

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

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

Repositorio frontend: <https://github.com/fafagorg/frontend>

Frontend: <http://frontend.fafago-pro.alexgd.es>

Backend: <http://messenger.fafago-pro.alexgd.es>

Integración continua comun (has sido invitado para poder ver la página):
<https://console.cloud.google.com/cloud-build/builds?authuser=1&project=fis-us>

Grupo 5

Fafago

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 fronted se ha realizado en react, y se ha integrado con los fronted 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 fronted 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.

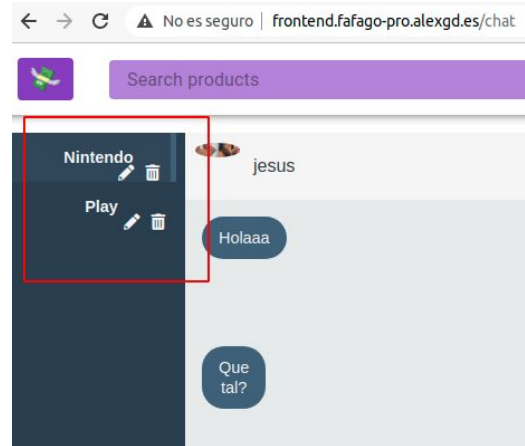
Como grupo nos presentamos al nivel más alto (nivel hasta 10 puntos) y como pareja nos presentamos al nivel hasta 10 puntos también.

Justificación

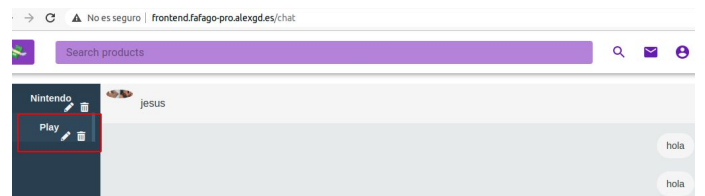
En este apartado explicaremos cómo hemos conseguido realizar cada requisito del apartado de microservicio básico y avanzado propuestos en la documentación del proyecto de la asignatura.

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

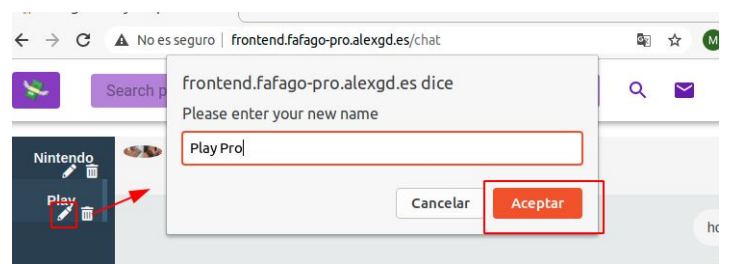
Nivel Básico	
Requisito	Descripción
El backend debe ser una API REST tal como se ha visto en clase implementando al menos los métodos GET, POST, PUT y DELETE y devolviendo un conjunto de códigos de estado adecuado.	<p>Los endpoint realizados ha sido:</p> <ol style="list-style-type: none">1. GET /v1/messenger/room2. GET /v1/messenger/room/{roomId}3. PUT /v1/messenger/room/{roomId}4. DELETE /v1/messenger/room/{roomId} <p>No tenemos método POST, ya que al ser nuestro microservicio un chat, empleamos comunicación en tiempo real mediante socket.io. Es el socket el que se encarga de recibir y enviar mensajes. Cuando un usuario se conecta al chat (socket) y manda un mensaje, este se crea en ese momento, y además si no existía una conversación previa entre los 2 usuarios, en ese momento también se crea la conversación (sala).</p> <p>Es por esto, por lo que el socket es el encargado de realizar POST en este microservicio.</p>
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).	<p>Se ha realizado un frontend común con todos los demás microservicios utilizando Redux y React. Al cual se puede acceder mediante esta URL: http://frontend.fafago-pro.alexgd.es</p> <p>Dentro de este frontend nuestro microservicio ocupa las vistas:</p> <ul style="list-style-type: none">- http://frontend.fafago-pro.alexgd.es/chat, en la que se podrá acceder desde el menú principal haciendo click en el icono de mensajes:  <p>Y en la que se podrá visualizar un listado con todas las conversaciones que tenga asociadas el usuario logueado.</p>

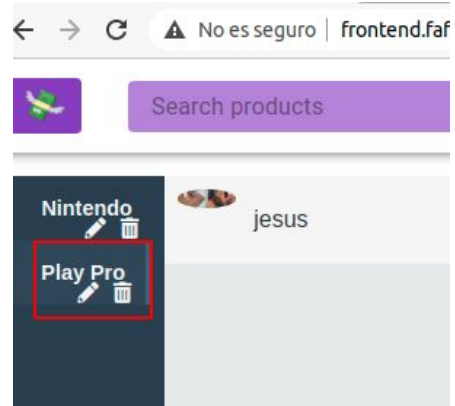


- <http://frontend.fafago-pro.alexgd.es/chat?roomId=XXX>, que te permite hablar en una sala concreta. Es la url que sale cuando se inicia una conversación por primera vez (momento de su creación, al enviar un mensaje). Después al seleccionar las conversaciones en el listado la url será la de /chat.

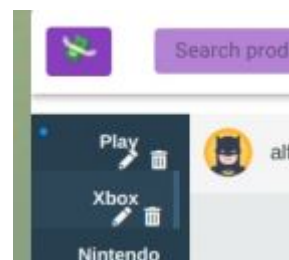


- Al hacer click en el botón de editar, nos aparece una alerta para poder renombrar el nombre de la conversación, este renombramiento no será aplicado al otro usuario.

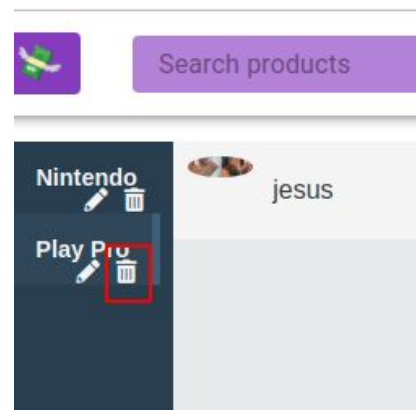




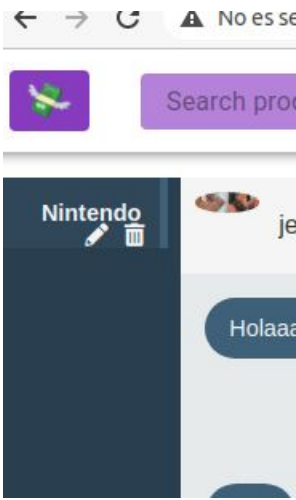
- Cuando recibes un mensaje y no estás metido en la conversación aparecerá un icono azul indicando que tienes mensajes pendiente.

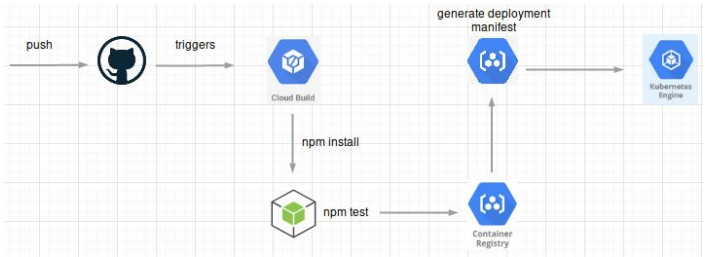
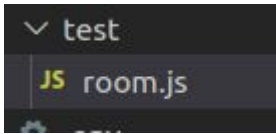


- Cuando haces click en el icono de la papelera, se procede a borrar la conversación y sus mensajes, pero únicamente para el usuario logueado



Una vez pulsado el botón se borra del listado:

	
<p>Debe estar desplegado en la nube y ser accesible en una URL.</p>	<p>Se ha desplegado en Google Cloud utilizando el servicio GKE(Google Kubernetes Engine) y CloudBuild(CI/CD). Para realizar este despliegue se ha creado un deployments, un service interno, un ingress y un persistent volumen para persistir los datos de Redis. Estos elementos están almacenados en el repositorio backend, en la carpeta k8s.</p> <p> messenger.fafago-dev.alexgd.es messenger.fafago-pre.alexgd.es messenger.fafago-pro.alexgd.es </p> <p> frontend.fafago-dev.alexgd.es frontend.fafago-pre.alexgd.es frontend.fafago-pro.alexgd.es </p>
<p>La API que gestione el recurso también debe ser accesible en una dirección bien versionada.</p>	<p>Podemos observar en el apartado 1 como los endpoint están versionados, ahora mismo tenemos la versión v1.</p>
<p>Se debe tener una documentación de todas las operaciones de la API incluyendo las posibles peticiones y las respuestas recibidas.</p>	<p>La api ha sido documentada utilizando swagger, podemos acceder a su documentación mediante:</p> <p> http://messenger.fafago-dev.alexgd.es/docs http://messenger.fafago-pre.alexgd.es/docs http://messenger.fafago-pro.alexgd.es/docs </p>
<p>Debe tener persistencia utilizando MongoDB</p>	<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>

<p>El código debe estar subido a un repositorio de Github siguiendo Github flow</p>	<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>
<p>El código debe compilarse y probarse automáticamente usando Travis.ci en cada commit</p>	<p>Para la integración continua, en lugar de usar travis, hemos optado por usar Google Cloud Build, que es igualmente válido y autogestionado, de esta manera no te tienes que preocupar por el servidor que lo gestiona..</p> <p>Cada vez que se realiza un commit a la rama de develop, master o release, se despliega automáticamente.</p>  <pre> graph LR push --> GitHub GitHub -- triggers --> CloudBuild[Cloud Build] CloudBuild -- "npm install" --> npmInstall[npm install] npmInstall --> npmTest[npm test] npmTest --> ContainerRegistry[Container Registry] ContainerRegistry -- "generate deployment manifest" --> KubernetesEngine[Kubernetes Engine] </pre>
<p>Debe haber definida una imagen Docker del proyecto</p>	<p>Se ha definido un Dockerfile y un docker-compose para levantar el servicio utilizando una imagen de Redis y express. Además, se ha realizado un segundo Dockerfile para producción en el que no se inicia el servidor con el comando nodemon, ya que este es únicamente para desarrollo.</p>
<p>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>	<p>Se han implementado test unitarios en mocha, chai y nokk.</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>  <p>Se ha testeado tanto la base de datos como el código interno de cada uno de los endpoint.</p>

Debe haber pruebas de integración con la base de datos	<p>Se han realizado pruebas con la base de datos en los test.</p> <p>Estos test están en el fichero test/room.js</p> <p>Se han creado salas y mensajes para posteriormente eliminarlos con los test que atacan a la api.</p>
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 frontend en react común con todos los microservicios, el cual consta de rutas y navegación. Como se ha detallado en el apartado referente al frontend del Nivel Básico.</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	<p>Se ha implementado un mecanismo de caché empleando Redis, este sistema se encarga de cachear todas las peticiones que devuelven un 200, una vez una petición ha sido cacheada no vuelve a ser cacheada a menos que un sistema la deshabilite.</p> <p>Esto puede ocurrir por ejemplo si cacheamos el listado de conversaciones y de repente nos llega un mensaje de una nueva conversación, la siguiente petición para obtener el listado de salas no será devuelto por la cache.</p>
Uso de redux como forma de gestionar el estado de los componentes de React en el frontend	Se emplea para gestionar el token en cada vista, de esta manera podemos obtener la información del usuario logueado.
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, para ello hacemos una petición al método /auth/validate del microservicio de auth.

	Este método devuelve la información del usuario si el token es válido
Tener versionado cada imagen docker	Se ha implementado en todas las imágenes docker, tanto las imágenes de dev, pre y pro, estas imágenes se suben a GCR, un sistema gestionado por Google Cloud. Es similar a Docker Hub. De esta manera podemos volver a una versión concreta para testearla o por si en nuestra versión actual de producción hay un error.
CI/CD en Kubernetes	<p>Se ha integrado un despliegue continuo en Kubernetes para que se despliegue con cada push en la rama main, release y develop. Esto está hecho en CloudBuild y utiliza el fichero cloudbuild-deploy.yaml</p> <p>El profesor ha sido invitado al proyecto de Google Cloud para que pueda ver el funcionamiento de este servicio, le ha llegado un email al correo manuel.resinas@gmail.com</p>
Utilización socket.io	Implementación de SocketIO para la comunicación en tiempo real, entre el navegador y el servidor. Se verifica la autenticación del usuario en la petición inicial de conexión, esta validación se hace gracias al microservicio de Auth

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 otro 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