FUNDAMENTOS DE INGENIERÍA DEL SOFTWARE

MEMORIA TRABAJO PRÁCTICO

GRUPO 5 - PRODUCTS

Máster en Ingeniería del Software: Datos, Cloud y Gestión TI

Universidad de Sevilla

Microservicio Products

Bravo Llanos, Alfonso

García Barco, Pablo

Organización GitHub: https://github.com/fafagorg

Repositorio microservicio: https://github.com/fafagorg/products

Repositorio frontend (común): https://github.com/fafagorg/frontend

Curso 2020-21

1. Nivel de acabado al que nos presentamos

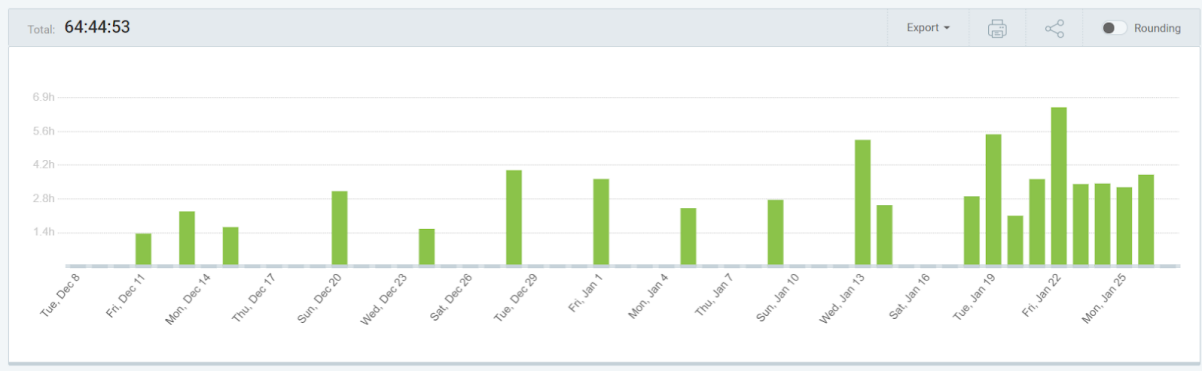
Nivel hasta 10 puntos

2. Análisis de los esfuerzos (en horas)

La gran mayoría del trabajo la hemos desarrollado de forma simultánea, estando ambos miembros de la pareja en llamada colaborativa. De esta manera, hemos avanzado mucho más rápido que por separado, aportándonos ideas mutuamente, localizando y corrigiendo los errores de forma más eficaz, y sobre todo, ambos hemos aprendido sobre el total de la aplicación.

Por ello, no hay diferencias significativas entre las horas dedicadas por cada uno de los dos miembros de la pareja, y por tanto, el análisis de esfuerzos que mostramos a continuación aplicaría para ambos por igual.

Como podemos apreciar en la gráfica inferior (y sobre todo en el reporte dividido por tareas que adjuntamos en el repositorio), antes de las vacaciones de navidad conseguimos dejar lista las funcionalidades más básicas del backend. Durante las fiestas fue algo difícil coordinarnos y coincidir para avanzar, entre compromisos, familia, viajes y celebraciones, algo que se ve reflejado en el descenso de las horas dedicadas durante este período, destinadas principalmente a los tests del backend. Por último, podemos ver cómo tras las vacaciones la dedicación aumentó notablemente, permitiendo también hacer tareas que no podíamos hacer con anterioridad como por ejemplo la integración con los microservicios de otros compañeros o el frontend común. Además, por norma general, un gran número de horas se agrupan en los últimos días, con retoques de última hora, elaboración de la documentación, o revisiones de que todo funciona correctamente.



En el siguiente enlace adjuntamos el reporte de horas detallado, indicando los trabajos realizados para cada bloque, pudiendo analizar más detalladamente las horas dedicadas a cada sección del proyecto, atendiendo al nombre de la tarea correspondiente:

https://github.com/fafagorg/products/tree/documentation

3. Cumplimiento y justificación de los requisitos

Microservicio básico que gestione un recurso:

- El backend debe ser una API REST tal como se ha visto en clase implementando al menos los métodos GET, POST, PUT y DELETE y devolviendo un conjunto de códigos de estado adecuado.

  http://products.fafago-dev.alexgd.es/docs/

Tal y como está recogido en la documentación (previamente enlazada), nuestro microservicio cuenta con un total de ocho métodos, cuya utilidad y uso en nuestra aplicación queda brevemente explicada en el campo “descripción” de cada método. Con estos ocho métodos podremos gestionar de forma completa los productos en nuestra aplicación, realizando sobre ellos todas las operaciones necesarias.

- Debe tener un frontend que permita hacer todas las operaciones de la API (este frontend puede ser individual o estar integrado con el resto de frontends).

Ver requisito “Implementar un frontend con rutas y navegación” del microservicio avanzado, en el que explicamos la configuración del frontend, las distintas vistas que lo conforman, así como las operaciones a realizar en cada una de ellas.

- Debe estar desplegado en la nube y ser accesible en una URL.

  http://products.fafago-dev.alexgd.es/

Nuestra API, al igual que los otros tres microservicios de nuestros compañeros, así como el frontend común, se encuentra desplegado en Google cloud, haciendo uso de Kubernetes, tal y como se explica en el requisito de la aplicación avanzada “Uso de Kubernetes para el despliegue completo de todos los microservicios”.

Es accesible en la URL enlazada anteriormente, donde encontraremos el /home (sin funcionalidad) de nuestro microservicio. En la URL “/docs”, adjuntada para el requisito previo se pueden encontrar el resto de endpoints accesibles y funcionales.

- La API que gestione el recurso también debe ser accesible en una dirección bien versionada.

  http://products.fafago-dev.alexgd.es/ (Home)

Tal y como podemos ver en la documentación de swagger de nuestro microservicio, la API está versionada tal y como se muestra en la fotografía superior, con los parámetros “/api/v1”. El endpoint “home”, enlazado anteriormente, es el único que no cuenta con este versionado, al no ser necesario en ese caso en particular, ya que no es una URL funcional.

- Se debe tener una documentación de todas las operaciones de la API incluyendo las posibles peticiones y las respuestas recibidas.

  http://products.fafago-dev.alexgd.es/docs/

  (Ver requisito del microservicio avanzado de Documentación con swagger)

- Debe tener persistencia utilizando MongoDB

  https://github.com/fafagorg/products/blob/master/k8s/pvc.yaml

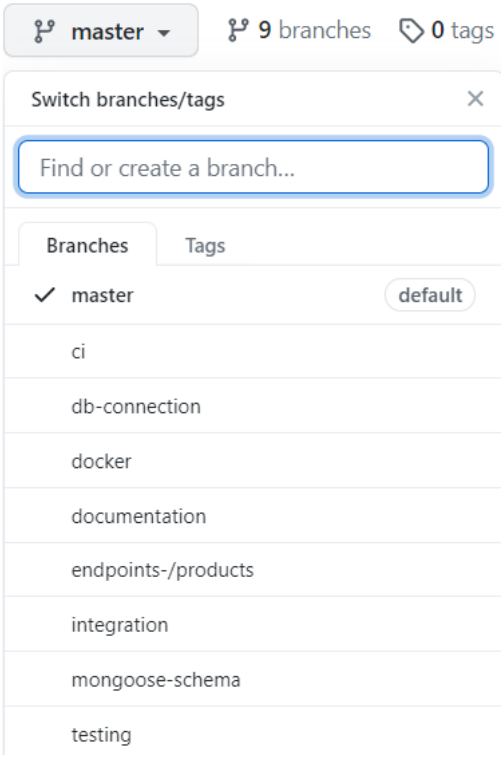
Hemos creado un volumen para la persistencia de datos (enlazado anteriormente), con lo que cada vez que se redespliega se asocia este volumen al despliegue, haciendo que los datos almacenados se mantengan persistentes, independientemente de que haya cambios en los pods.

- Utilizar gestión del código fuente y mecanismos de integración continua:

  1. El código debe estar subido a un repositorio de Github siguiendo Github flow

     https://github.com/fafagorg/products

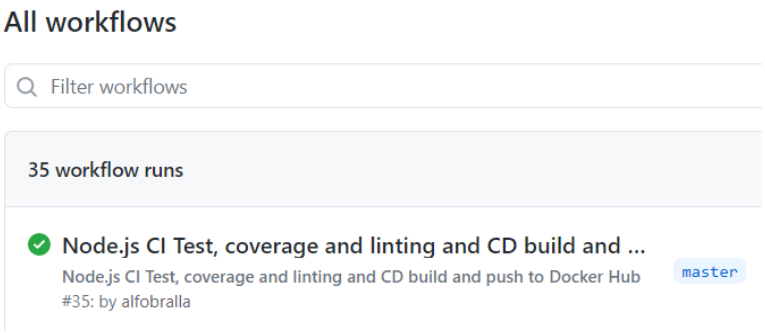
     Como podemos apreciar en la imagen siguiente, hemos utilizado un total de nueve ramas de desarrollo distintas, que se corresponden a groso modo con las distintas funcionalidades y secciones que componen nuestro microservicio. De esta manera, hemos separado en esas nueve ramas independientes los desarrollos correspondientes a cada funcionalidad, realizando las respectivas “Pull Request” para mergear nuestro código a la rama “Master”. Conforme hemos ido haciendo esto, hemos ido borrando las ramas para no mantener activas las ramas que ya no se usen.



2. El código debe compilarse y probarse automáticamente usando Travis.ci en cada commit

   https://github.com/fafagorg/products/actions

Usamos GitHub Actions para compilar y testear nuestro código de forma automática con cada push a nuestro repositorio. Como podemos apreciar en la imagen inferior, este proceso se lleva a cabo de forma satisfactoria.



- Debe haber definida una imagen Docker del proyecto

  https://hub.docker.com/repository/docker/fafagoproducts/productsapi

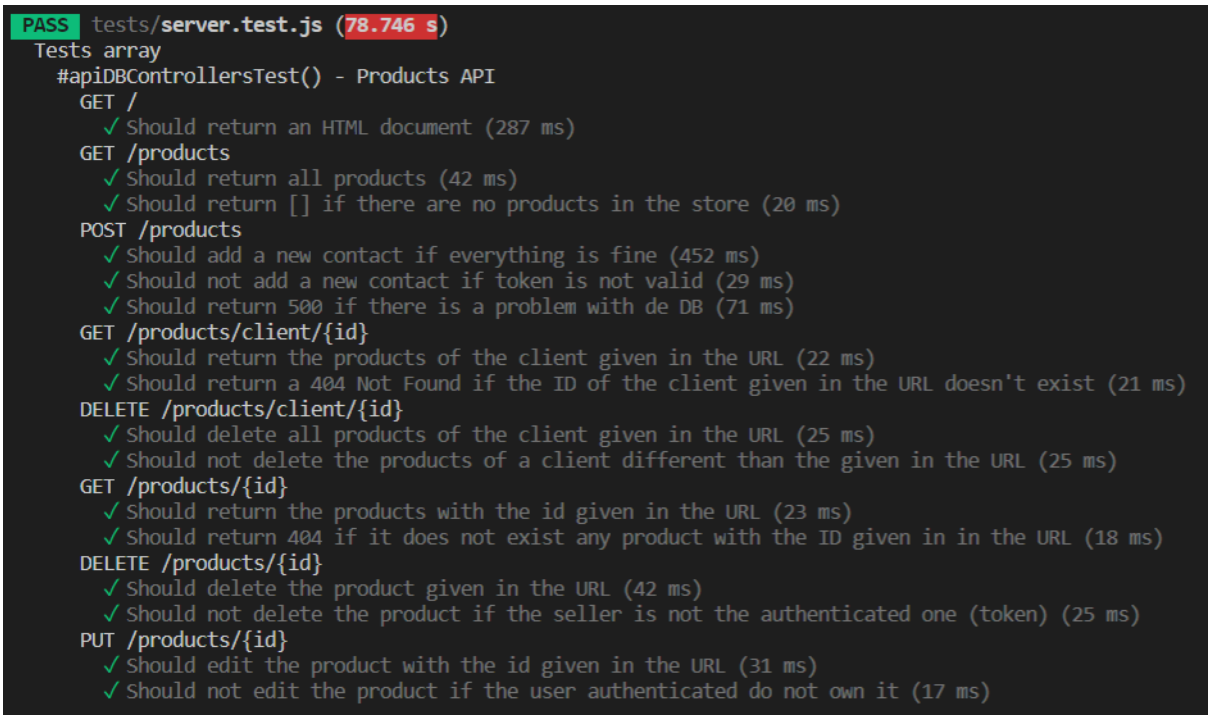
Hemos incluido un archivo Dockerfile en nuestro repositorio (https://github.com/fafagorg/products/blob/master/Dockerfile). De forma automática, gracias a la integración continua, con cada push se buildea una nueva imagen de Docker del mismo.

- Debe haber pruebas unitarias implementadas en Javascript para el código del backend utilizando Jest (el usado en los ejercicios) o Mocha y Chai o similar. Como norma general debe haber tests para todas las funciones no triviales de la aplicación. Probando tanto escenarios positivos como negativos.

  https://github.com/fafagorg/products/blob/master/tests/server.test.js

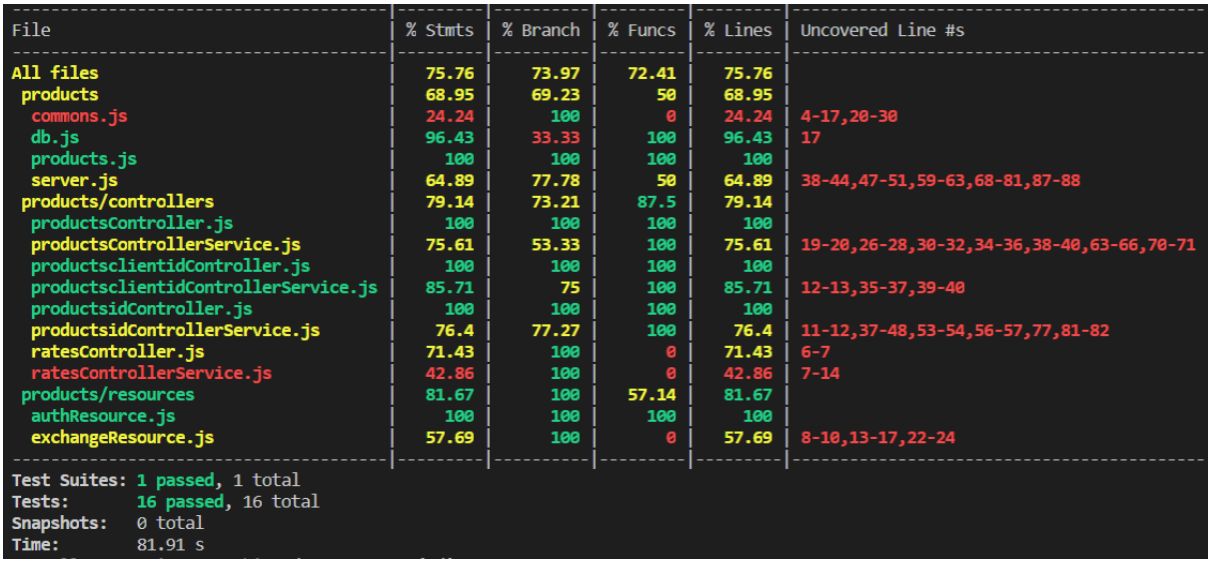
En este archivo de test, encontramos todas las funciones de test sobre los métodos de nuestra API. En todos los casos se comprueban los correspondientes casos positivos y negativos; por ejemplo, para un GET se comprueba que devuelva el elemento esperado y también se prueba que no encuentre el elemento (devolviendo un error 404-Not Found). Esto se repite con cada uno de los métodos de nuestra API y las diversas comprobaciones positivas y negativas posibles.

Como podemos apreciar en las capturas de pantalla inferiores, con el comando "npm test server.test.js" ejecutaríamos este conjunto de tests, obteniendo los resultados recogidos en dichas fotografías. Tal y como en ella se muestra, tenemos un total de 16 pruebas para todos los métodos de nuestro microservicio (distribuidos en los respectivos archivos "ControllerServices.js" de la carpeta "Controllers" de nuestro repositorio), indicando en estas breves descripciones lo que comprobamos con cada test.



Por otro lado, en la imagen inferior podemos comprobar la cobertura que tiene nuestro fichero de test sobre el total del código de nuestro microservicio. Como podemos apreciar en dicha fotografía, el citado porcentaje es alto, siendo especialmente importante la buena cobertura que tienen nuestros ficheros de mayor peso (los controladores de los métodos de nuestra API). Independientemente de esta cifra, usada únicamente como indicativo, con estos 16 tests probamos todas y cada una de las funcionalidades de nuestro

microservicio, tanto de forma positiva como negativa, siendo ésto mucho más importante que la cifra de coverage en sí.



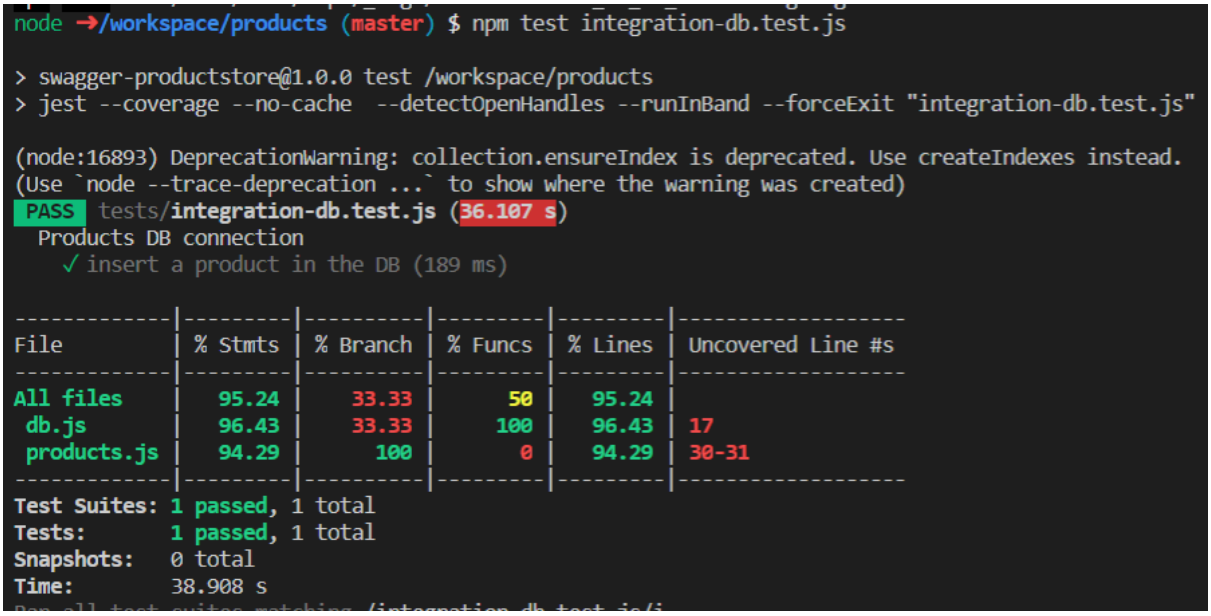
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	75.76	73.97	72.41	75.76	
products	68.95	69.23	50	68.95	
commons.js	24.24	100	0	24.24	4-17,20-30
db.js	96.43	33.33	100	96.43	17
products.js	100	100	100	100	
server.js	64.89	77.78	50	64.89	38-44,47-51,59-63,68-81,87-88
products/controllers	79.14	73.21	87.5	79.14	
productsController.js	100	100	100	100	
productsControllerService.js	75.61	53.33	100	75.61	19-20,26-28,30-32,34-36,38-40,63-66,70-71
productsclientidController.js	100	100	100	100	
productsclientidControllerService.js	85.71	75	100	85.71	12-13,35-37,39-40
productsidController.js	100	100	100	100	
productsidControllerService.js	76.4	77.27	100	76.4	11-12,37-48,53-54,56-57,77,81-82
ratesController.js	71.43	100	0	71.43	6-7
ratesControllerService.js	42.86	100	0	42.86	7-14
products/resources	81.67	100	57.14	81.67	
authResource.js	100	100	100	100	
exchangeResource.js	57.69	100	0	57.69	8-10,13-17,22-24

Test Suites: 1 passed, 1 total
Tests: 16 passed, 16 total
Snapshots: 0 total
Time: 81.91 s

- Debe haber pruebas de integración con la base de datos.

  https://github.com/fafagorg/products/blob/master/tests/integration-db.test.js

Siguiendo con el ejemplo propuesto en el vídeo de prácticas L06 - Step 3, en el fichero de test adjunto anteriormente probamos la conexión a nuestra base de datos, insertamos un nuevo producto, y por último comprobamos que se ha añadido correctamente (y que por tanto, el array de productos contiene un elemento tras la inserción anterior). Como podemos apreciar en la captura de pantalla inferior, con el comando “npm test integration-db.test.js”, la citada prueba asegura y confirma que funciona correctamente.



- Debe tener un mecanismo de autenticación en la API

Ver requisito “Implementación de un mecanismo de autenticación más completo” del microservicio avanzado que gestione un recurso.

Microservicio avanzado que gestione un recurso:

- Implementar un frontend con rutas y navegación

  http://frontend.fafago-dev.alexgd.es/ (Home)

Hemos implementado un frontend común, tal y como queda explicado en el requisito de la aplicación avanzada correspondiente. Hemos utilizado REACT-Route para manejar el enrutamiento entre las distintas vistas que lo componen y poder navegar por ellas. Nuestro microservicio de productos hace uso de una manera más particular de las siguientes vistas:

- http://frontend.fafago-dev.alexgd.es/search (Búsqueda de productos por parámetros - GET a /products, pudiendo aplicar parámetros de búsqueda).

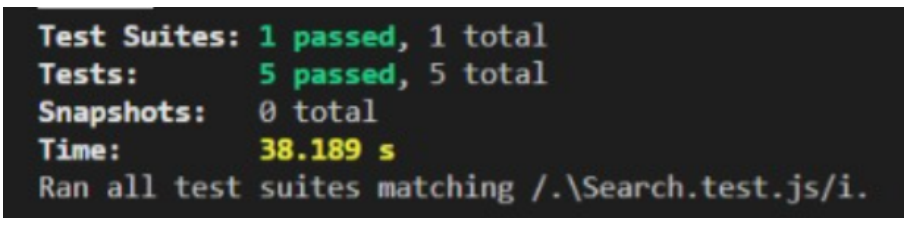
- http://frontend.fafago-dev.alexgd.es/product?id=1 (Vista de un producto - GET a /products/id, desde donde podemos borrar (DELETE a /product/id) o editar (PUT a /product/id), siempre y cuando hayamos iniciado sesión con el usuario que posea los productos). Desde esta vista también podemos acceder a las reviews de un producto en particular (GET a /reviews/product/id).

- http://frontend.fafago-dev.alexgd.es/product_client?username=alfpab
  Vista de los productos de un vendedor, desde donde podemos listarlos (GET a /products/client/id), editarlos (PUT a /products/id), añadir uno nuevo (POST a /products) o borrarlo (DELETE a /products). Estas tres últimas operaciones sólo estarán permitidas si estamos logueados con la cuenta del vendedor que posee dichos productos.

- Implementación de pruebas en la interfaz de usuario

  https://github.com/fafagorg/frontend/blob/develop/src/pages/products/Search.test.js

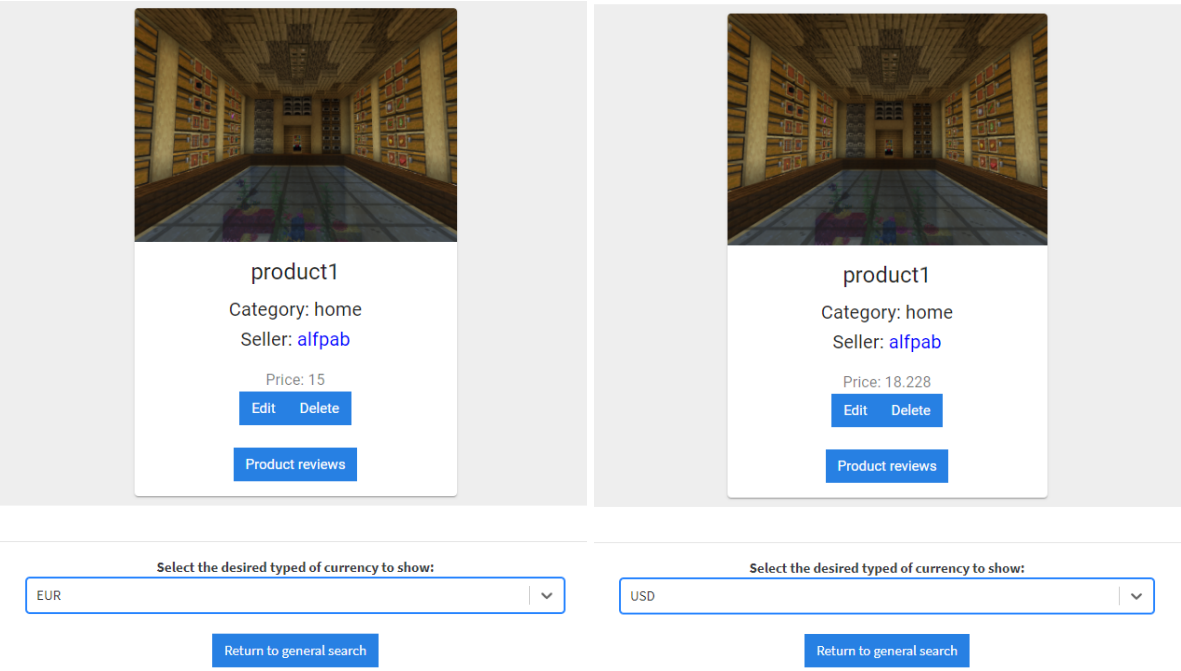
Siguiendo las indicaciones del vídeo L06 - Step 4, hemos utilizado el fichero enlazado anteriormente para testear las funcionalidades básicas de la interfaz de usuario. De esta manera, en dicho archivo comprobamos el correcto funcionamiento de alertas, botones, acciones y renderización de componentes, tal y como podemos apreciar su evaluación satisfactoria en la imagen inferior, para las cinco diferentes pruebas que incluimos para ello.



- Consumo de algún API externa (distinta de las de los grupos de práctica).

  https://api.exchangeratesapi.io/latest

Utilizamos la API enlazada anteriormente para poder aplicar un cambio de divisas. De esta forma, hacemos un GET a dicha API, y en nuestro frontend podemos seleccionar la moneda que deseemos. Así, automáticamente una vez cambiemos de divisa, el precio de cada producto será mostrado en el valor de dicha moneda (multiplicando el precio del producto almacenado en nuestra base de datos, en euros, por su base equivalente en la otra moneda). En el ejemplo de las imágenes inferiores, vemos el cambio de visualizar el precio en euros (precio original) y su posterior valor en dólares (a la derecha)



- Uso de redux como forma de gestionar el estado de los componentes de React en el frontend.

El token con el que el microservicio de autenticación gestiona la sesión de los usuarios hace uso de redux. Por este motivo, para acceder al token y comprobar la autenticación de cada usuario, estaremos haciendo uso de redux en nuestro frontend. Esto sucede tanto cuando necesitamos hacer un post, un put o un delete a un producto, ya que estas son las acciones que requieren de inicio de sesión (ya que es necesario comprobar que el usuario que va a borrar, editar o añadir un producto es quien dice ser mediante la validación de su token).

- Añadir validación no trivial a los formularios del frontend.

  https://github.com/fafagorg/frontend/blob/develop/src/components/product/products.js

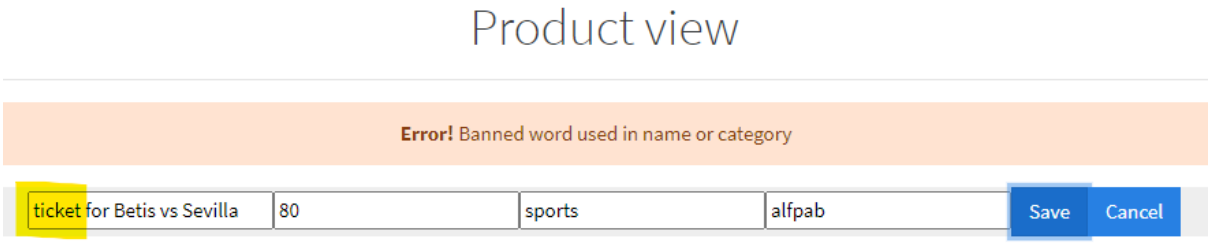
Dado que nuestra aplicación está orientada a la compra y venta de productos, hemos estimado oportuno incluir sendas comprobaciones en los campos textuales para

asegurar un buen uso de nuestro microservicio. Así, siguiendo el ejemplo de plataformas como Wallapop, hemos decidido incluir una lista de palabras baneadas (ver “BannedWords” en la imagen inferior). De esta forma, hemos intentado de evitar la reventa de entradas de eventos (prohibiendo el uso de la palabra “ticket”), el spam, los intercambios (con la palabra “exchange”, ya que nuestros productos tienen precio fijo, no queremos permitir trueques),el uso de nuestra aplicación para citas (excluyendo las palabras “date”, “dating”, “dm”) y el uso de “WhatsApp” para comunicación con un vendedor, en lugar del servicio de mensajería con el que cuenta nuestra aplicación.

```
class Products extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            errorInfo: null,
            products: [],
            isEditing: {},
            userId: '',
            exchangeRates: [],
            currentRate: {label:"EUR",value: 1},
            token: props.token,
            bannedWords: ['ticket','spam','exchange','date','dating','dm','whatsapp']

        };
        this.handleCloseError = this.handleCloseError.bind(this);
        this.addProduct = this.addProduct.bind(this);
        this.filterProduct = this.filterProduct.bind(this);
    }
```

Todas las palabras anteriormente citadas serán automáticamente detectadas en los campos de texto de nuestros formularios (nombre del producto y categoría), de forma que no permita su uso, tal y como mostramos en el ejemplo de la imagen siguiente, en la que tratamos de actualizar un producto e introducir la palabra “ticket” en su nombre, para revender una entrada de un partido de fútbol.



- Uso de mongoose para validar los objetos antes de almacenarlos en la base de datos.

https://github.com/fafagorg/products/blob/ci/products.js

En el fichero anteriormente enlazado podemos ver el modelo del elemento “product”, que es con el que trabajamos en todo momento en nuestro microservicio. Como se aprecia en él, tenemos cinco parámetros: nombre del producto, categoría en la que se clasifica, precio (en euros), vendedor (userId con el que está registrado en la aplicación) e identificador (id) del producto.

Tanto el nombre como el id se han declarado como únicos, para no contar con productos repetidos. En ambos casos, si intentamos introducir un producto con un nombre o un id que ya exista nos dará error. Esto último sólo sería posible desde el backend, ya que en el frontend no permitimos crear ni modificar el id (se genera calculándolo automáticamente).

Todos los atributos son requeridos, ya que no tendría sentido insertar un producto sin ninguno de estos parámetros. En caso de que faltara alguno de ellos, al igual que en caso de repetición de nombre o id (como hemos explicado anteriormente), no dejaría insertar ni editar dicho producto.

- Tener el API REST documentado con swagger.

  http://products.fafago-dev.alexgd.es/docs/

La API fue generada con la herramienta OAS Generator (https://www.npmjs.com/package/oas-generator), por lo que al definir los endpoints y sus respectivos parámetros, su documentación con Swagger fue generada automáticamente, con solo realizar leves modificaciones sobre el fichero “oas-doc.yaml”. De esta manera, en el enlace adjunto anteriormente, podemos encontrar la documentación de nuestra API, sus endpoints, modelos, parámetros y respuestas esperadas, así como una breve descripción de la utilidad y uso de cada operación en el contexto de nuestra aplicación (campo “description” de cada método.

- Implementación de un mecanismo de autenticación más completo.

Hemos usado el mecanismo de autenticación JWT, tecnología de token de cliente, que nos ha proporcionado el microservicio de autenticación de nuestra aplicación. Este mecanismo de autenticación ha sido requerido por nuestro microservicio de productos para restringir las operaciones no seguras contra nuestra API (POST; PUT y DELETE), validando previamente que el usuario que desea realizar alguna de dichas operaciones es realmente quien dice ser. De esta forma, no podremos añadir productos en nombre de otro usuario, ni editar ni borrar los productos de alguien que no seamos nosotros mismos (con nuestra sesión debidamente iniciada y validada por el microservicio de autenticación, al que le pasamos el token de la sesión y nos debe devolver un 200-OK).

Aplicación basada en microservicios avanzada:

Al tratarse de requisitos implementados de forma común y con una misma justificación de uso para cada uno de los cuatro microservicios, hemos decidido recoger los requisitos de la aplicación en un documento común a los cuatro subgrupos, a fin de no repetir las mismas justificaciones en cada una de las memorias por separado. Este documento se encuentra accesible en el siguiente enlace:

https://github.com/fafagorg/documents/blob/master/Requisitos%20de%20la%20aplicaci%C3%B3n%20basada%20en%20microservicios%20avanzada.pdf