabResponse.figures.filter((figure) =>
figure.id !== id);
    this.prompt = "<<";
    this.matlabService.runCommand(this.session.sid, 'close '+id).subscribe(
      (result) => {
        this.plotFigures();
        this.prompt = ">>";
      },
      (error) => {
        console.log(error);
      }
    );
  }

  clearConsole(): void{
    if(document.getElementsByClassName('p-terminal-content')[0]){
    document.getElementsByClassName('p-terminal-content')[0].innerHTML = "";
    }
  }
}
```

app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { TerminalModule } from 'primeng/terminal';
import { HttpClientModule } from '@angular/common/http';
import { MenubarModule } from 'primeng/menubar';
import { MessagesModule } from 'primeng/messages';
import { CarouselModule } from 'primeng/carousel';
import {ButtonModule} from 'primeng/button';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    TerminalModule,
    HttpClientModule,
    MenubarModule,
    MessagesModule,
    CarouselModule,
    ButtonModule
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

matlab.service.ts:

```
import { Injectable } from '@angular/core';
import {
  HttpClient,
  HttpHeaders,
  HttpUrlEncodingCodec,
} from '@angular/common/http';
import { Observable } from 'rxjs';
import { MatlabResponse } from './models';

@Injectable({
  providedIn: 'root',
})
export class MatlabService {
  constructor(private http: HttpClient) {}
  private apiUrl = 'http://localhost:8000';

  getSessions(): Observable<MatlabResponse> {
    return this.http.get<MatlabResponse>(this.apiUrl + '/sessions');
  }

  newWorkspace(): Observable<MatlabResponse> {
    return this.http.get<MatlabResponse>(this.apiUrl + '/newSession');
  }

  runCommand(sid: number, commands: string): Observable<MatlabResponse> {
    return this.http.get<MatlabResponse>(
      this.apiUrl + '/run?sid='+ sid + '&commands=' + encodeURIComponent(commands)
    );
  }

  stopMatlab(sid: number,restart: boolean = false): Observable<MatlabResponse> {
    return this.http.get<MatlabResponse>(
      this.apiUrl + '/stopMatlab?sid=' + sid + '&restart=' + restart
    );
  }
```

```
  getFigures(sid: number) {
    return this.runCommand(sid, 'figures');
  }
}


export class MatlabResponse {
  result: string;
  figures: Figure[];
  session: MatlabSession;
  sessions: MatlabSession[];
}

export class MatlabSession {
  pid: number;
  sid: number;
}

export class Figure {
  id: number;
  base64: string;
}
```

6.1.1 Backend

Página principal del repositorio: https://github.com/nilodude/mow-api

Historial de cambios: https://github.com/nilodude/mow-api/commits/master

Se adjuntan los ficheros del backend:

main.py:

```
from typing import Optional
from fastapi import FastAPI
from pydantic import BaseModel
from matlab_interface import MatlabInterface
from utils import Utils
from session import Session
from fastapi.middleware.cors import CORSMiddleware

import json
import os
import os.path

tags_metadata = [
    {
        "name": "main",
        "description": "Operations needed for the main application",
    },
    {
        "name": "extra",
        "description": "",
    }
]

app = FastAPI(title="Matlab Online Workspace API")

origins = [
    "http://localhost:4200",
]
```

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

MAX_SESSIONS = 3
HOME = 'D:\\Dropbox\\tfg\\Shazam-MATLAB\\app\\'

utils = Utils()
sessions = [Session(i) for i in range(1, MAX_SESSIONS+1)]


@app.get("/")
def read_root():
    return {"Add /docs to URL to acces Matlab Online Workspace API"}


def getSession(sid: int):
    availables = list(
        filter(lambda s:  s.sid == sid, sessions))
    return availables[0] if len(availables) > 0 else None


@app.get("/sessions", tags=["main"])
def getSessions():
    return {"sessions": [s.toJSON() for s in sessions]}


@app.get("/taskList", tags=["main"])
def tasklist():
    tasks = utils.taskList()
    return utils.toJSON(tasks)


@app.get("/newSession", tags=["main"])
def newSession():
    availables = list(filter(lambda s: s.pid is None, sessions))
    itsNotFull = len(availables) > 0

    if itsNotFull:
        utils.printS('Available Sessions:', availables)
        s = availables[0]
        index = sessions.index(s)
        sessions.pop(index)
        print('Initializing Matlab Session '+str(s.sid)+'...')
        s.matlab = MatlabInterface()
        # s.matlab.run_command('checkStart',False)
        s.pid = s.matlab.run_command("clear,feature('getpid')", False)
        message = 'New Matlab process with PID='+s.pid

        sessions.insert(index, s)

        utils.printS('Updated Sessions: '+message, sessions)
        response = {"result": message, "session": s.toJSON()}
    else:
        message = 'No available sessions'
        response = {"result": message}
    return response
```

```
@app.get("/startMatlab", tags=["main"])
def startMatlab(sid: int):
    response = None
    s = getSession(sid)

    isItAvailable = (s is not None) & (s.pid is None)

    if isItAvailable:
        utils.printS('Selected Session:', [s])
        index = sessions.index(s)
        sessions.pop(index)
        print('Initializing Matlab Session '+str(sid)+'...')
        s.matlab = MatlabInterface()
        # s.matlab.run_script('checkStart')
        s.pid = s.matlab.run_command("clear,feature('getpid')", False)
        s.matlab.run_command('clear', False)
        message = 'New Matlab process with PID='+s.pid
        sessions.insert(index, s)

        utils.printS('Updated Sessions: '+message, sessions)
        response = {"result": message, "session": s.toJSON()}
    else:
        message = 'Session ' + str(sid)+' not available, already running PID=' +
str(s.pid)
        response = {"result": message}
        print(message)
    return response


@app.get("/stopMatlab", tags=["main"])
def stopMatlab(sid: int, restart: Optional[bool] = False):
    option = ' Restart ' if restart else ' Stop '
    print('Session '+str(sid)+option+'Selected')
    session = getSession(sid)
    if hasattr(session, 'matlab') & (session.pid is not None):
        session.matlab.stop()
        session.pid = None
        session.matlabPID = None
        session.matlab = None
        msg = 'Session '+str(sid) + ' stopped'
        response = startMatlab(session.sid) if restart else {"result": msg}
    else:
        msg = 'Session '+str(sid) + ' is not currently running!'
        response = {"result": msg}
        print(msg)
    return response


@app.get("/run", tags=["main"])
def run(sid: int, commands: str):
    session = getSession(sid)
    figures = []
    if hasattr(session, 'matlab') & (session.pid is not None):
        print('Session '+str(sid)+': ')
        res = session.matlab.run_command(commands, True)

        figures = session.matlab.run_command('figures', False)
        try:
            figures = figures.replace('\r', '').replace('\n', '')
        except:
            figures = figures
        try:
            result = json.loads(res, strict=False)
        except:
            result = res
        try:
            figures = json.loads(figures, strict=False)
        except:
            figures = figures
```

```
    else:
        result = 'Session '+str(sid) + ' is not currently running!'
    return {"result": result, "figures": figures}


@app.get("/getJSON", tags=["extra"])
def getJSON(fileName: str):
    with open(HOME+'db\\json\\'+fileName, 'r') as f:
        res = f.read()
    jsonRes = json.loads(res, strict=False)
    return jsonRes
```

matlab_interface.py:

```
try:
    input = raw_input
except NameError:
    pass

import os
from io import StringIO

global import_fail
try:  # Check if the Matlab Engine is installed
    import matlab.engine
    from matlab.engine import RejectedExecutionError as MatlabTerminated
except ImportError:
    print("Matlab Engine for Python cannot be detected. Please install it for the
extension to work")
    import_fail = True
else:
    import_fail = False


class MatlabInterface:
    global import_fail

    def __init__(self):
        # OS checks related work
        if os.name == 'nt':
            self.cls_str = 'cls'
        else:
            self.cls_str = 'clear'
        if not import_fail:
            print("Starting Matlab...")
            self.eng = matlab.engine.start_matlab()
        else:
            print("Could not start Matlab")

    def stop(self):
        print("stopping Matlab...")
        self.eng.quit()
        return "stopped OK"

    def run_command(self, command, verbose):
        if not import_fail:
            try:
                if verbose:
                    print("Running Command: {}".format(command))
                stream = StringIO()
                err_stream = StringIO()
                self.eng.eval(command, nargout=0,
                              stdout=stream, stderr=err_stream)
                return stream.getvalue().replace('ans =', '').strip()
            except MatlabTerminated:
                print(stream.getvalue(), err_stream.getvalue(), sep="\n")
                print("Matlab terminated. Restarting the engine...")
```

```
                self.eng = matlab.engine.start_matlab()
                return "Matlab restarted OK"
            except:  # The other exceptions are handled by Matlab
                errList = err_stream.getvalue().split('\n\n')
                newList = [error.replace('\n', '') for error in errList]
                return list(filter(None, newList))
```

task.py:

```
import os

class Task:
    imageName: str
    pid: int
    cpuTime: int
    windowName: str

    def __init__(self, task):
        fields = task.split(',')
        fields = [f.replace('"', '') for f in fields]
        # print(fields)
        self.imageName = fields[0]
        self.pid = fields[1]
        time = fields[7].split(':')
        h = int(time[0])
        m = int(time[1])
        s = int(time[2])
        h = h if h == 0 else h*3600
        m = m if m == 0 else m*60
        self.cpuTime = int(h+m+s)
        self.windowName = fields[8]
```

session.py:

```
import json
import os
from matlab_interface import MatlabInterface

class Session:
    sid = None
    matlab: MatlabInterface
    pid = None

    def __init__(self, sid):
        self.sid = sid

    def toJSON(self):
        return {"sid": self.sid, "pid": self.pid}
```

utils.py:

```
import json
import os
import subprocess
from task import Task


class Utils:

    def taskList(self):
        output = subprocess.run(
            'tasklist /FI "imagename eq matlab.exe" /v /fo csv /nh', check=True,
stdout=subprocess.PIPE).stdout
        if str(output).__contains__('criterios especificados.'):
            tasks = []
```

```
        else:
            output = output.decode('utf-8')
            taskList = list(filter(None, output.split('\r\n')))
            tasks = [Task(task) for task in taskList]
        return tasks

    def printS(self, title, sessions):
        print(title)
        for s in sessions:
            print(' ', vars(s))

    def toJSON(self, input):
        response = []
        for s in input:
            if not hasattr(s, 'matlab'):
                response.append(json.loads(json.dumps(vars(s))))
            else:
                response.append({
                    "pid": s.pid, "sid": s.sid})
        return response
```

figures.m:

```
g = groot;
numFigures = length(g.Children);

if ~isempty(g.Children)
    response = '[';

    for i = 1:length(g.Children)
        bytes = figToImStream('figHandle', g.Children(i), 'imageFormat', 'bmp',
'outputType', 'uint8');

        b64 = base64encode(bytes);

        response = strcat(response,'{"id":"');
        response = strcat(response,num2str(i));
        response = strcat(response,'","base64":"');
        response = strcat(response, b64);
        response = strcat(response,'"}');
        if (numFigures > 1 && i ~= numFigures)
            response = strcat(response, ',');
        end

    end

    response = strcat(response, ']');

    disp(response);
end
clear b64
clear bytes
clear i
clear response
clear g
clear numFigures
```

base64encode.m:

```
function output = base64encode(input)
   %BASE64ENCODE Encode a byte array using Base64 codec.
   %
   %    output = base64encode(input)
   %
   % The function takes a char, int8, or uint8 array INPUT and returns Base64
   % encoded string OUTPUT. JAVA must be running to use this function. Note
   % that encoding doesn't preserve input dimensions.
   %
   % See also base64decode

   % error(nargchk(1, 1, nargin));
   error(javachk('jvm'));
   if ischar(input), input = uint8(input); end

    output = char(org.apache.commons.codec.binary.Base64.encodeBase64Chunked(input))';

end
```

REFERENCIAS

[1] D. Ellis, «Robust Landmark-Based Audio Fingerprinting,» 2009. [En línea]. Available: https://www.ee.columbia.edu/~dpwe/resources/matlab/fingerprint/. [Último acceso: Junio 2019].

[2] A. Gascón, «Shazam-MATLAB,» [En línea]. Available: https://github.com/AlexGascon/Shazam-MATLAB. [Último acceso: Junio 2019].

[3] A. L.-C. Wang, «An Industrial Strength Audio Search Algorithm,» de *International Symposium on Music Information Retrieval*, Baltimore, 2003.

[4] A. Pommel, «Matlab Interactive Terminal for Visual Studio Code,» [En línea]. Available: https://github.com/apommel/vscode-matlab-interactive-terminal. [Último acceso: Diciembre 2020].

[5] MathWorks, «Start and Stop MATLAB Engine for Python,» [En línea]. Available: https://www.mathworks.com/help/matlab/matlab_external/start-the-matlab-engine-for-python.html. [Último acceso: Enero 2021].

[6] M. F. Rafid, «FastAPI - The Good, the bad and the ugly.,» [En línea]. Available: https://dev.to/fuadrafid/fastapi-the-good-the-bad-and-the-ugly-20ob. [Último acceso: Marzo 2021].

[7] Angular Guide, «Communicating with backend services using HTTP,» [En línea]. Available: https://angular.io/guide/http. [Último acceso: Abril 2021].

[8] Angular Guide, «Using observables to pass values,» [En línea]. Available: https://angular.io/guide/observables. [Último acceso: Abril 2021].

[9] Angular Guide, «Setting up the local environment and workspace,» [En línea]. Available: https://angular.io/guide/setup-local. [Último acceso: Abril 2021].

[10] MathWorks, «External Language Interfaces,» 2021. [En línea]. Available: https://www.mathworks.com/help/matlab/external-language-interfaces.html?s_tid=CRUX_lftnav. [Último acceso: Enero 2021].

[11] MathWorks, «Versions of Python Compatible with MATLAB Products by Release,» [En línea]. Available: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/support/sysreq/files/python-compatibility.pdf. [Último acceso: Enero 2021].

[12] MathWorks, «Install MATLAB Engine API for Python,» 2021. [En línea]. Available: https://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html. [Último acceso: Enero 2021].

[13] MathWorks, «Limitations to MATLAB Engine API for Python,» [En línea]. Available: https://www.mathworks.com/help/matlab/matlab_external/limitations-to-the-matlab-engine-for-python.html. [Último acceso: Enero 2021].

[14] «What is a REST API?,» [En línea]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[Último acceso: Junio 2021].

[15] «Introducing JSON,» [En línea]. Available: https://www.json.org/json-en.html. [Último acceso: Julio 2021].

[16] «When do I use path params vs. query params in a RESTful API?,» [En línea]. Available: https://stackoverflow.com/questions/30967822/when-do-i-use-path-params-vs-query-params-in-a-restful-api. [Último acceso: Agosto 2021].

[17] ASGI Team, «ASGI Documentation,» [En línea]. Available: https://asgi.readthedocs.io/en/latest/. [Último acceso: Abril 2021].

[18] ASGI Team, «ASGI Documentation Introduction,» [En línea]. Available: https://asgi.readthedocs.io/en/latest/introduction.html. [Último acceso: Abril 2021].

[19] P. Eby, «Python Web Server Gateway Interface v1.0.1,» [En línea]. Available: https://www.python.org/dev/peps/pep-3333/. [Último acceso: Abril 2021].

[20] MDN Web Docs, «General asynchronous programming concepts,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Concepts. [Último acceso: Mayo 2021].

[21] S. Ramírez, «FastAPI,» [En línea]. Available: https://fastapi.tiangolo.com/. [Último acceso: Mayo 2021].

[22] A. D. Eastlake, «Reserved Top Level DNS Names,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc2606. [Último acceso: Agosto 2021].

[23] Y. Altman, «Undocumented feature() function,» 2010. [En línea]. Available: https://undocumentedmatlab.com/articles/undocumented-feature-function. [Último acceso: Abril 2021].

[24] «Tasklist,» [En línea]. Available: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/tasklist. [Último acceso: Febrero 2021].

[25] Angular Guide, «Workspace and project file structure,» [En línea]. Available: https://angular.io/guide/file-structure. [Último acceso: Junio 2021].

[26] PrimeTek, «Menubar,» [En línea]. Available: https://primefaces.org/primeng/showcase/#/menubar. [Último acceso: Junio 2021].

[27] PrimeTek, «Messages,» [En línea]. Available: https://primefaces.org/primeng/showcase/#/messages. [Último acceso: Junio 2021].

[28] PrimeTek, «Terminal,» [En línea]. Available: https://www.primefaces.org/primeng/showcase/#/terminal. [Último acceso: Junio 2021].

[29] MDN Web Docs, «Document,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document. [Último acceso: Julio 2021].

[30] PrimeTek, «Carousel,» [En línea]. Available: https://www.primefaces.org/primeng/showcase/#/carousel. [Último acceso: Julio 2021].

[31] MDN Web Docs, «Data URLs,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URIs. [Último acceso: Julio 2021].

[32] Angular Guide, «How Angular assigns values to template variables,» [En línea]. Available: https://angular.io/guide/template-reference-variables#how-angular-assigns-values-to-template-variables. [Último acceso: Junio 2021].

[33] «When to use 'npm start' and when to use 'ng serve'?,» [En línea]. Available: https://stackoverflow.com/questions/40190538/when-to-use-npm-start-and-when-to-use-ng-serve. [Último acceso: Junio 2021].