**AOS: Práctica 2**

Despliegue y puesta en marcha de un API de gestión de productos

Miembros del grupo:

* Amit Karamchandani Batra
* Rubén García Cid
* Kristian Magjistari Ziu

1. **Memoria descriptiva del despliegue de los servicios**

Descripción de los contenedores usados:

Para Docker-Compose:

* **pollitorex/(productos | pedidos | facturación | logging):** Versión modificada de Prism, que incluye la especificación de cada servicio dentro de la imagen, de esta forma, arrancamos Prism con dicha especificación para poder hacer el mocking de la API.

**Nota**: Estas 4 imágenes no se encuentran en Docker Hub, es necesario ejecutar el fichero ‘buildAll.bat’ presente en la carpeta ‘Servicios’.

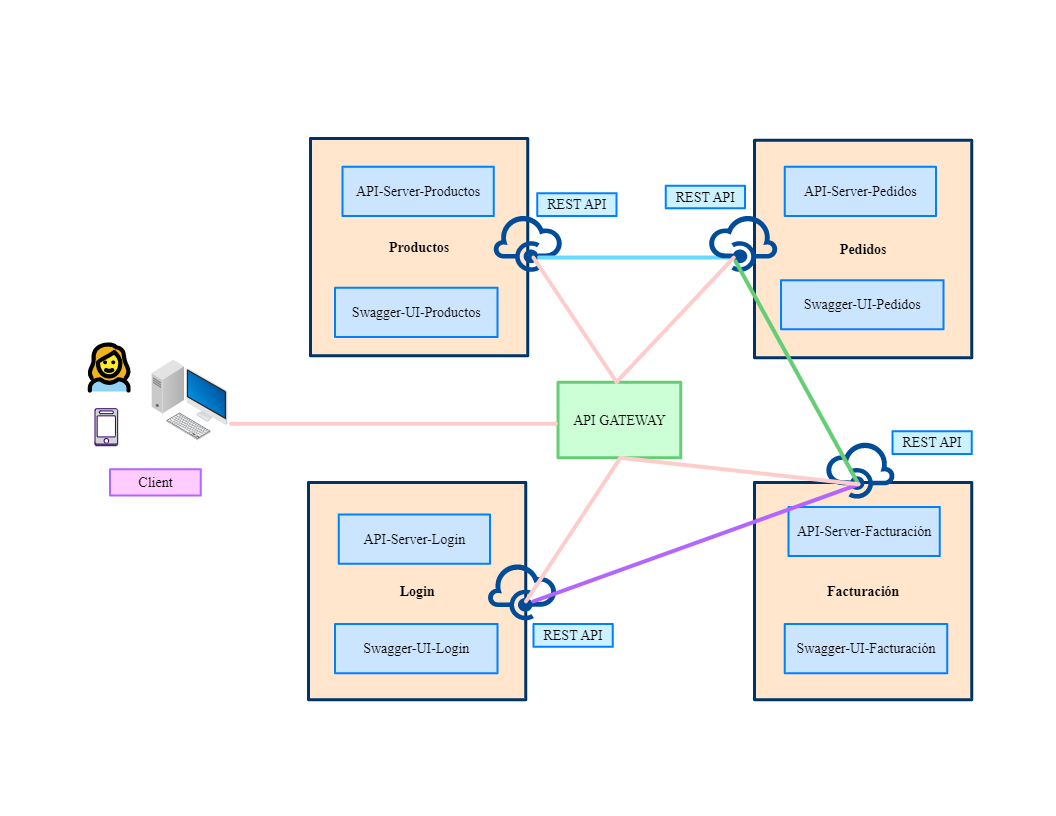
* **swaggerapi/swagger-ui**: Imagen oficial de Swagger UI, la hemos usado para poder interactuar con nuestra especificación, sin necesidad de haberla implementado previamente. De esta forma, podremos comprobar de una forma muy visual los distintos ‘endpoints’ expuestos en nuestra API, formatos de entrada y salida de los datos, las diversas operaciones a las que podemos llamar, los códigos de error que podemos obtener, entre otras cosas.

Para Kubernetes:

En config.yaml hemos especificado los servicios y el despliegue de los mismos siendo el api-mock y el swagger-ui.

* **pollitorex/(productos | pedidos | facturación | logging):** Las mismas cuatro imágenes ya usadas para realizar el mocking de la API de cada uno de los cuatro servicios que usamos para Docker Compose.
* **pollitorex/swagger-ui-(productos | pedidos | facturación | logging):** Por cada uno de los 4 servicios (productos, pedidos, facturación y logging), hemos creado una imagen modificada de Swagger UI, hemos tomado como base la imagen oficial de Swagger-UI, introducimos el fichero ‘openapi.yaml’, en el que se encuentra nuestra especificación, dentro de la imagen, en la ruta ./usr/share/nginx/html/, para poder cargarla desde Swagger-UI introduciendo el nombre del fichero que contiene la especificación en la barra de dirección de la interfaz de Swagger, luego guardamos el contenedor en vivo en una imagen, usando el comando ‘commit’ y lo subimos a un repositorio de DockerHub (<https://hub.docker.com/u/pollitorex>), mediante los comandos ‘tag’ y ‘push’. Como hemos mencionado anteriormente, esta imagen la usamos para interactuar con nuestra especificación fácilmente.

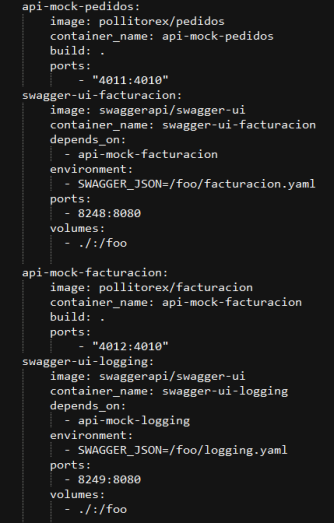
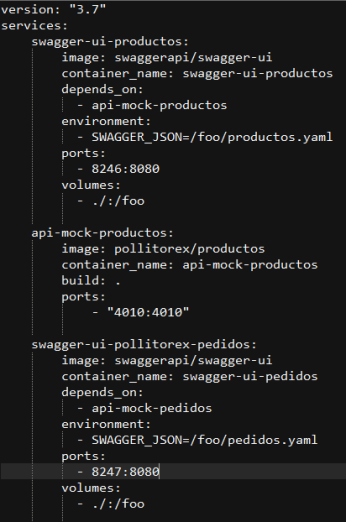
También hemos necesitado definir ‘NodePorts’ para poder exponer los puertos de cada pod al host y así poder comprobar el resultado del despliegue.

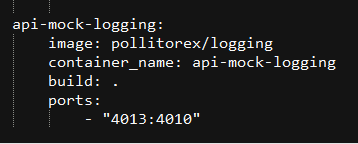
**Diagrama de Despliegue de los Servicios**

1. **Despliegue con Docker Compose**

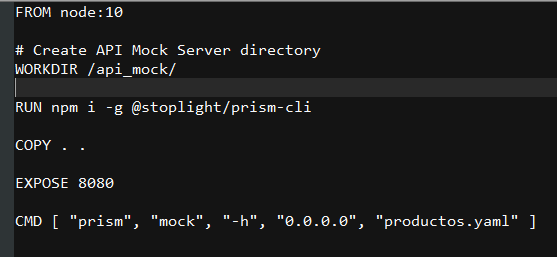
Partiendo de la especificación corregida para solventar aquellos fallos que fueron observadores por el profesor, hemos añadido los siguientes ficheros a la carpeta llamada “Docker”, desde la cual se realizará el despliegue con Docker Compose:

* Un fichero YAML de docker-compose, en el cual definimos la especificación necesaria para iniciar todos los servicios de los que se compone nuestro sistema, además de todos aquellos parámetros (dependencias con otros contenedores, variables de entorno, puertos, volúmenes, etc.) que se requieren para que puedan funcionar correctamente y puedan comunicarse entre ellos y con el host. En total hay 8 contenedores, por cada servicio tenemos, un contenedor de Swagger-UI (swagger-ui-<nombre\_del\_servicio>) y otro con la imagen de Prism modificada con cada una de las especificaciones de los distintos servicios incluida dentro de la propia imagen (api-mock-<nombre\_del\_servicio>).



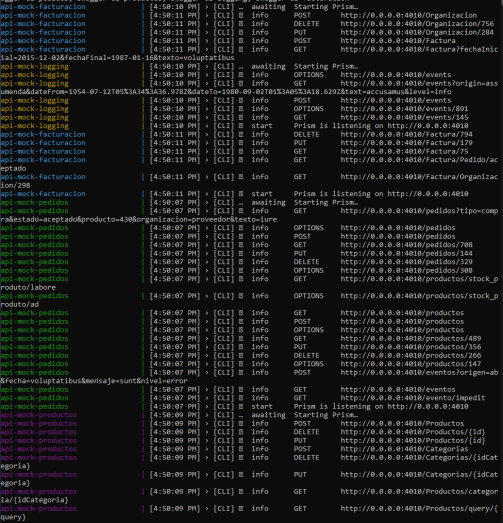
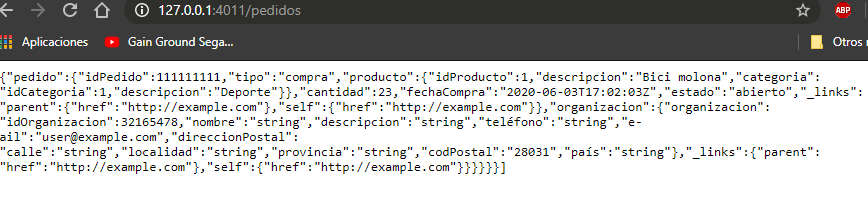


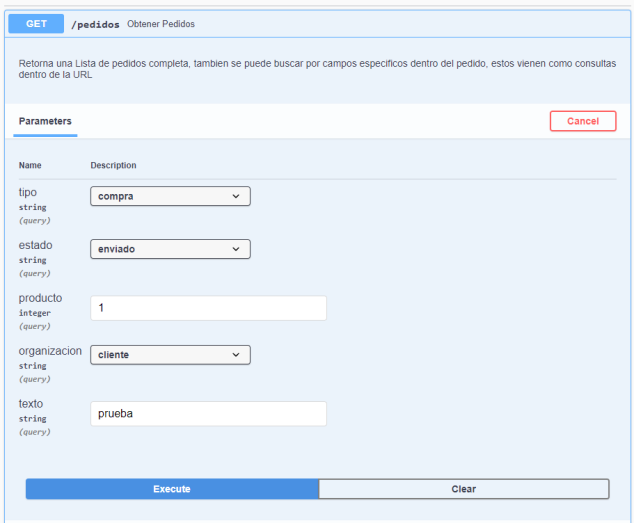
* Cuatro ficheros Dockerfile usado para construir la imagen modificada de Prism, en la cual incluimos las especificaciones de cada uno de los servicios que componen nuestro sistema para que pueda usarse durante el mock.

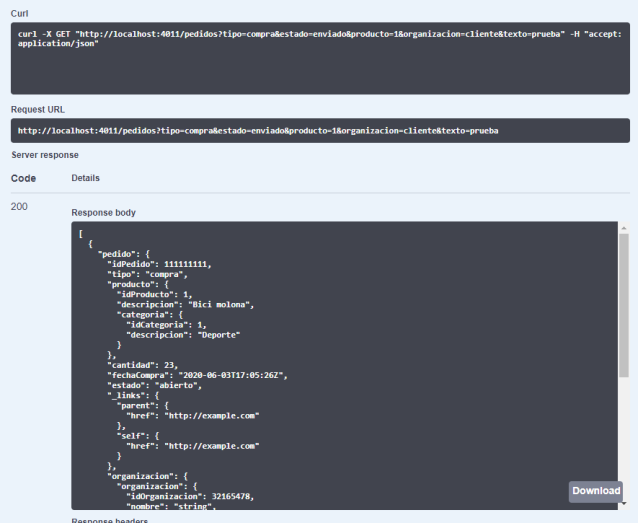


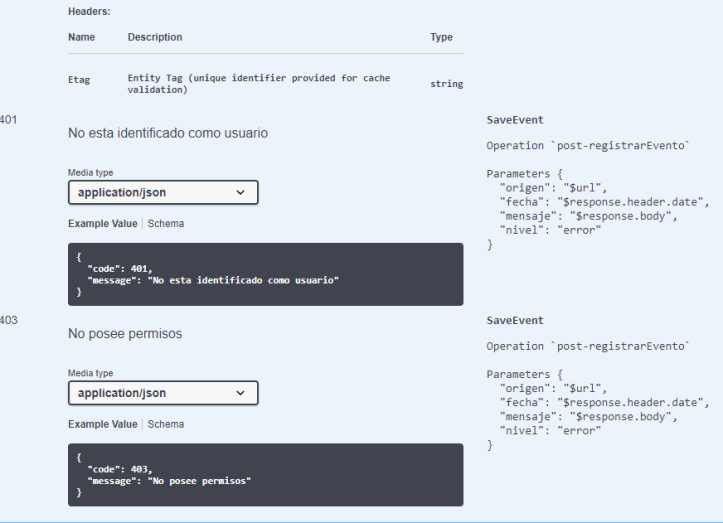
El resdo de Dockerfile son iguales, solo cambia el archivo de especificación del servicio introducido (facturación.yaml, pedidos.yaml y logging.yaml).

* Un fichero llamado ‘Run.bat’ que ejecutara los comandos (docker-compose build y docker-compose up) necesarios para poder arrancar nuestro conjunto de servicios correctamente.









1. **Despliegue con Kubernetes**

Hemos incluido todos los ficheros & ejecutables necesarios (carpetas de configuración, kompose.exe, kubectl.exe, etc) dentro de una carpeta llamada “Kubernetes”. Dentro de esta, tenemos 2 ficheros .bat.

* Run.bat incluye la inicialización de Minikube y el comando de despliegue: 

seguido de el comando:



con el cual se obtiene la IP y puerto del servicio de Swagger-ui.

* Stop.bat lo incluimos para hacer “Minikube stop” seguido de “Minikube delete” para parar rápidamente el cluster de Kubernetes si habíamos hecho algún cambio y queríamos ver el resultado.
* Un fichero “config.yaml” el cual contiene la configuración necesaria para el despliegue de todos los servicios, consiste en 2 ‘services’ por cada uno de los 4 servicios, uno para Swagger-UI y otro para el API mock, en estos services se ha especificado el puerto del servicio, el puerto del pod y el NodePort, para poder acceder desde el host. Además, se han definido 2 ‘deployments’ por cada servicio, uno para Swagger-UI y otro para el API mock, en ellos se han definido la imagen de cada contenedor, el nombre del contenedor, el puerto del contenedor y la política de reinicio.

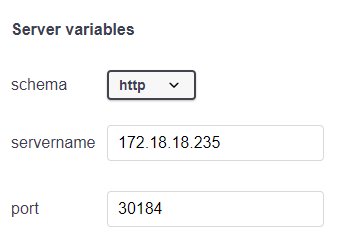
1. **Simulación del comportamiento de la API (Mocking)**

Hemos usado la herramienta de mocking ‘Prism’ para simular el comportamiento de nuestra API, para ello modificamos la imagen original mediante un fichero Dockerfile para incluir nuestra especificación dentro de ella, a esta imagen la hemos nombrado ‘api-mock’.

Para montar el servidor de Mocking, una vez puesto en marcha los servicios con el comando ‘Run.bat’, ya sea mediante Docker Compose o Kubernetes, debemos dirigirnos a la IP mostrada por pantalla tras ejecutar el script, una vez introducida dentro del navegador, necesitaremos también la IP del contenedor en el que se ejecuta api-mock para ello ejecutamos el siguiente comando:



E introducimos la IP y el puerto en la sección ‘Server Variables’ en la interfaz de Swagger-UI:



Para el caso del despliegue con Kubernetes, también necesitaremos escribir manualmente en la barra de dirección de Swagger-UI el nombre de la especificación a cargar:

Para el servicio de Productos: ‘productos.yaml’

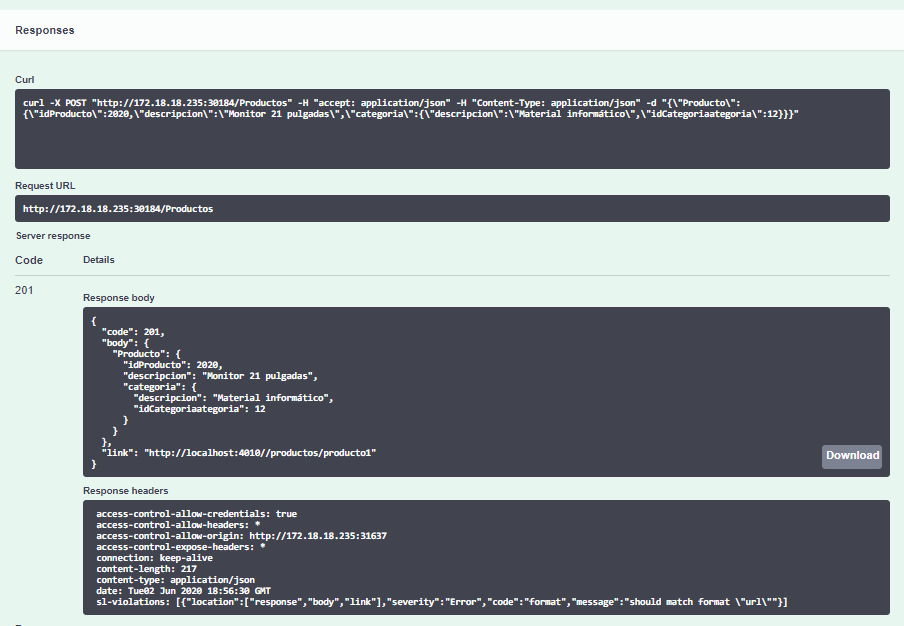
Para el servicio de Pedidos: ‘facturación.yaml’

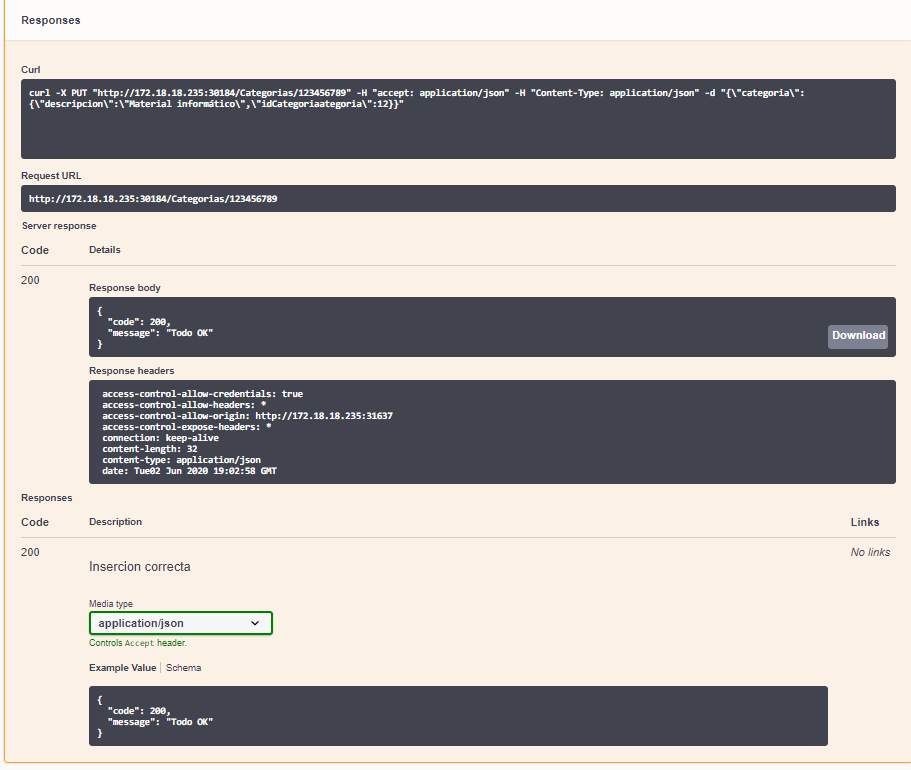
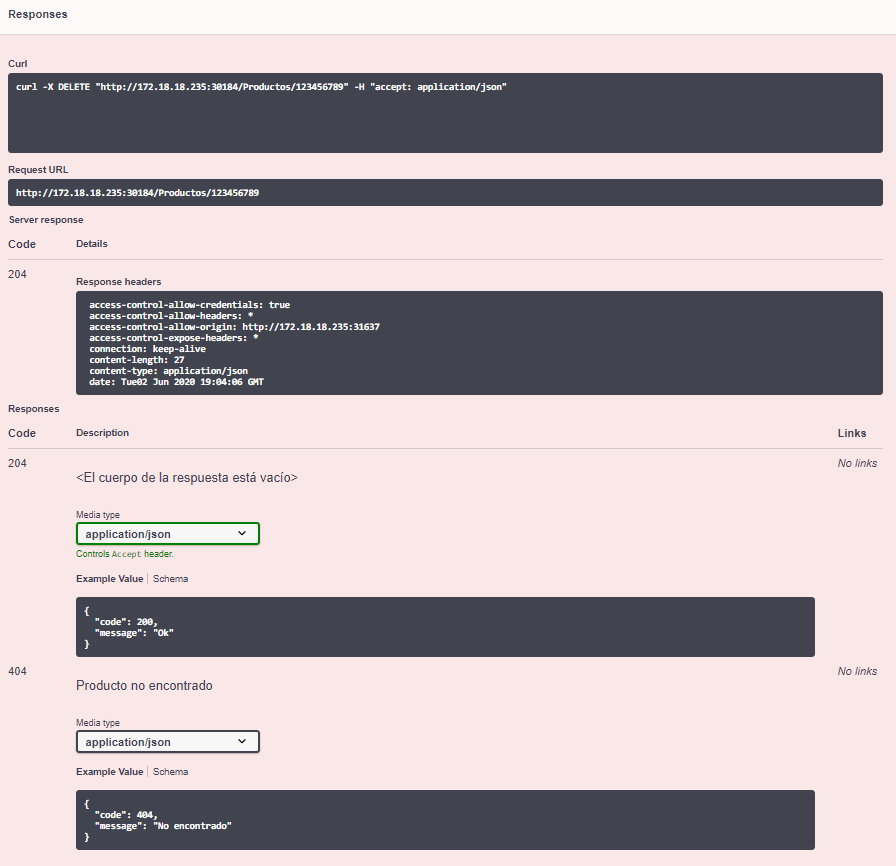
Para el servicio de Facturación: ‘facturación.yaml’

Para el servicio de Logging: ‘facturación.yaml’

La simulación, como era de esperar al no haber ninguna lógica implementada, no es completa, devuelve las respuestas esperadas cuando probamos operaciones como POST, PUT o DELETE, pero las operaciones GET devuelven un valor nulo.

Adjuntamos, capturas de pantalla de las pruebas con Prism:





1. **Implementación**

Hemos usado JavaScript como lenguaje de implementación, usando la plataforma Node JS para ejecutar el servidor como aplicación de escritorio, también hemos usado los paquetes Express, para montar el servidor web, y localStorage, para la persistencia de los datos que se introducen durante las peticiones al servidor.

El grado de implementación no es completo, en concreto, hemos implementado la respuesta en el caso de que la petición haya sido correcta para todas las operaciones, además, hemos implementado algunas respuestas ante casos en los que la solicitud no haya sido correcta o completa, según lo especificado en la API.

Posteriormente, hemos dockerizado la implementación del servidor, titulando la imagen como ‘api-server’, y hemos creado nuevas versiones de Docker Compose y Kubernetes, llamadas ‘Docker Compose – API Server’ y ‘Kubernetes – API Server’, en las cuales se han sustituido el contenedor ‘api-mock’ por la imagen de nuestro servidor implementado.

Para probarlo, se deben seguir los mismos pasos descritos en el apartado anterior.