

Universidad de Sevilla



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

**FUNDAMENTOS DE INGENIERÍA SOFTWARE PARA
CLOUD**

Realizado por

JESÚS MONDA CAÑA

MCARMEN ARENAS ZAYAS

Índice

| | |
|----------------------------------|----------|
| Introducción | 3 |
| Nivel | 3 |
| Justificación | 4 |
| Análisis de los esfuerzos | 7 |

Introducción

En este documento vamos a plasmar en términos generales los detalles del microservicio que hemos desarrollado para la asignatura.

El microservicio en cuestión lo hemos nombrado como, **Messenger**, y consiste en un chat. El cual se utilizará como medio para que entren en contacto 2 usuarios en torno a un producto concreto.

Nuestro microservicio forma parte del proyecto Fafago, el cual engloba a 3 microservicios más: auth, reviews y product. De estos 3 mencionados, Messenger consume de los microservicios de auth y product.

El backend está constituido por una API REST en node con express, además de socket.io para la comunicación en tiempo real. Para la base de datos se ha usado Redis (base de datos no relacional).

El frontend se ha realizado en react, y se ha integrado con los frontend de los demás microservicios constituyendo un frontend común para todos.

Todo el código del backend se puede encontrar en el siguiente repositorio de GitHub: <https://github.com/fafagorg/messenger>. Y el repositorio con el frontend común se puede encontrar en: <https://github.com/fafagorg/frontend>

Nivel

En este apartado expresaremos el nivel al que hemos decidido presentarnos, aunque cabe destacar que al ser nuestro microservicio un chat, nos ha resultado difícil encontrar un número suficiente de requisitos que cumplir para llegar al nivel más alto. Por ello hemos tenido que añadir algunos requisitos propios para tener el número requerido.

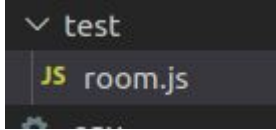
Como grupo nos presentamos al nivel más alto (nivel hasta 10 puntos) y como pareja nos presentamos al nivel hasta 10 puntos también.

Justificación

En este apartado explicaremos cómo hemos conseguido realizar cada requisito del apartado de microservicio básico y avanzado propuestos en la documentación del proyecto de la asignatura.

En la tabla que se mostrará a continuación se podrán ver el conjunto de los requisitos anteriormente mencionados que hemos implementado:

| Nivel Básico | |
|--|--|
| Requisito | Descripción |
| 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. | Los endpoint realizados ha sido: 1. GET /v1/messenger/room 2. GET /v1/messenger/room/{roomId} 3. PUT /v1/messenger/room/{roomId} 4. DELETE /v1/messenger/room/{roomId} No tenemos POST ya que de esto se encarga el socket |
| 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). | Se ha realizado un frontend común con todos los demás microservicios utilizando Redux y React. |
| Debe estar desplegado en la nube y ser accesible en una URL. | Se ha desplegado en Google Cloud utilizando el servicio GKE(Google Kubernetes Engine) y CloudBuild(CI/CD). messenger.fafago-dev.alexgd.es messenger.fafago-pre.alexgd.es messenger.fafago-pro.alexgd.es frontend.fafago-dev.alexgd.es frontend.fafago-pre.alexgd.es frontend.fafago-pro.alexgd.es |
| La API que gestione el recurso también debe ser accesible en una dirección bien versionada. | Podemos observar en el apartado 1 como los endpoint están versionadas. |
| Se debe tener una documentación de todas las operaciones de la API incluyendo las posibles peticiones y las respuestas recibidas. | La api ha sido documentada utilizando swagger, podemos acceder a su documentación mediante: http://messenger.fafago-dev.alexgd.es/docs http://messenger.fafago-pre.alexgd.es/docs http://messenger.fafago-pro.alexgd.es/docs |

| | |
|---|---|
| Debe tener persistencia utilizando MongoDB | <p>En nuestro caso para conseguir la persistencia de datos hemos optado por un usar un volumen de kubernetes. Así cuando se redesplice no se perderán los datos almacenados en la redis.</p> <p>En local utilizamos un docker-compose con un volumen.</p> |
| El código debe estar subido a un repositorio de Github siguiendo Github flow | <p>Se ha creado un repositorio con la rama develop, master, release y features, dentro de una organización común para todas las parejas del equipo.</p> <p>https://github.com/fafagorg/messenger</p> |
| El código debe compilarse y probarse automáticamente usando Travis.ci en cada commit | Para la integración continua, en lugar de usar travis, hemos optado por usar google cloud build, que es igualmente válido y autogestionado. |
| Debe haber definida una imagen Docker del proyecto | Se ha definido un Dockerfile y un docker-composed para la creación de las imágenes de redis y express. Y además, se ha realizado un segundo Dockerfile para producción en el que no se define el paquete nodemon. |
| 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 | <p>Se han implementado test unitarios en mocha, chai ynock.</p> <p>Estos test nos han ayudado a poder testear las funcionalidades incluso aunque los otros microservicios estén caídos (nock).</p> <p>Se pueden ver dentro de la carpeta test, en el archivo room.js :</p>  |
| Debe haber pruebas de integración con la base de datos | Se han realizado pruebas con la base de datos en los test |
| Debe tener un mecanismo de autenticación en la API | Complementa al avanzado |
| Nivel avanzado | |
| Requisito | Descripción |

| | |
|---|--|
| Implementar un frontend con rutas y navegación | <p>Se ha trabajado sobre un fronted en react común con todos los microservicios, el cual consta de rutas y navegación.</p> <p>frontend.fafago-dev.alexgd.es frontend.fafago-pre.alexgd.es frontend.fafago-pro.alexgd.es</p> |
| Implementación de cachés o algún mecanismo para optimizar el acceso a datos de otros recursos | Se ha implementado un mecanismo de caché empleando redis, en el que se guardan las peticiones de los endpoints (GET) con código 200. |
| Uso de redux como forma de gestionar el estado de los componentes de React en el frontend | Se emplea para gestionar el token. |
| Tener el API REST documentado con swagger | <p>Como ya se ha comentado anteriormente, se ha realizado una documentación en swagger:</p> <p>http://messenger.fafago-dev.alexgd.es/docs http://messenger.fafago-pre.alexgd.es/docs http://messenger.fafago-pro.alexgd.es/docs</p> |
| Implementación de un mecanismo de autenticación más completo | Se ha empleado la autenticación del microservicio de auth |
| Tener versionado cada imagen docker | Se ha implementado en todas las imágenes docker, tanto las imágenes de dev, pre y pro. De esta manera podemos volver a una versión concreta |
| CI/CD en Kubernetes | Se ha integrado despliegue continuo en kubernetes para que se despliegue con cada push en la rama main, release y develop |
| Utilización socket.io | Implementación de socket.io para la comunicación en tiempo real, entre el navegador y el servidor. |

Como se puede ver, las 3 últimas filas de la tabla, se han marcado con un color gris claro, esto se debe a que son requisitos que hemos añadido al margen de los ya marcados en la documentación de los requisitos de proyecto proporcionados por la asignatura. Estos son los que se mencionan en el apartado de nivel.

Análisis de los esfuerzos

Para contabilizar las horas empleadas en la asignatura decidimos utilizar la herramienta "Clockify", ya que la habíamos empleado anteriormente en otros proyectos con buenos resultados.

Teniendo en cuenta que cada día se ha invertido una media aproximada de 1 hora por parte de cada integrante de la pareja durante 46 días (9/12/2020 - 22/01/2021), sale un total de 46.4 horas en MCarmen y 45.2 para Jesús