Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Plataforma Web de contratación de servicios del hogar basada en una arquitectura orientada microservicios

Student: Miguel Pérez Febrero
Direction: Fernando Bellas Permuy

A Coruña, febreiro de 2021.

A mis padres, por el apoyo hasta en los momentos más difíciles

Agradecimientos

Gracias a todas las personas que estuvieron ahí, especialmente en los malos momentos. Gracias también a la FIC por hacer el camino más fácil.

Resumen

El objetivo de este proyecto es experimentar con las arquitecturas orientadas a microservicios. Para ello se implementará una plataforma web de contratación de servicios del hogar, que permitirá a el cliente registrarse, buscar servicios, solicitar su contratación y puntuarlos; mientras que el trabajador podrá ofrecer sus servicios y gestionar las solicitudes.

La arquitecutra a tratar consiste en un conjunto de pequeños servicios que recogen la lógica de negocio de la plataforma. Al tratarse de servicios pequeños, favorece la escalibidad y el mantenimiento del sistema.

El backend consiste en una nube de microservicios implementada en Java, con la ayuda de Spring Framework. Los datos serán guardados en MongoDB.

En cuanto al frontend consistirá en dos aplicaciones web SPA implementada usando JavaScript, apoyada en la librería React.

Abstract

The goal of this project is to experiment with microservices-oriented architectures. For this purpouse , a web platform for contracting home services will be implemented, that will allow the client to register, search for services, request their hiring and rate them;while the worker will be able to offer his services and manage the requests.

The architecture to be treated consists of a set of small services that collect the platform business logic. This small services favors scalability and system maintenance.

The backend consists of a cloud of microservices implemented in Java, with the Spring Framework help. The data will be saved in MongoDB.

As for the frontend, it will consist of two SPA web applications implemented using JavaScript, with React library.

Palabras clave:

- SPA
- Microservicios
- Java
- Spring
- JavaScript
- React

Keywords:

- SPA
- Microservices
- Java
- Spring
- JavaScript
- React

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introdución

Internet lleva años ganando terreno a las tiendas físicas, esto se vio acrecentado por la llegada del SARS-CoV-2. Los hábitos de consumo están cambiando hacia las tiendas online por varios motivos; no hay colas, mayor stock, no hay necesidad de desplazarse a la tienda física y, sobre todo, precios más competitivos. Sin embargo, hay sectores que se resisten a este cambio de paradigma debido a que no es posible realizar sus trabajos de manera online, como por ejemplo los servicios del hogar.

Actualmente, casi todos tenemos al alcance de nuestra mano un móvil con conexión a Internet. Tardamos pocos segundos en buscar empresas o personas que nos puedan ofrecer un servicio de hogar (un trabajo de pintura, limpieza, etc.), pero pocas veces podemos saber si la calidad de ese servicio será buena. Para solventar este problema, se creará una plataforma en la cual un trabajador podrá ofrecerse para un determinado servicio y un usuario podrá encontrar a trabajadores que realicen el servicio que les fuera de interés.

Los usuarios podrán puntuar a los trabajadores, lo que disminuirá la incertidumbre al contratar a alguien, por lo tanto, esta plataforma servirá de escaparate online a los profesionales.

1.1 Objetivos

El objetivo principal de este proyecto es crear una plataforma web basada en una arquitectura de microservicios. El foco de este trabajo será estudiar y trabajar sobre este tipo de arquitecturas. Para experimentar con la arquitectura, se construirá una plataforma web que consiste en la contratación de servicios del hogar.

Las arquitecturas basadas en microservicios se componen por pequeños servicios independientes que recogen la lógica de negocio y se comunican a través de APIs. Esas piezas son denominadas microservicios. La ventaja principal de esta arquitectura es la escalabilidad, debido a que los microservicios se pueden replicar de manera independiente en función de las necesidades del sistema. Además se ve reducida la complejidad del mantenimiento gracias a

que los servicios son mucho más pequeños y no acumulan grandes cantidades de código.

Desde el punto de vista funcional la plataforma constará de dos aplicaciones web diferentes, una para los trabajadores y otra para los usuarios. El objetivo de esta plataforma es comunicar a trabajadores que quieran ofrecer un servicio con clientes que quieran contratarlo. Por un lado, los trabajadores desde su aplicación podrán registrarse y ofrecer uno o varios tipos de servicios del hogar (como fontanería, jardinería, etc.). Por otro lado, los usuarios podrán registrarse, buscar y solicitar la realización de un servicio.

El administrador se encargaría de gestionar los tipos de servicios que se ofrecen en la plataforma y gestionar los trabajadores, para evitar fraudes y expulsar a los que no son competentes. Para acotar el proyecto al tamaño de un TFG, el administrador queda fuera del alcance de este.

Desde el punto de vista tecnológico el backend será una nube de microservicios y el frontend consistirá en dos aplicaciones web SPA.

1.2 Visión global del sistema

Como se menciona en el apartado anterior el backend será una nube de microservicios implementada en Java, con la ayuda de Spring Framework. Para persistir los datos del sistema, se hará uso de una base datos NoSql, MongoDB.

En cuanto al frontend, consta de dos aplicaciones web SPA que están implementadas usando JavasScript, en concreto con la ayuda de la librería React.

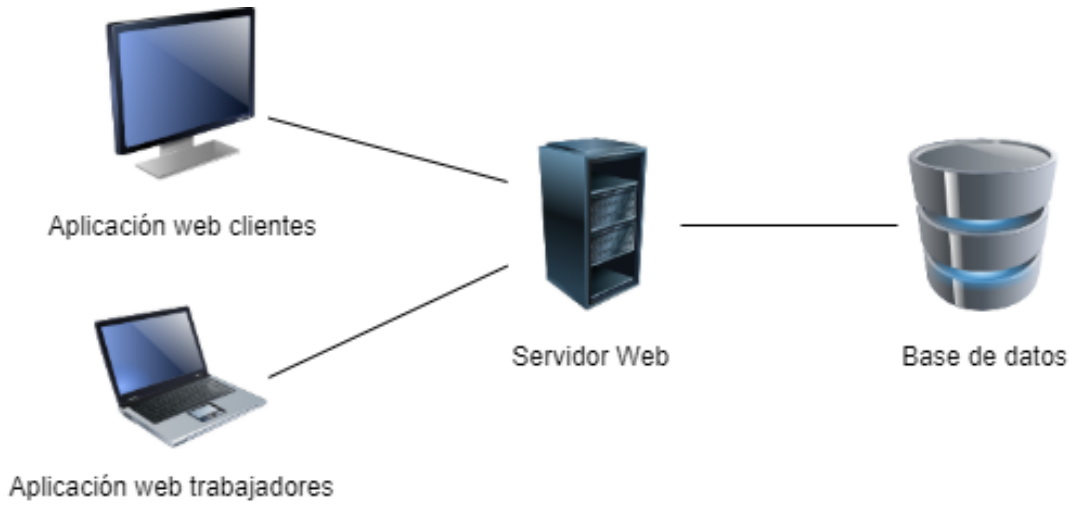


Figura 1.1: Visión global monolítico

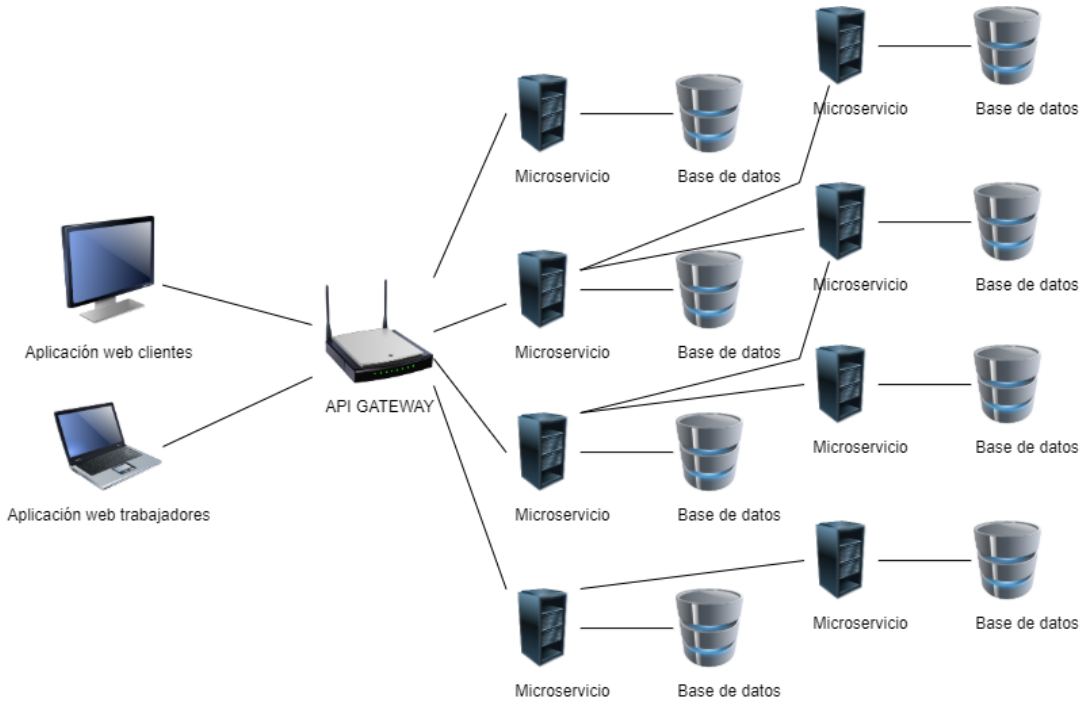


Figura 1.2: Visión global microservicios

Capítulo 2

Arquitecturas basadas en microservicios

2.1 Arquitectura monolítica

Las arquitecturas monolíticas son las más comunes. Son aquellas en las que el software se estructura en un único programa. Todos los aspectos quedan acoplados (interfaz de usuario, acceso a datos y lógica de negocio) y sujetos a este. Los problemas y ventajas principales que presenta una arquitectura monolítica son los siguientes:

2.1.1 Inconvenientes

- Un punto de fallo, significa un fallo en el sistema entero.
- Difícil de escalar a nivel de infraestructura. A la hora de escalar el servicio en función de la carga, es necesario replicar todo el servicio, en lugar de poder replicar cada una de sus partes de forma independiente según las necesidades de escalabilidad de esa parte.
- Mayor complejidad para el desarrollador. Se necesita un conocimiento amplio de todo el sistema. Cuanto más grande es el sistema mayor complejidad.
- Difícil de mantener.
- Difícil de integrar con otras aplicaciones.

2.1.2 Ventajas

- Es la manera natural de desarrollar (crear un proyecto e ir añadiendo funcionalidades).
- Eficiencia, la latencia interna de las aplicaciones suele ser baja.

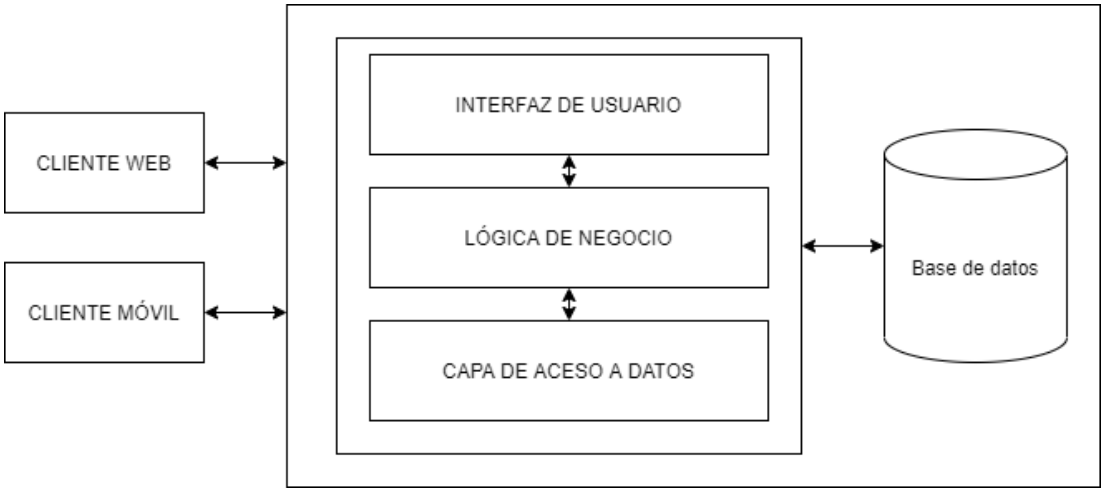


Figura 2.1: Arquitectura monolítica

2.2 Arquitectura basada en microservicios

Las arquitecturas basadas en microservicios [1] consiste en una aplicación que está compuesta por un conjunto de servicios pequeños, que recogen la lógica de negocio. Estos pequeños servicios, independientes entre sí, se ejecutan en diferentes procesos y cada uno se encarga de un pequeño conjunto de funcionalidades relacionadas. Al ser procesos diferentes, pueden ser desplegados de manera independiente e incluso usar diferentes lenguajes de programación y diferentes tecnologías de almacenamiento de datos.

2.2.1 Ventajas

- Si falla un microservicio, no falla todo el sistema.
- Es fácilmente escalable. Los microservicios se podrán replicar de manera independiente en función de las necesidades de escalabilidad del sistema.
- Menor complejidad para el desarrollador, debido a que "N" microservicios se implementan en "N" proyectos diferentes.
- Código más mantenible y más amigable con el cambio.
- Permiten emplear la mejor tecnología para cada problema.

2.2.2 Inconvenientes

- Aumenta la complejidad del sistema.
- Aumenta la latencia al haber comunicaciones entre microservicios.

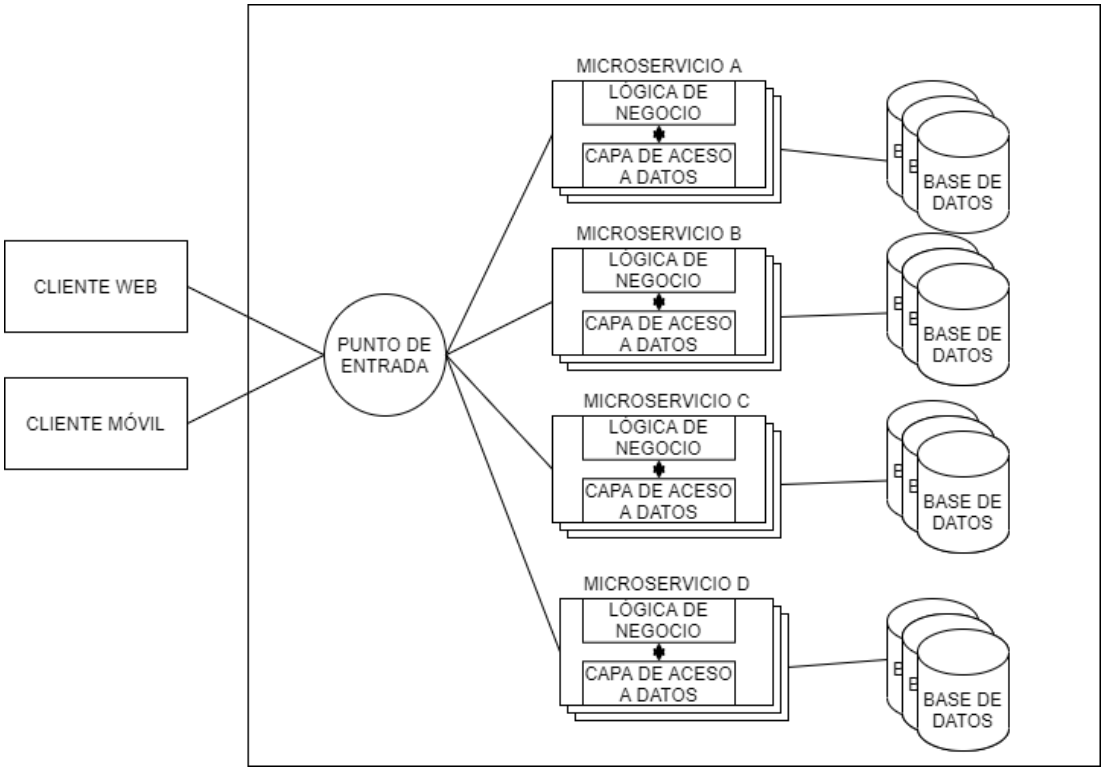


Figura 2.2: Arquitectura basada en microservicios

2.3 Monolito vs microservicios

Una vez vistos los rasgos principales de cada arquitectura es relevante abordar en qué situación priorizar una sobre la otra dependiendo de varios factores:

Por un lado, las arquitecturas monolíticas son buenas en entornos muy poco cambiantes, que no tengan necesidad de escalar o simplemente sean usadas por pocos usuarios. Por otro lado, las arquitecturas basadas en microservicios son buenas en entornos cambiantes o no muy bien definidos, que se espere una gran cantidad de usuarios o picos muy grandes de carga. Para ejemplificar, podríamos hacer referencia a la festividad black friday que ocurre una vez al año en los comercios electrónicos.

Por último, otro aspecto a tener en cuenta es el equipo disponible para realizar el proyecto, puesto que, no todos los equipos pueden asumir la implementación de una arquitectura basada en microservicios.

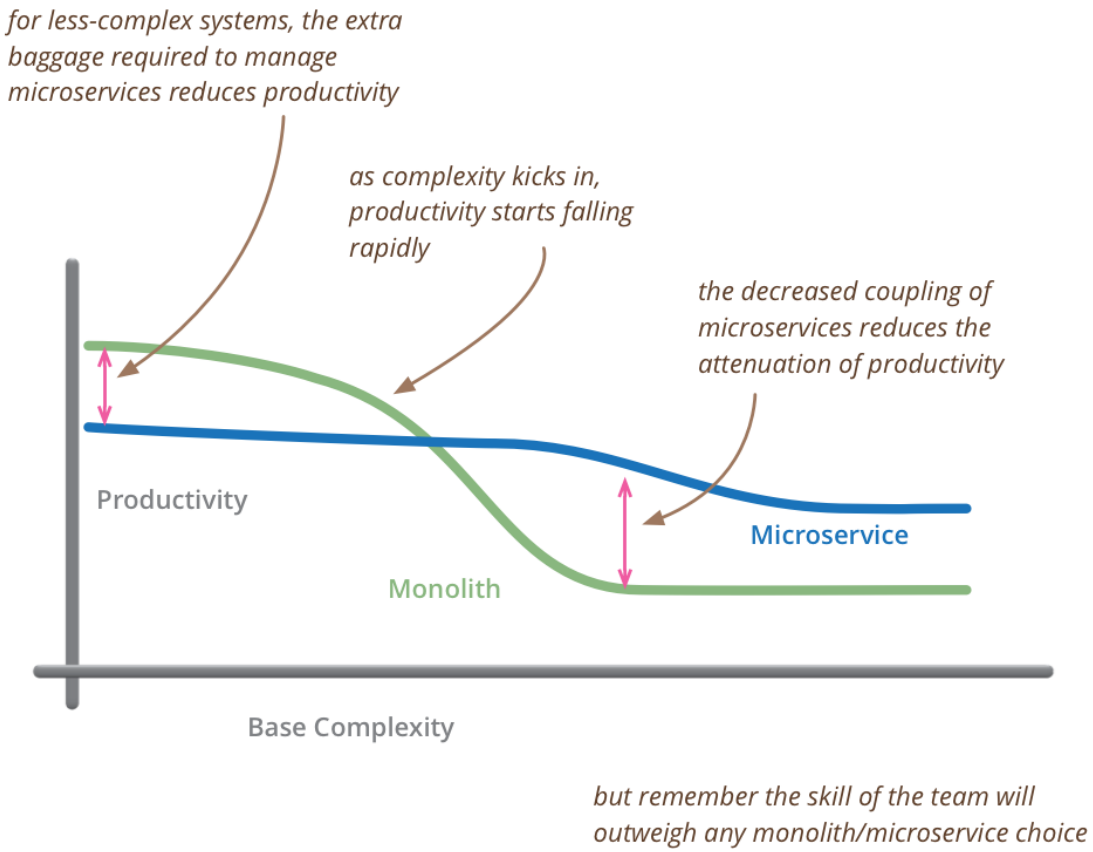


Figura 2.3: Diagrama de productividad de Martin Fowler

2.4 ¿Que es un microservicio?

El objetivo principal de un microservicio es encargarse de manera óptima de un conjunto de funcionalidades dentro de un dominio del sistema (por ejemplo, un microservicio de un sistema contable podría encargarse de las facturas). Idealmente, cada microservicio debería tener su base de datos privada y exponer un API para interactuar con el.

2.4.1 Microservicio Core

Un microservicio core, es un microservicio básico. Se encarga única y exclusivamente de su conjunto de funcionalidades y exponerlas a través de un API.

2.4.2 Microservicio de composición

Un microservicio de composición es más complejo. Este servicio tiene como funcionalidad operaciones que dependen de más de un microservicio. Este será el encargado de hacerse con

los datos de los microservicios que necesite y realizar sus operaciones, además de exponer en un API su funcionalidad. Un ejemplo claro de lo que sería un microservicio de composición es una compra; ya que necesitaría los datos del usuario que realizó la compra y el producto (o productos) que ha comprado.

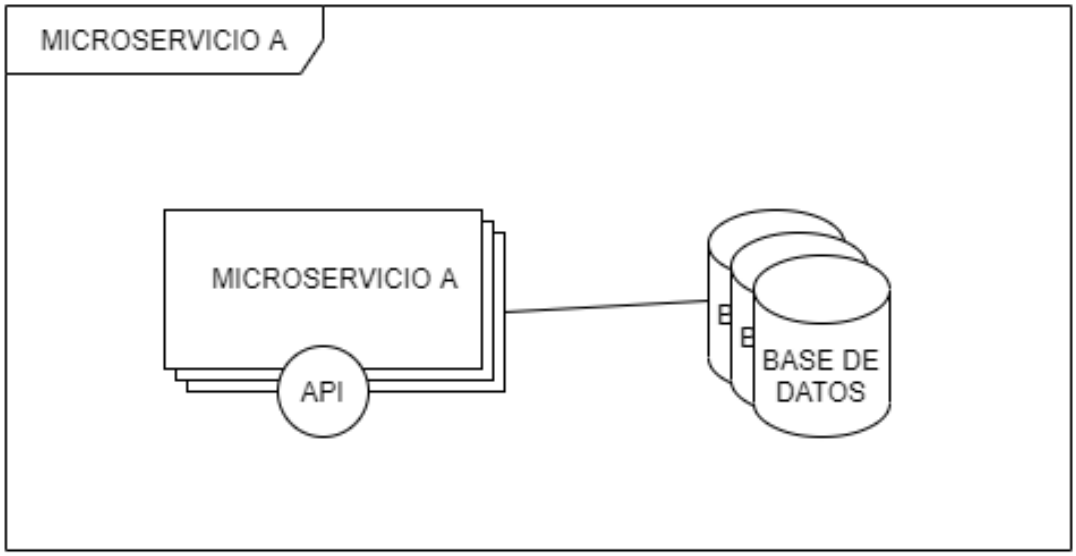


Figura 2.4: Microservicio core

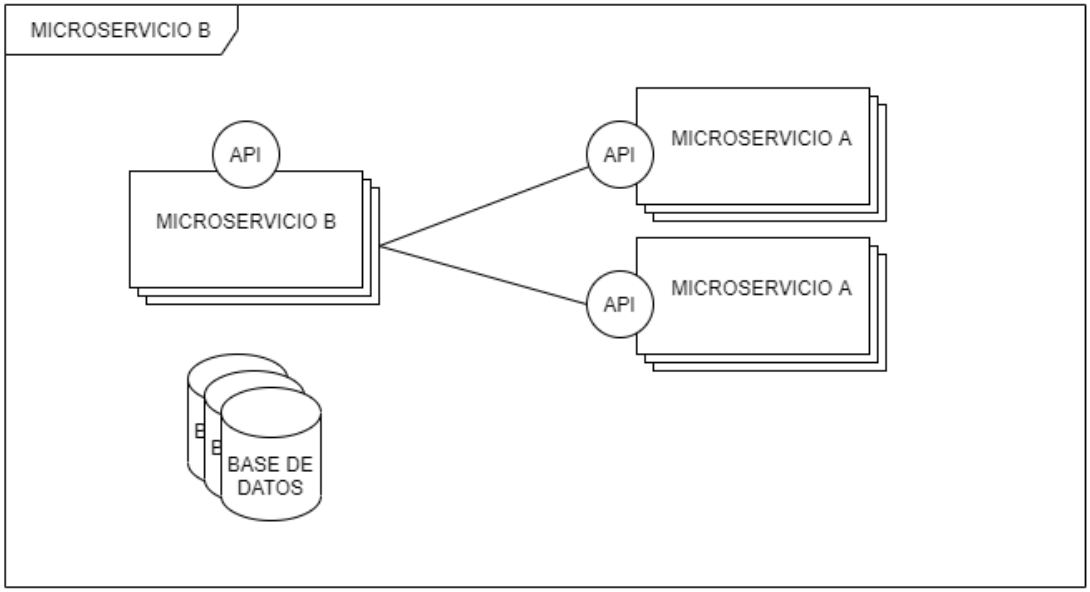


Figura 2.5: Microservicio de composición

2.5 Componentes

2.5.1 Servidor de configuración

Dentro del mundo del desarrollo es buena práctica separar la configuración de una aplicación del código fuente de esta. Al hacer esta separación, se ahorra tener que modificar el código y, a su vez, recompilarlo al querer hacer algún cambio en la configuración. En líneas generales, se recurre a un fichero de propiedades para almacenar la información. Los ficheros con formato YAML, JSON o XML son los más comunes.

Este enfoque funciona con sistemas pequeños, pero no está pensado para aplicaciones basadas en microservicios. La gestión de la configuración en este tipo de aplicaciones se convierte en un problema, debido a que los diferentes microservicios y sus instancias pueden acabar alojados en diferentes máquinas y no es manejable replicar la información de configuración en cada instancia. Por tanto, el desarrollo de estas aplicaciones enfatiza:

- La configuración debe ser separada de manera completa del código.
- Construir un servidor de configuración que sea inmutable.

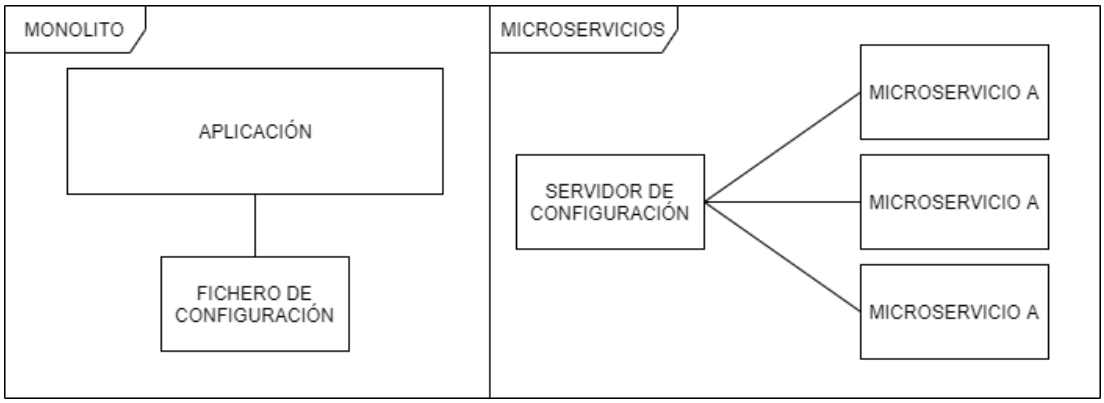


Figura 2.6: Servidor de configuración

El servidor de configuración debería seguir los siguientes principios:

- Separar completamente la información de los servicios de configuración de la instancia física del servicio. La configuración no deberá ser desplegada con el servicio, sino que deberá leerse de un repositorio centralizado como, por ejemplo, GIT.
- Abstraer el acceso a los datos de configuración. En lugar de escribir código que recoja los datos del repositorio, hacer un servicio REST para recuperar los datos.
- Centralizar los repositorios de configuración en tan pocos como sea posible.

- Al separar completamente la configuración de los microservicios se crea una dependencia. Como consecuencia, será necesario dotar al servidor de alta disponibilidad y redundancia.

2.5.2 Descubrimiento de servicios

En cualquier arquitectura distribuida es necesario encontrar la dirección física en la que se encuentra cada servicio. El descubrimiento de servicios es esencial para las aplicaciones basadas en microservicios por dos razones:

- En primer lugar, con un balanceador de carga ofrece a la aplicación la capacidad de escalar de manera horizontal rápidamente y, además, reduce la cantidad de instancias que se ejecutan en un mismo entorno.
- En segundo lugar, los servicios consumidores se abstraen de la localización física de las instancias debido al descubrimiento de servicios. Como consecuencia, los servicios consumidores no conocen la dirección física de las instancias, lo que facilita que se puedan tanto añadir como eliminar las instancias del "pool" de servicios disponibles.

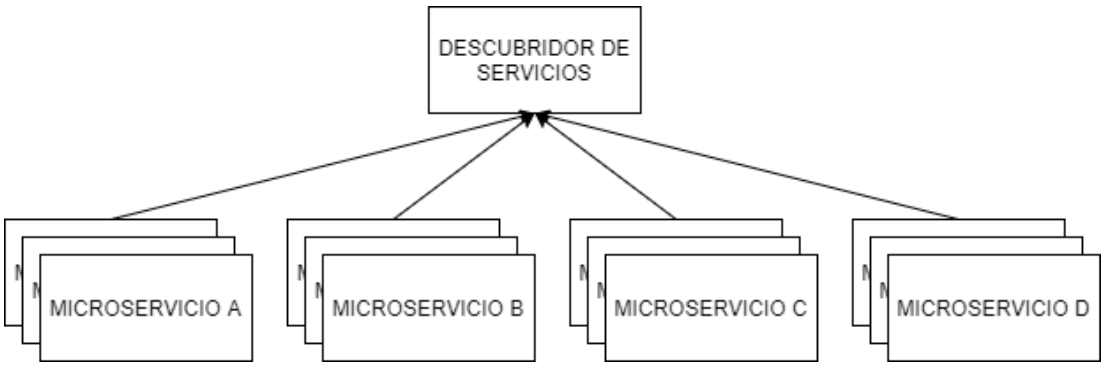


Figura 2.7: Eureka

Esta manera de escalar los servicios sin interrumpir a los consumidores es tremendamente útil para los desarrolladores acostumbrados a lidiar con aplicaciones monolíticas. Con estas últimas, la única manera de escalar es de manera vertical, agregando hardware más potente para correr el único servicio, lo que limita mucho la escalabilidad. Con una arquitectura orientada a microservicios se escala de manera horizontal, por lo tanto se corren los servicios en un gran número de máquinas de bajo coste y favorece la escalabilidad.

Otro gran beneficio que nos ofrece es resiliencia. Puesto que, cuando una instancia de un microservicio se encuentra inestable o inalcanzable, se retira de la lista interna de servicios disponibles. El daño causado por la caída de la instancia es mínimo, esto se debe a que se enruta de nuevo a una instancia disponible.

Eureka server [2] forma parte del framework de Spring Cloud [3]. Como su nombre indica, está diseñado para ejecutarse como un servidor y, en nuestro caso, como un microservicio. Será nuestro descubridor de servicios.

2.5.3 Gestión de fallos

En cualquier momento nuestro sistema puede experimentar fallos, especialmente al ser un sistema distribuido. Por tanto, hay que tenerlo en cuenta a la hora de construir nuestra aplicación. Es frecuente que para construir sistemas resistentes se centre el foco en la falla total de una pieza de la infraestructura o un servicio clave. En conclusión, se hace hincapié en crear redundancia, equilibrar la carga y segregar la infraestructura en varias ubicaciones.

A pesar de que estos enfoques son muy buenos ante una pérdida, no abordan el total de los problemas. Resulta sencillo detectar que un servicio no se encuentra disponible, sin embargo, cuando este funciona con lentitud, detectar el bajo rendimiento es extremadamente difícil por varios problemas:

- La degradación del servicio puede ser intermitente y acabar colapsando.
- Los servicios suelen ser síncronos y, por lo tanto, no interrumpen una llamada, esperan a que vuelva.
- Las aplicaciones suelen estar diseñadas para hacer frente a fallos completos, no parciales. Si un recurso no falla por completo, se continuará realizando peticiones hasta que colapse.

En este tipo de aplicaciones distribuidas, un servicio que no esté funcionando correctamente puede desencadenar un efecto en cascada. Al componerse de una gran cantidad de servicios distribuidos, podría verse corrupta una transacción que afecte a varios servicios y poner en peligro el ecosistema de la aplicación. Para evitar esto existen varios patrones que ponen el foco en el "fallo rápido". Este fallo rápido permite que no sean consumidos recursos valiosos como conexiones a bases de datos o thread pools.

- Circuit Breaker: Recibe su nombre por el aparato eléctrico, puesto que funcionan de manera similar. En un sistema eléctrico, se activa si la corriente que fluye es mayor de lo que debería, lo que conlleva a romper la conexión del sistema eléctrico para evitar que los aparatos conectados a este no sufran daños.

  El circuit breaker software supervisará cuando se realice una llamada a un servicio remoto. Si tarda demasiado el disyuntor se activará y "matará" la llamada. Cuando a un servicio se le interceden muchas llamadas se cortará la comunicación con este, provocando un "fallo rápido" y evitando futuras llamadas al recurso.

- Procesamiento de respaldo: Cuando una llamada a un recurso remoto falla, en lugar de generar una excepción se intentará (por parte del consumidor) ejecutar una ruta de código alternativa. Estas alternativas pueden ser buscar los datos de otra fuente o ponerlo en una cola para un procesamiento futuro. Por ejemplo: en una aplicación de comercio electrónico que haga recomendaciones personalizadas, si el microservicio de recomendaciones falla en vez de generar una excepción podría mostrar los productos más vendidos.

Para implementar todos estos patrones se requiere un conocimiento muy amplio en concurrencia, sin embargo, Spring Cloud tiene una librería open source (construida por Netflix[4]) que facilita la implementación de todos los patrones mencionados anteriormente, su nombre es Netflix's Hystrix[5]. Esta librería nos brinda una solución más que probada que usa a diario esta gran multinacional en su arquitectura basada en microservicios.

2.5.4 Enrutamiento de servicios

Hemos visto como un descubridor de servicios nos facilita consumir las instancias de nuestro servicios. Sin embargo, si no queremos tener que llamarlos individualmente necesitamos un punto de entrada al sistema.

El punto de entrada o API Gateway actúa como intermediario entre el cliente de la aplicación y el sistema. A través de este, se exponen al exterior de la nube de microservicios todas las operaciones con las que el cliente puede interactuar. De esta manera, la aplicación cliente envía peticiones a un único endpoint (el API Gateway) y permanece abstraída de la nube de microservicios.

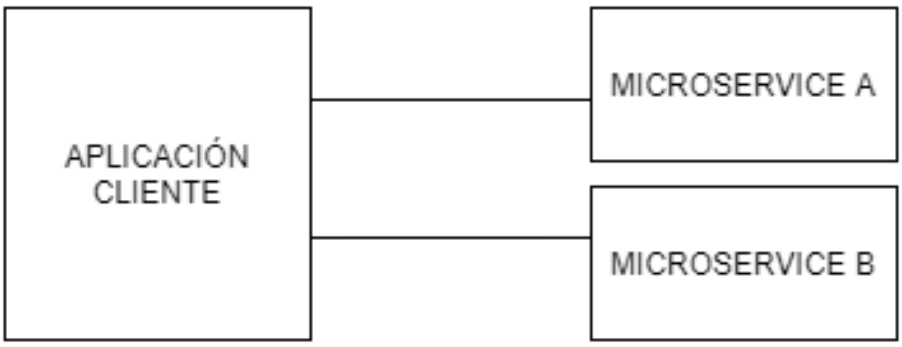


Figura 2.8: Sin API Gateway

Como resultado de que el punto de entrada se encuentre entre las llamadas del cliente y el sistema, se puede centralizar parte de la secularización de las operaciones que ofrece

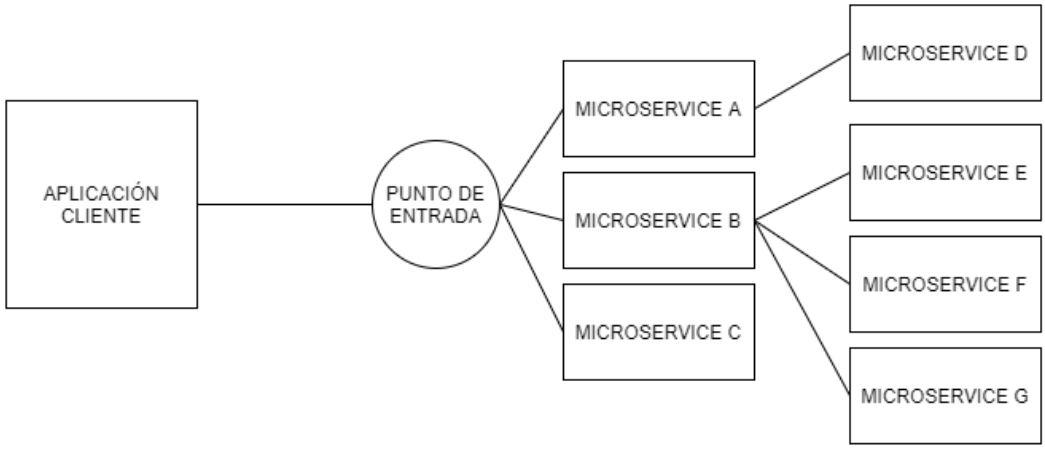


Figura 2.9: Con API Gateway

la nube de microservicios en el API Gateway, en lugar de que cada microservicio tenga que encargarse completamente de este aspecto. Esto facilita a los desarrolladores a no tenerlas que implementar en sus respectivos microservicios.

El API Gateway nos ofrece:

- Rutas estáticas: se colocan todas las llamadas a servicios a través de una única URL. Simplifica el desarrollo, ya que los desarrolladores solo tienen que conocer el punto final o endpoint para todos sus servicios.
- Autorización y autenticación: gracias a que todas las peticiones pasan por este punto de entrada, es un buen lugar para comprobar si la petición la realiza alguien autenticado y además está autorizado para realizar la llamada.
- Métricas y logs: este también es un buen lugar para realizar métricas y logs, que pueden estar complementados con los microservicios.

Spring Cloud integra Zuul[6], mediante la aportación de Netlfix. Zuul es un servicio gateway, por lo que será tratado como un microservicio. Zuul nos permitirá implementar de manera sencilla y a grano muy fino, las rutas. Además, cuenta con un sistema de filtros que ayuda de manera notoria a inspeccionar las peticiones que pasan a través de este y realizar alguna operación si fuese necesario.

2.5.5 Seguridad

Una aplicación segura conlleva múltiples capas de protección:

- Asegurarse de que existan controles de usuario adecuados para poder validar que un usuario es quien dice ser y que tiene permiso para hacer lo que intenta.
- Mantener la infraestructura en buen estado. Esto quiere decir, parcheada, actualizada y con controles de acceso muy exhaustivos y restringidos.

En este apartado haremos hincapié en el primer punto, autenticación y autorización del usuario. El segundo punto son temas extremadamente amplios que están fuera del alcance de este proyecto.

Un framework que es muy usado para gestionar la autenticación y autorización es OAuth2[7]. Sus principales componentes son:

- Recurso protegido: Recurso (en este caso, microservicio) que se desea proteger y garantizar que solo los usuarios autenticados y autorizados puedan acceder.
- Propietario del recurso: Define que aplicaciones pueden llamar a los microservicios, que usuarios pueden acceder y qué es lo que pueden hacer.
- Una aplicación: Esta llamará al servicio en nombre del usuario. Los usuarios rara vez invocan a un servicio directamente, lo hacen a través de aplicaciones.
- Servidor de autenticación OAuth2: Es el intermediario entre la aplicación y los servicios. El servidor OAuth2 permite al usuario autenticarse sin tener que pasar las credenciales a cada uno de los microservicios que la aplicación llame en nombre del usuario.

2.5.6 Comunicación entre microservicios

En este proyecto, los microservicios se comunicarán a través de una API REST y son consumidos mediante un Feign client.

Feign client[8] es una librería de Spring Cloud creada por Netlfix. Esta librería adopta un enfoque diferente para llamar a un servicio REST. El desarrollador define una interfaz con anotaciones de Spring Cloud y las operaciones que sean necesarias. Spring Cloud framework de manera dinámica, generará una clase proxy que será usada para invocar al servicio REST. Gracias a esta librería no será necesario escribir código para llamar al servicio, solo la definición de la interfaz.

```java
@FeignClient(name = "microservicejob")
@RequestMapping("/jobs")
public interface JobClient {

    @PostMapping("/newJob")
    public String newJob(@RequestBody JobCreate jobDto);
```

Figura 2.10: Feign Client

Capítulo 3

Metodología

Una metodología en el desarrollo software se refiere a un marco de trabajo que sirve como guía para estructurar, planificar, construir y mantener un proyecto. Existen muchos tipos de metodologías para el desarrollo software, todas ellas diseñadas para el trabajo en equipo. El uso de una metodología ágil da más flexibilidad a la hora del cambio y aumenta la productividad. El marco en el que nos basaremos será SCRUM, aunque no se adoptará en su plenitud por no ser aplicable a un proyecto de una sola persona.

3.1 Scrum

Scrum[9] es un marco de trabajo para el desarrollo ágil de software. Se trata de un proceso en el que se aplican un conjunto de buenas prácticas con el objetivo de obtener un resultado óptimo en los proyectos. Scrum se basa en un desarrollo incremental, realizando entregas parciales y regulares del producto final. Las entregas serán priorizadas por el valor que aporten al cliente. Su uso es altamente recomendado en proyectos con los requisitos poco definidos o cambiantes, además de proyectos que precisan obtener resultados pronto.

3.1.1 Roles

Scrum es un marco de trabajo para el desarrollo ágil de software. Se trata de un proceso en el que se aplican un conjunto de buenas prácticas con el objetivo de obtener un resultado óptimo en los proyectos. Scrum se basa en un desarrollo incremental, realizando entregas parciales y regulares del producto final. Las entregas serán priorizadas por el valor que aporten al cliente. Su uso es altamente recomendado en proyectos con los requisitos poco definidos o cambiantes, además de proyectos que precisan obtener resultados pronto.

- PRODUCT OWNER: es el encargado de definir los requisitos, que serán recogidos en el product backlog, de priorizarlos y validarlos.

- SCRUM MASTER: su objetivo es eliminar los obstáculos del equipo de desarrollo y guiarlo para optimizar el funcionamiento de la metodología.
- DESARROLLADORES: son un equipo multidisciplinar cuyo objetivo es realizar las tareas que sean necesarias para hacer la entrega planificada de cada iteración

3.1.2 Sprint

El sprint es un período temporal que suele ser definido por el equipo en base a su experiencia, teniendo en cuenta que su duración recomendada es entre 2 y 4 semanas. En cada período se seleccionará una lista de tareas a realizar (Sprint Backlog) y al final del sprint, se presentan los avances obtenidos, que potencialmente podrán ser entregados al cliente como un incremento del producto.

3.1.3 Artefactos

Los artefactos fueron diseñados para maximizar la transparencia de información clave, para que dentro del equipo se tenga la misma comprensión sobre el proyecto.

- Product Backlog: es una lista, normalmente ordenada por prioridad, que recoge todas las necesidades y requisitos que son de interés para el cliente. Esta, que es responsabilidad del Product Owner, puede verse modificada a lo largo del proyecto.
- Sprint Backlog: recoge un conjunto de elementos del Product Backlog que intentarán ser abordados en un sprint. No debe modificarse cuando el sprint está en curso.
- Historias de usuario: describen una funcionalidad concreta del producto. Se expresan desde el punto de vista del usuario empleando un lenguaje coloquial. Ayudan al desarrollador a una mejor compresión del requisito. Se estructuran de la siguiente forma: Como [ROL DE USUARIO] quiero [acción] para [valor añadido]

3.1.4 Reuniones

La comunicación en Scrum es clave. Por ese motivo, el marco de trabajo establece una serie de reuniones con el objetivo de que la comunicación dentro del equipo se produzca con fluidez.

- Sprint Planning: el objetivo principal es establecer el trabajo que se realizará en el siguiente Sprint. El equipo al completo discutirá sobre los elementos que se intentarán abarcar, se descompondrán en tareas y se hará una estimación de las mismas. El resultado de esta reunión es el Sprint Backlog.

- Daily Scrum: Es una reunión diaria en la que participa el Scrum Master y el equipo de desarrollo. Su duración no debería exceder los 15 minutos. El objetivo es que todos los participantes sepan el estado en el que se encuentra el trabajo. Cada integrante deberá responder a las siguientes preguntas:
  - ¿Qué hice ayer?
  - ¿Qué voy a hacer hoy?
  - ¿Qué dificultades y problemas tengo?
- Sprint review: esta reunión se lleva a cabo al final de cada sprint. Su objetivo es revisar el incremento realizado en este. El equipo mostrará el resultado de su trabajo, normalmente con una "demo" al Product Owner.
- Sprint Retrospective: se pretende revisar el proceso y la relación del equipo para planificar formas de aumentar su calidad y eficacia. Se intenta responder a lo siguiente:
  - ¿Qué salió bien en el Sprint?
  - ¿Qué se puede mejorar?
  - ¿Qué nos comprometemos a mejorar en el próximo Sprint?

3.2 Aplicación de scrum a nuestra metodología

Como se mencionó anteriormente, al ser solo un único participante en el proyecto se harán adaptaciones de la metodología Scrum a nuestro proyecto. El proyecto se llevará a cabo mediante un método iterativo e incremental. Cada iteración consistirá en análisis, diseño, implementación y pruebas. En cada una de estas se implementa un subconjunto de funcionalidades del sistema.

Al ser un único participante en el equipo, los roles no aplican debido a que es un proyecto personal, por tanto, la figura de product owner y scrum master están recogidas en la misma persona. Asimismo, las reuniones no serán realizadas por el motivo citado anteriormente.

Cada iteración del proyecto será un Sprint. Para cada Sprint se propondrá un Sprint Backlog que contendrá un conjunto de funcionalidades sacadas del Product Backlog.

Capítulo 4

Análisis de requisitos global

Para comenzar, en este capítulo se describirán los roles de usuario y los requisitos funcionales. La captura de requisitos se realiza mediante historias de usuario.

4.1 Roles

- Usuario sin autenticar: un usuario sin autenticar podrá registrarse, buscar trabajos y ver en detalle a los trabajadores.

- Usuario autenticado: un usuario autenticado podrá, además de buscar trabajos y ver en detalle a los trabajadores, contratar un trabajo, ver el historial de contrataciones, puntuar un trabajo que ha sido realizado, cambiar su contraseña y actualizar sus datos.

- Trabajador sin autenticar: un trabajador sin autenticar solo podrá registrarse.

- Trabajador autenticado: a un trabajador autenticado se le permitirá dar de alta, modificar y eliminar un trabajo. A su vez, podrá ver su historial de contrataciones, cambiar su contraseña y actualizar sus datos.

- Administrador: este rol queda fuera del dominio de este trabajo para acotar el proyecto al tamaño esperado de un TFG. Este se encargaría de la gestión de las categorías y de la gestión de los trabajadores para evitar fraudes y expulsar a trabajadores no competentes.

4.2 Product backlog

En este apartado, serán recogidas las historias de usuario en la siguiente tabla:

ID	Nombre	Descripción
US01	Registro	Como usuario y trabajador sin autenticar quiero registrarme
US02	Autenticación	Como usuario y trabajador sin autenticar quiero iniciar sesión
US03	Cerrar sesión	Como usuario y trabajador autenticado quiero cerrar sesión.
US04	Cambio de contraseña	Como usuario y trabajador autenticado quiero cambiar la contraseña.
US05	Actualización de datos	Como usuario y trabajador autenticado quiero actualizar mis datos.
US06	Crear servicio	Como trabajador quiero poder crear un servicio para poder vender mi trabajo.
US07	Ver servicio	Como usuario y trabajador quiero poder ver el detalle de un servicio
US08	Buscar servicios por categoría	Como usuario quiero buscar los servicios, ordenados por puntuación, por categoría.
US09	Ver servicios propios	Como trabajador quiero ver todos los trabajos que ofrezco.
US10	Solicitar contratación	Como usuario autenticado quiero solicitar la contratación de un servicio.
US11	Ver historial de contrataciones de un usuario	Como usuario quiero ver un historial de contrataciones filtradas por estado.
US12	Ver detalle de contratación	Como usuario quiero ver el detalle de una solicitud de contratación.
US13	Puntuar servicio realizado	Como usuario quiero poder puntuar un servicio realizado.
US14	Ver historial de contrataciones de un servicio	Como trabajador quiero ver el historial de contrataciones filtradas por estado.
US15	Aceptar y rechazar solicitud de servicio	Como trabajador quiero poder aceptar o rechazar la solicitud de un servicio.
US16	Finalizar o cancelar un servicio aceptado	Como trabajador, una vez acepte el servicio, quiero poder cancelar o dar por finalizado el servicio aceptado.

Tabla 4.1: Product backlog

4.3 Funcionalidades

US01-Registro

Para el registro, el usuario proporcionará los siguientes datos:

- Nombre de usuario (único)

- Email (válido)
- Contraseña
- Nombre y apellidos
- Teléfono

En caso de estar todo correcto, se realizará el registro y se autenticará de manera automática, lo que dará acceso a todas las funcionalidades de usuario dentro de la aplicación. De existir algún error aparecerá el fallo cometido.

Figura 4.1: Mockup de registro de usuarios

US02- Autenticación

La autenticación se realizará con el nombre de usuario y la contraseña. Si el par usuario-contraseña es correcto se autenticará al usuario y se le dará acceso a todas las funcionalidades

de la aplicación. En caso contrario, se mostrará un mensaje de error.

US03- Cerrar sesión

Un usuario autenticado podrá cerrar sesión pulsando el botón de Cerrar sesión. Después, no se tendrá acceso a todas las funcionalidades hasta que se vuelva a autenticar.

US04-Cambio de contraseña

Un usuario autenticado podrá cambiar su contraseña. Para esto habrá que pulsar en un botón de cambiar de contraseña, rellenar el formulario con la contraseña actual y la nueva. Si se puso la contraseña actual correcta se actualizará y se redirigirá a la página principal. En caso contrario saldrá un mensaje de error.

US05-Actualización de datos

Un usuario autenticado podrá actualizar sus datos. Existirá un botón de Actualizar datos que al pulsarlo le llevará a un formulario auto-rellenado con los datos actuales, que se podrán modificar. Si todos los datos actualizados son válidos se hará la actualización y se redirigirá a la página principal. Si algún dato no es válido se mostrará el mensaje de error.

Crear servicio

Un trabajador autenticado tendrá la opción de crear un servicio. Habrá un botón de crear servicio que nos llevará a un formulario que requerirá los siguientes datos:

- Nombre del servicio
- Lugar
- Dirección
- Precio
- Categoría

Figura 4.2: Mockup de crear servicio

Ver servicio

4.3.1 usuario

Un usuario que haga click en un servicio, abrirá el detalle de este, que contendrá la siguiente información:

- Id
- Nombre del servicio
- Nombre del trabajador
- Categoría a la que pertenece
- Lugar en el que se ofrece el servicio
- Precio
- Media de puntuación del servicio
- Puntuaciones totales

- Botón para solicitar la contratación del servicio (solo si está autenticado)
- Lista paginada de las valoraciones del servicio

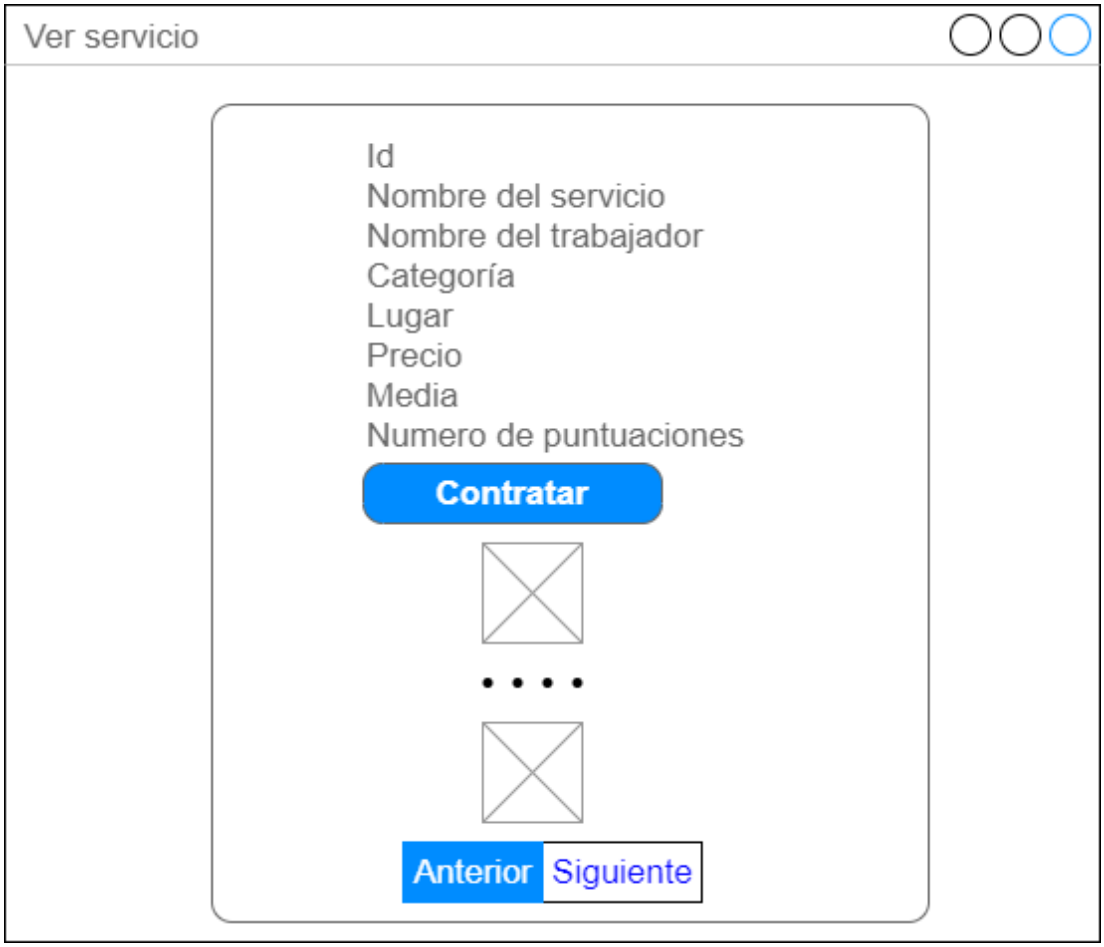


Figura 4.3: Mockup de ver servicio por parte del usuario

4.3.2 Trabajador

Un trabajador que haga click en un servicio, abrirá el detalle de este, que contendrá la siguiente información:

- Id
- Nombre del servicio
- Nombre del trabajador
- Categoría a la que pertenece

- Lugar en el que se ofrece el servicio
- Precio
- Media de puntuación del servicio
- Puntuaciones totales
- Enlace a las solicitudes de servicio pendientes para ese servicio
- Enlace a los servicios en curso para ese servicio
- Enlace a los servicios completados para ese servicio
- Lista paginada de las valoraciones del servicio
- Nombre del servicio

US08- Buscar servicios por categoría

En la página principal de la aplicación de usuarios habrá un desplegable con las categorías disponibles para buscar un servicio. El resultado de la búsqueda será una lista paginada, ordenada por puntuación, con los servicios ofrecidos para esa categoría.

US09- Ver servicios propios

En la página principal de la aplicación de trabajadores habrá una lista paginada con todos los servicios que está prestando actualmente. Por cada servicio de la lista se mostrará lo siguiente:

- Enlace al detalle del servicio en el nombre de este
- Categoría a la que pertenece
- Lugar en el que se ofrece el servicio
- Enlace a las solicitudes de servicio pendientes para ese servicio
- Enlace a los servicios en curso para ese servicio
- Enlace a los servicios completados para ese servicio

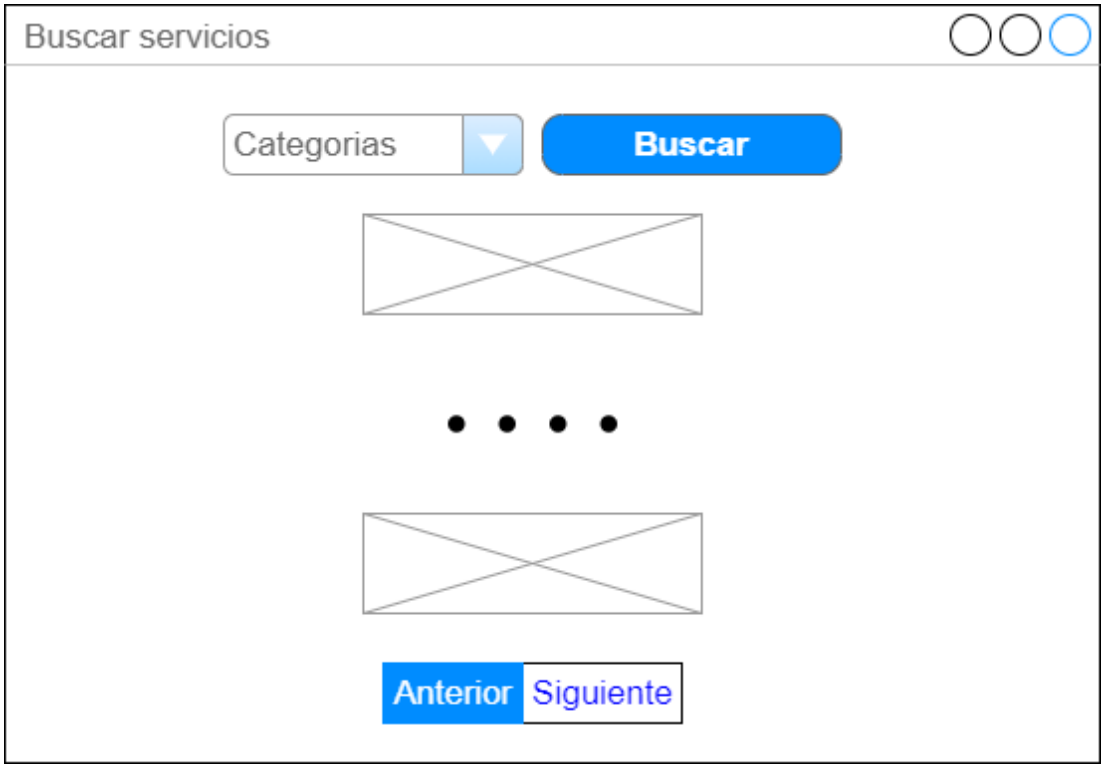


Figura 4.4: Mockup de buscar servicios

US10- Solicitar contratación

En la aplicación de usuarios, dentro de ver servicio (US07), si el usuario está autenticado aparecerá un botón para solicitar la contratación. Este botón nos llevará a un formulario que nos pedirá lo siguiente:

- Lugar
- Dirección

US11- Ver solicitudes de contrataciones de un usuario

Un usuario autenticado podrá ver todo su historial filtrado por estado. Para ello, tendrá un botón que le llevará a una lista paginada con las contrataciones del estado seleccionado. El contenido de la lista por cada contratación será el siguiente:

- Enlace con el id que llevará al detalle de la solicitud.
- Categoría del servicio solicitado

- Nombre del servicio
- Dirección
- Lugar
- Tiempo en minutos
- Estado

US12- Ver solicitud de contratación

Usuario

Un usuario autenticado podrá ver el detalle de sus solicitudes. Se podrá acceder haciendo click en el id de una solicitud de la lista obtenida en US11. El detalle mostrará lo siguiente:

- Id
- Nombre del servicio
- Categoría a la que pertenece
- Dirección
- Lugar
- Tiempo en minutos
- Estado

Trabajador

Un trabajador podrá ver en detalle sus solicitudes e, igualmente, acceder haciendo click en el id de una solicitud de la lista obtenida en US14. El detalle mostrará lo mismo que para un usuario, pero además el contacto del usuario que lo solicita. A mayores, para cada estado se mostrará lo siguiente:

- Pendiente: botones para aceptar o rechazar la solicitud. (US15)
- En curso: botones para finalizar o cancelar el servicio. (US16)
- Finalizado: mostrará la valoración del usuario en caso de haberla realizado.

US13- Puntuar un servicio realizado

Desde el detalle de una solicitud de contratación, una vez haya sido aceptada y completada, se podrá valorar el servicio ofrecido. El usuario autenticado podrá poner una puntuación (de 1 a 5) y hacer un comentario.

Figura 4.5: Mockup de puntuar

US14- Ver solicitudes de contrataciones de un servicio

Un trabajador podrá ver todas las solicitudes de contratación para un determinado servicio filtrado por su estado, accediendo mediante diferentes enlaces (US09). El contenido será una lista paginada que contendrá lo siguiente:

- Id

- Categoría
- Dirección
- Lugar
- Tiempo en minutos
- Estado

US15 - Aceptar o rechazar solicitud de servicio

El trabajador podrá aceptar o rechazar la solicitud desde el detalle de la misma. (US12)

Figura 4.6: Mockup de aceptar solicitud

US16 - Finalizar o cancelar una solicitud aceptada

El trabajador podrá finalizar o cancelar un servicio ya aceptado desde el detalle de la misma. (US12)

Capítulo 5
Planificación

En este capítulo se explicará la planificación que se ha llevado a cabo en este proyecto.

5.1 Sprints

Por la ausencia de experiencia con este tipo de arquitectura, no es posible realizar una buena planificación temporal. En base a lo aprendido en cada Sprint se intentará hacer ir haciendo una mejor estimación a lo largo del proyecto.

5.1.1 Sprint 0: Formación

Como se comentó anteriormente, no hay experiencia previa con las arquitecturas basadas en microservicios. Por este motivo, se decide dedicar el sprint inicial a formarse en este tipo de arquitecturas y las tecnologías que sean usadas en estas.

5.1.2 Sprint 1: Análisis del proyecto

En este sprint se hará un análisis de los casos de uso del sistema. Este analisis servirá para desgranar las funcionalidades, comprender cuales están relacionadas y agruparlas en un microservicio. El resultado de este sprint será el modelo conceptual de la nube de microservicios.

5.1.3 Sprint 2: Gestión de usuarios

En esta iteración se abordarán las funcionalidades de gestión de usuarios y trabajadores, concretamente:

- US01 - Registro
- US02 - Autenticación
- US03 - Cerrar sesión

- US04 - Cambiar de contraseña
- US05 - Actualizar datos

5.1.4 Sprint 3: Componentes de la arquitectura

Una vez que tengamos microservicios funcionales se implementarán los componentes más importantes en nuestra arquitectura.

- Zuul
- Eureka

5.1.5 Sprint 4: Gestión de servicios

Se implementarán las funcionalidades relacionadas con la gestión de servicios. Se abordará:

- US06 - Crear servicio
- US07 - Ver servicio
- US08 - Buscar servicios por categoría
- US09 - Ver servicios propios

5.1.6 Sprint 5: Gestión de contrataciones

En este sprint se tratarán las funcionalidades relacionadas con la gestión de contrataciones de los servicios. Para ello serán implementadas:

- US10 - Solicitar Contratación
- US15 - Aceptar o rechazar solicitud de servicio
- US16 - Finalizar o cancelar un servicio aceptado

5.1.7 Sprint 6: Visualización de contrataciones

Tras incorporar las operaciones de gestión de las contrataciones, se decide implementar las funcionalidades de búsqueda que quedan recogidas:

- US11 - Ver historial de contrataciones de un usuario
- US12 - Ver detalle de contratación
- US14 - Ver historial de contrataciones de un servicio

5.1.8 Sprint 7: Puntuación de servicios realizados

En este Sprint se implementará la US13- Puntuar servicio realizado. Además se integrará a las funcionalidades ya existentes que hacen uso de la puntuación. Como por ejemplo para la media de los servicios, para ver las puntuaciones de un servicio, etc..

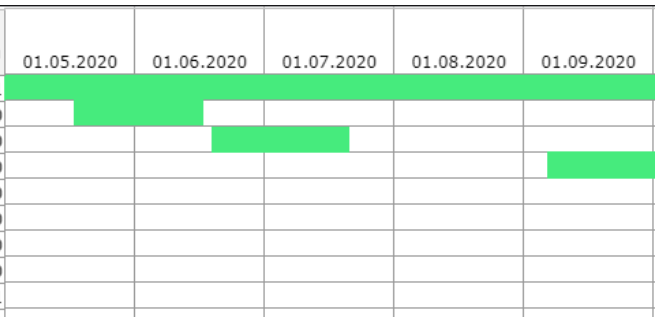
5.1.9 Sprint 8: Pruebas de aceptación y elaboración de la memoria

En esta última iteración se realizarán pruebas de aceptación sobre el sistema, para comprobar que funciona, en su conjunto, como es esperado. Además, se elaborará la memoria.

5.2 Planificación temporal

De manera ideal, los sprints deberían ser de una duración similar. Como se comentó al principio del capítulo, la falta de experiencia dificulta una correcta estimación en las primeras iteraciones. Además, al tratarse de un trabajo de fin de grado, las horas por sprint varían. Esto es así porque no hay una carga fija como en un trabajo, que la jornada laboral es de 8 horas y se adecuan las tareas a ello. En la figura 5.1 se muestra un diagrama de Gantt con la planificación final y la figura 5.2 con las horas dedicadas a cada Sprint.

Nombre de la tarea	Fecha de inicio	Fecha de finalización
TFG	04.05.2020	23.02.2021
Sprint 0	19.05.2020	16.06.2020
Sprint 1	19.06.2020	16.07.2020
Sprint 2	08.09.2020	02.10.2020
Sprint 3	05.10.2020	19.10.2020
Sprint 4	20.10.2020	03.11.2020
Sprint 5	04.11.2020	25.11.2020
Sprint 6	27.11.2020	18.12.2020
Sprint 7	07.01.2021	09.02.2021
Sprint 8	11.02.2021	23.02.2021



Nombre de la tarea	Fecha de inicio	Fecha de finalización
TFG	04.05.2020	23.02.2021
Sprint 0	19.05.2020	16.06.2020
Sprint 1	19.06.2020	16.07.2020
Sprint 2	08.09.2020	02.10.2020
Sprint 3	05.10.2020	19.10.2020
Sprint 4	20.10.2020	03.11.2020
Sprint 5	04.11.2020	25.11.2020
Sprint 6	27.11.2020	18.12.2020
Sprint 7	07.01.2021	09.02.2021
Sprint 8	11.02.2021	23.02.2021

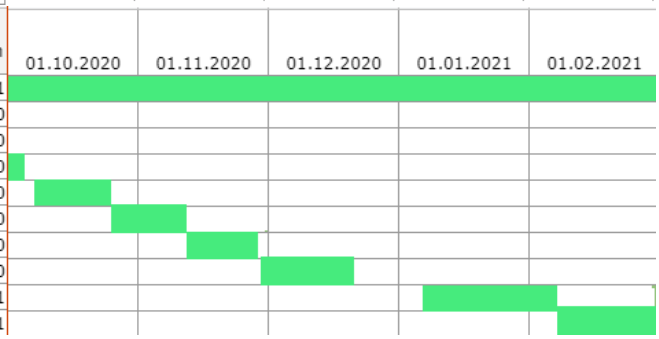


Figura 5.1: Diagrama de Gant

Sprint	Fecha inicio	Fecha fin	Horas
Sprint 0	19.05.2020	16.06.2020	97
Sprint 1	19.06.2020	16.07.2020	102
Sprint 2	08.09.2020	02.10.2020	80
Sprint 3	05.10.2020	19.10.2020	55
Sprint 4	20.10.2020	03.11.2020	55
Sprint 5	04.11.2020	25.11.2020	55
Sprint 6	27.11.2020	18.12.2020	55
Sprint 7	07.01.2021	09.02.2021	73
Sprint 8	11.02.2021	23.02.2021	41
Total	19.05.2020	23.02.2021	613

Figura 5.2: Planificación temporal y horas dedicadas

Capítulo 6

Fundamentos tecnológicos

En lo que respecta a este punto nos encontramos con diversas tecnologías, utilizadas en tres grupos notoriamente diferenciados: backend, frontend y desarrollo.

6.1 Tecnologías empleadas en el backend

El backend consiste en una nube de microservicios. Cada microservicio de la nube, emplea una API REST para comunicarse entre ellos. Los microservicios están implementados con el lenguaje de programación Java y como base de datos usan MongoDB.

6.1.1 Java

Java [10] es un lenguaje de programación orientado a objetos y multiplataforma. Este lenguaje puede ser ejecutado en cualquier plataforma que soporte la máquina virtual de Java o JVM. La JVM se basa en el concepto de un procesador virtual, el cual, ejecuta programas escritos en este lenguaje. Concretamente ejecuta código en formato bytecode, que es el resultado de la compilación del código fuente.

6.1.2 Spring

Spring [11] es un marco de trabajo para el desarrollo de aplicaciones y contenedor de inversión del control para Java. El contenedor de inversión de control permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos, se lleva a cabo principalmente a través de la inyección de dependencias. La inyección de dependencias es un patrón de diseño orientado a objetos, que consiste en proveer objetos a una clase en vez de ser la clase la que cree los objetos. Los objetos no son creados por nuestras clases, sino que los proporciona otra clase contenedor. Gracias a todo lo citado anteriormente, Spring implementa las estructuras de alto nivel mientras que el desarrollador puede centrarse en la lógica de la aplicación. Dentro de Spring Framework serán usados los siguientes módulos:

- Spring Security: Para la autenticación y autorización.
- Spring Cloud: Para la arquitectura basada en microservicios.
- Spring Boot: Para facilitar el uso de los distintos frameworks del ecosistema Spring.

6.1.3 Apache Maven

Es una herramienta de software para la gestión y construcción de proyectos Java [12]. Su configuración se basa en un fichero con formato XML, llamado POM (Project Object Model). En el POM se describe el proyecto software y las dependencias de módulos y componentes externos que sean necesarios para el proyecto. Las funcionalidades principales de Maven son las siguientes:

- Automatiza la compilación y el empaquetado del software.
- Gestiona de manera simple las dependencias de módulos del proyecto o librerías externas. Maven tiene un extenso repositorio de versiones que facilita la gestión de librerías externas.

6.1.4 MongoDB

MongoDB [13] es una base de datos NoSQL, orientada a documentos y de código abierto. Es una base de datos muy flexible, gracias a que tiene un esquema dinámico. Ese esquema facilita la integración de los datos a las aplicaciones. Los datos se guardan en estructuras de datos BSON (similar a JSON). Las ventajas que nos ofrece MongoDB son las siguientes:

- Es distribuida, por lo que puede usarse una única instancia o aumentarlas creando un "cluster". Esto hace que tenga una gran escalabilidad.
- Gracias a su sistema de esquemas dinámico permite, con gran facilidad, cambiar el modelo de datos.
- Agilidad, MongoDB es más rápido que las bases de datos tradicionales SQL, gracias a su capacidad de manejar grandes volúmenes de datos.

6.1.5 Eclipse

Eclipse [14] es un IDE (Integrated Development Environment) de código abierto. Es apto para múltiples lenguajes de programación, aunque su uso mayoritario es para el entorno Java. Eclipse tiene un marketplace que permite, de manera muy simple, la descarga, instalación e integración de plugins. Al conseguir "centralizar" las herramientas necesarias para el desarrollo en un solo entorno, mejora con creces la productividad del desarrollador.

6.2 Tecnologías empleadas en el frontend

6.2.1 Javascript

Javascript [15] es un lenguaje de programación interpretado, de tipado débil y dinámico. Este lenguaje es el estándar en la implementación de aplicaciones del lado cliente en la web. Todos los navegadores modernos interpretan este lenguaje. Debido a que está muy extendido en el desarrollo de aplicaciones web existen infinidad de frameworks,los más conocidos son: Angular, Vue.js y React.

6.2.2 React

React [16] es una librería de Javascript creada por Facebook. Fue diseñada con el objetivo de facilitar el desarrollo del frontend de aplicaciones web mordernas SPA (Single Page Aplication). React fue creado para solventar uno de los principales problemas que tenía Facebook: El mantenimiento del código que gestiona los anuncios dentro de sus plataformas. Por lo tanto, ayuda a crear aplicaciones cuyos datos están en continuo cambio. Las principales características de react son las siguientes:

- Virtual DOM: Tiene un DOM propio, en vez de confiar solamente en el navegador. Esto permite que sea React el que determine los cambios en el DOM y determine como actualizar de manera eficiente el DOM del navegador.
- JSX: Es una sintaxis parecida a HTML. Hace el código más legible y escribirlo es una experiencia similar a HTML. Su uso no es necesario para poder usar React.

6.2.3 Bootstrap

Bootstrap [17] es una biblioteca para diseño de sitios y aplicaciones web. Contiene múltiples plantillas de diseño de elementos basados en HTML y CSS. Facilita el diseño e implementación de páginas y aplicaciones web. Sus principales características son:

- Compatibilidad con la mayoría de navegadores.
- Diseño “responsive”, que significa que es adaptable a todo tipo de pantallas incluidas móviles.

6.2.4 Visual Studio Code

Es un editor de código fuente desarrollado por Microsoft[18]. Está basado en Electron por lo que es multiplataforma. Dispone de una multitud de plugins que permiten desarrollar de manera eficiente. Soporta múltiples lenguajes entre ellos Javascript.

6.3 Tecnologías transversales

6.3.1 Docker

Docker[19] es una herramienta de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software.Docker permite empaquetar una aplicación y sus dependencias en un contenedor virtual. Utiliza el kernel de linux para sus contenedores, por lo tanto para su virtualización no requiere un sistema operativo independiente. Se basa en las funcionalidades del kernel y el aislamiento de recursos tales como: CPU, memoria, etc.. Desde 2016, Windows incluye un soporte del kernel de linux por lo que es posible correr contenedores docker en Windows. Gracias a esto se extendió de manera notoria la flexibilidad y portabilidad de estos contenedores tanto en instalaciones físicas, nubes públicas, nubes privadas, etc.. Gracias a esta abstracción del sistema operativo, permite crear contenedores mucho más livianos que las típicas máquinas virtuales. Debido a que solo recoge las dependencias que son necesarias.

6.3.2 GIT

Git [20] es un software distribuido de control de versiones. Actualmente es el software de control de versiones más usado. Sus características principales son:

- Los usuarios poseen un repositorio local.
- Ofrece buenos resultados en el desarrollo no lineal, gracias a la gestión de las ramas.
- Eficiente en la gestión de diferencias entre ficheros.

Capítulo 7

Desarrollo

7.1 Sprint 0: Formación

Al emplear tecnologías nuevas, es necesario llevar a cabo una fase de formación que consta de varias partes:

- Análisis y comprensión de las arquitecturas orientadas a microservicios. Principalmente en los patrones de diseño orientados a esta arquitectura[21].
- Después de estudiar y comprender los patrones de diseño, se hizo una búsqueda de las tecnologías que harían más sencillo el desarrollo. Se optó por las del stack tecnológico que ofrece Spring Cloud.
- Estudio de los componentes ofrecidos por Spring cloud. Este estudio se basó principalmente en el libro Spirng Microservices in Action [22].

7.2 Sprint 1: Análisis del proyecto

En este Sprint se desgranan las funcionalidades y se agrupan en diferentes microservicios. Este análisis es crucial para el proyecto, puesto que, una mala agrupación puede comprometer el rendimiento de la plataforma.

En primer lugar, para este análisis hay que buscar los microservicios core. Estos microservicios se encargan exclusivamente de su conjunto de funcionalidades y, por lo general, son microservicios pequeños que no dependen de otros para realizar sus operaciones. Las entidades fundamentales detectadas en este sistema son las siguientes: clientes, trabajadores, servicios y contrataciones. A mayores, un servicio pertenece a una categoría y tiene valoraciones.

7.2.1 Microservicios core

Usuarios

La plataforma que se está implementando consta de dos aplicaciones web diferentes para cada tipo de usuario (clientes y trabajadores). Los datos que deseamos de cada tipo de usuario son diferentes y, además, las necesidades de escalabilidad entre clientes y trabajadores no están relacionadas. Debido a todo esto, se decide separar en dos microservicios las gestiones de usuario; uno para clientes y otro para trabajadores. Las historias de usuario que conciernen a estos microservicios son las siguientes: US01, US02, US03, US04 y US05.

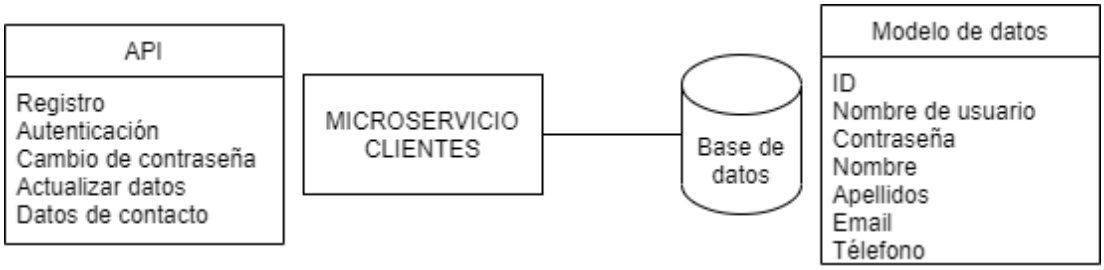


Figura 7.1: API y modelo de datos del microservicio clientes

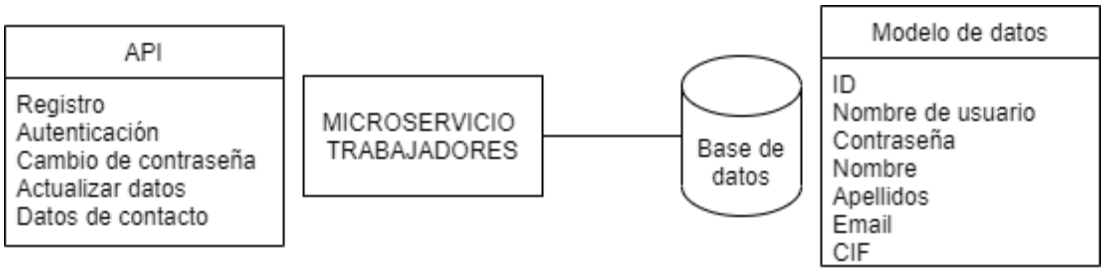


Figura 7.2: API y modelo de datos del microservicio trabajadores

Servicios

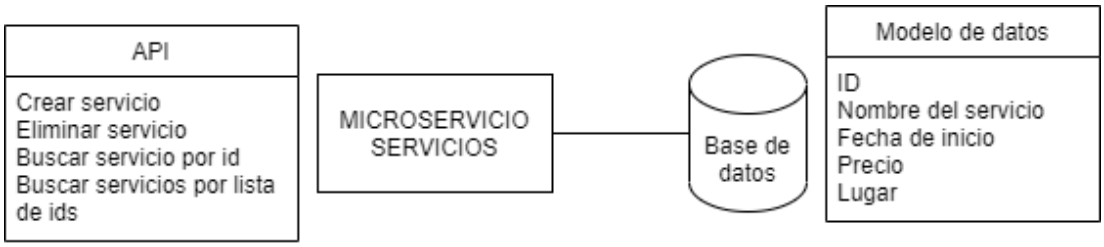


Figura 7.3: API y modelo de datos del microservicio core servicios

Contrataciones

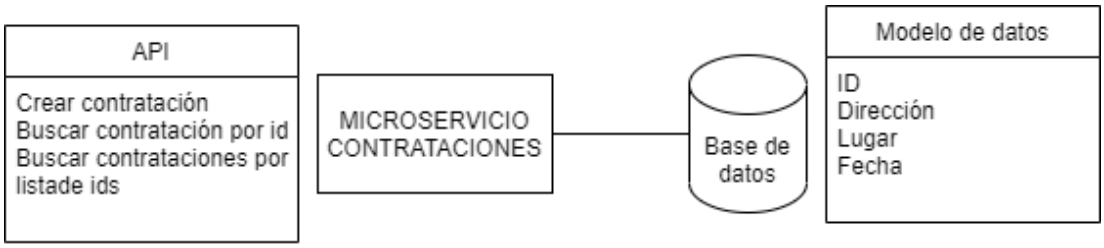


Figura 7.4: API y modelo de datos del microservicio core contrataciones

Categorías

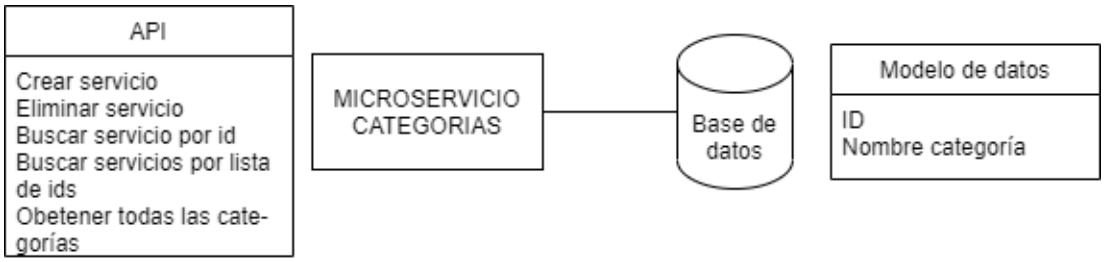


Figura 7.5: API y modelo de datos del microservicio categorías

Valoraciones

Figura 7.6: API y modelo de datos del microservicio valoraciones

7.2.2 Microservicios de composición

Detectar microservicios de composición es más complejo. Este tipo de microservicios no responden a las entidades, sino que lo hacen a las historias de usuario. Detectamos en las historias 3 microservicios de composición:

Servicios

Agrupa las historias de usuario US06, US07, US08 y US09. Su flujo queda descrito en la figura 7.7

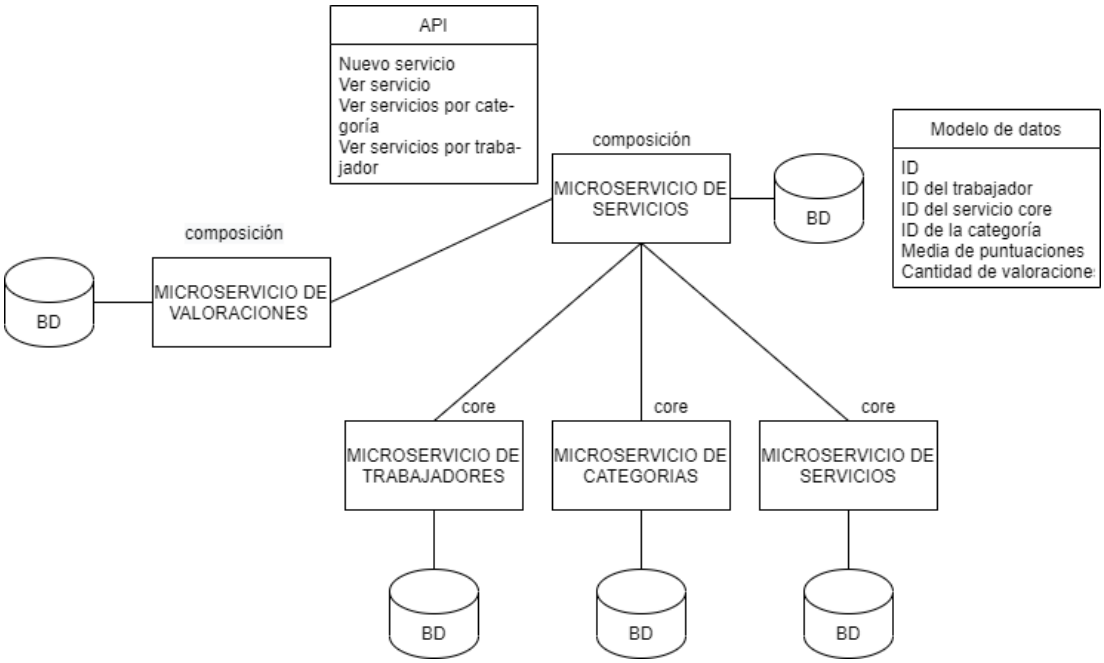


Figura 7.7: API y modelo de datos del microservicio servicios de composición

Contrataciones

Agrupa las historias de usuario US10, US11, US12, US14, US15 Y US16. Su flujo queda descrito en la figura 7.8

Valoraciones

Solo tiene una única historia de usuario, la US13. Su flujo queda descrito en la figura 7.9

Plataforma

Como se puede apreciar en las figuras anteriores, cada microservicio tiene su propia base de datos. De este modo, las dependencias entre los microservicios son exclusivamente las API REST de los microservicios con los que se comunica. En la siguiente figura7.17, se puede apreciar la arquitectura entera, incluyendo los componentes de esta. Se excluyó de la figura 7.17 las relaciones con las bases de datos para que sea más visual.

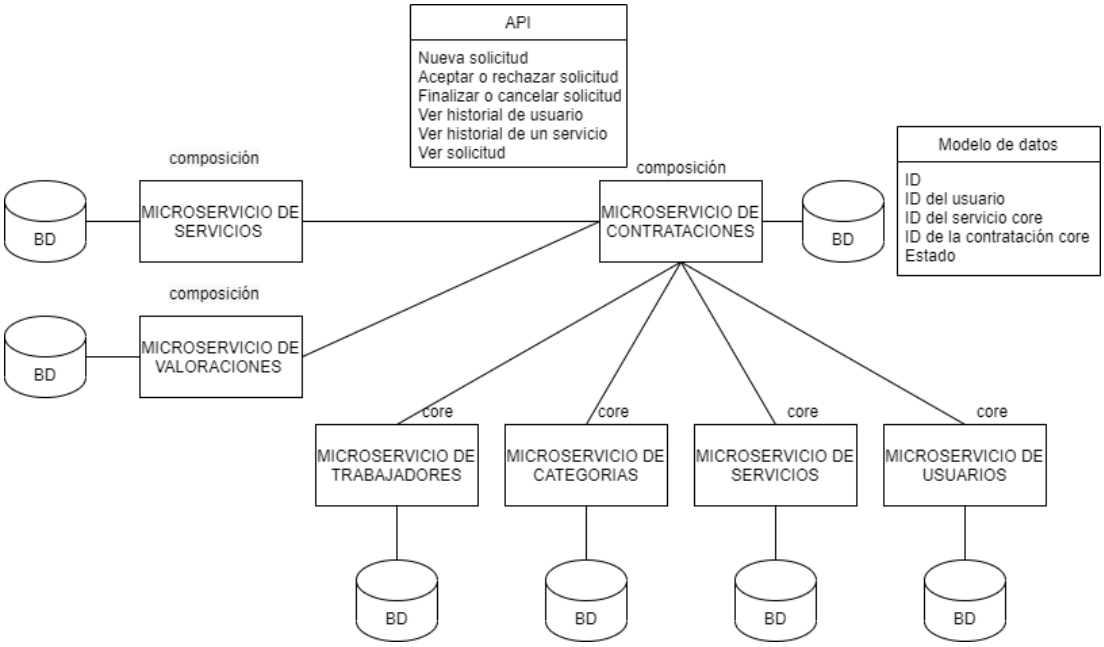


Figura 7.8: API y modelo de datos del microservicio contrataciones de composición

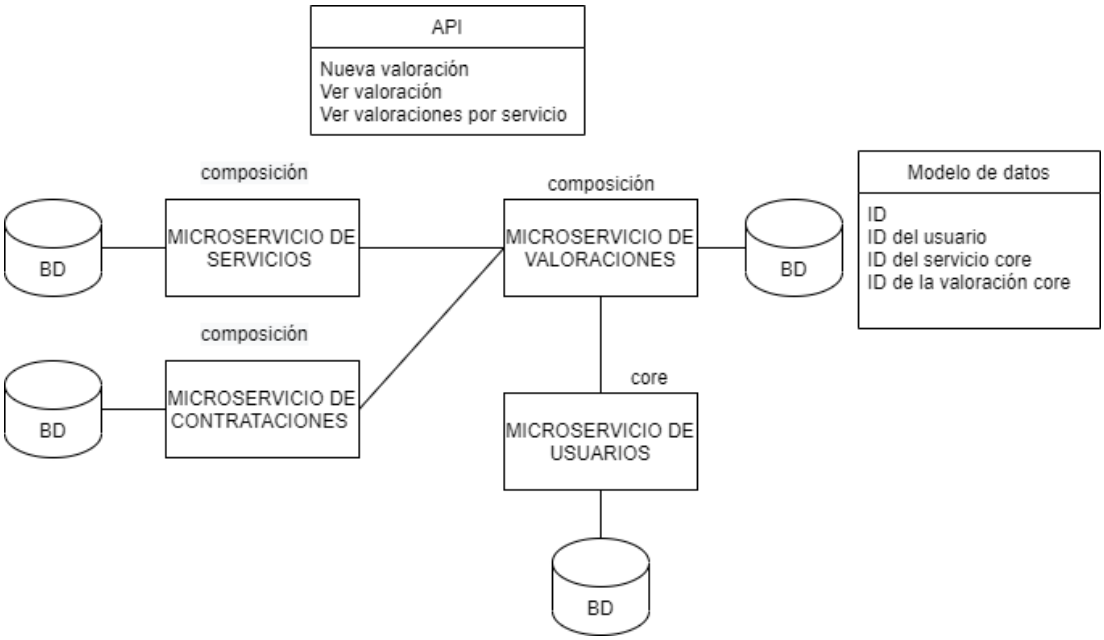


Figura 7.9: API y modelo de datos del microservicio valoraciones de composición

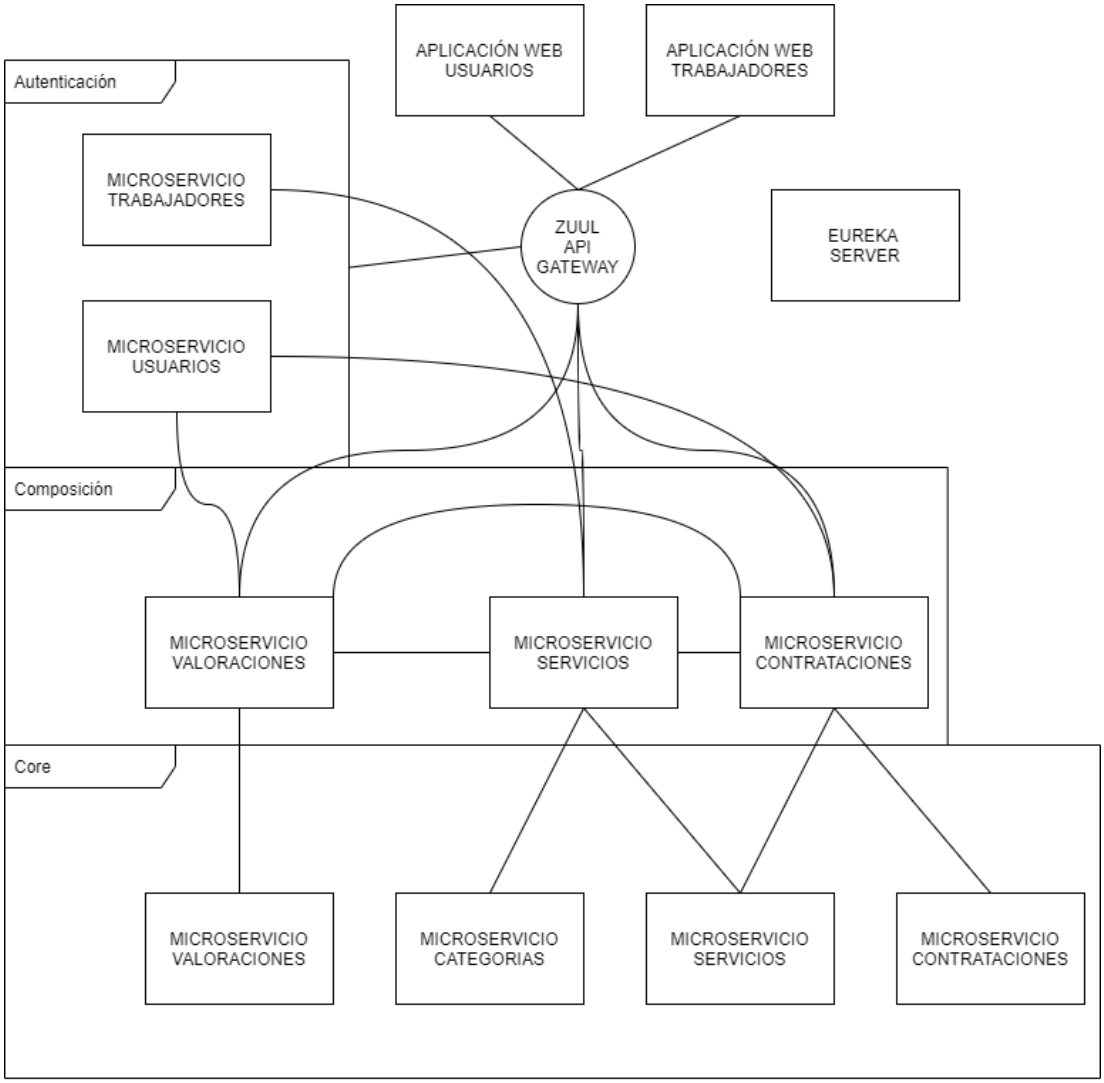


Figura 7.10: Visión global de la plataforma

7.2.3 Creación de microservicios

Cada microservicio se desarrollará en un nuevo proyecto independiente. La herramienta que usaremos será Spring inizializr [23]. Es una aplicación web de Spring, con la que se puede crear un proyecto Spring Boot y añadirle directamente las dependencias necesarias.

7.3 Sprint 2: Gestión de usuarios

En esta iteración se implementará la gestión de usuarios. Para esto, se crearán dos microservicios; uno para los clientes y otro para los trabajadores. Estos dos microservicios ofrecerán

las mismas funcionalidades pero, a su vez, tendrán un diferente modelo de datos.

7.3.1 Tareas

US01 - REGISTRO
Como usuario y trabajador sin autenticar quiero registrarme
TAREAS
Implementación de la operación registro en el servicio REST
Crear el formulario de registro en cada una de las aplicaciones web

Tabla 7.1: US01 - REGISTRO

US02 - Autenticación
Como usuario y trabajador sin autenticar quiero iniciar sesión
TAREAS
Implementación de la operación de autenticación en el servicio REST
Crear la vista de la autenticación

Tabla 7.2: US02 - Autenticación

US03 - Cerrar sesión
Como usuario y trabajador autenticado quiero cerrar sesión.
TAREAS
Implementar cerrar sesión en el frontend

Tabla 7.3: US03 - Cerrar sesión

US04 - Cambiar contraseña
Como usuario y trabajador autenticado quiero cambiar la contraseña.
TAREAS
Implementación de la operación cambio de contraseña en el servicio REST
Crear formulario de cambio de contraseña

Tabla 7.4: US04 - Cambiar contraseña

7.3.2 Implementación

En este sprint se comienza creando dos proyectos, el microservicio de usuarios y de trabajadores. Acto seguido, se comenzará con la implementación de las historias de usuario.

US05 - Actualización de datos
Como usuario y trabajador autenticado quiero actualizar mis datos
TAREAS
Implementación de la operación actualización de datos en el servicio REST
Crear el formulario de actualización de datos en cada una de las aplicaciones web

Tabla 7.5: US05 - Actualización de datos

US01 - Registro

Esta operación recibe los datos de usuario, valida que son correctos, almacena los datos en la base de datos y genera un Token (US02 - Autenticación).

US02 - Autenticación

Para su implementación se optó por el estándar JWT (JSON Web Token) por varios motivos, es fácilmente integrable con Spring Security y permite a la nube de microservicios ser stateless. Esto quiere decir que no se necesita mantener un registro de los tokens, es el cliente el que se encarga de enviarlo en cada petición. El usuario se autentica dando sus credenciales, el servicio comprueba si son válidas y en caso afirmativo emite el token JWT que será guardado por el cliente (normalmente gestionado por el navegador web).

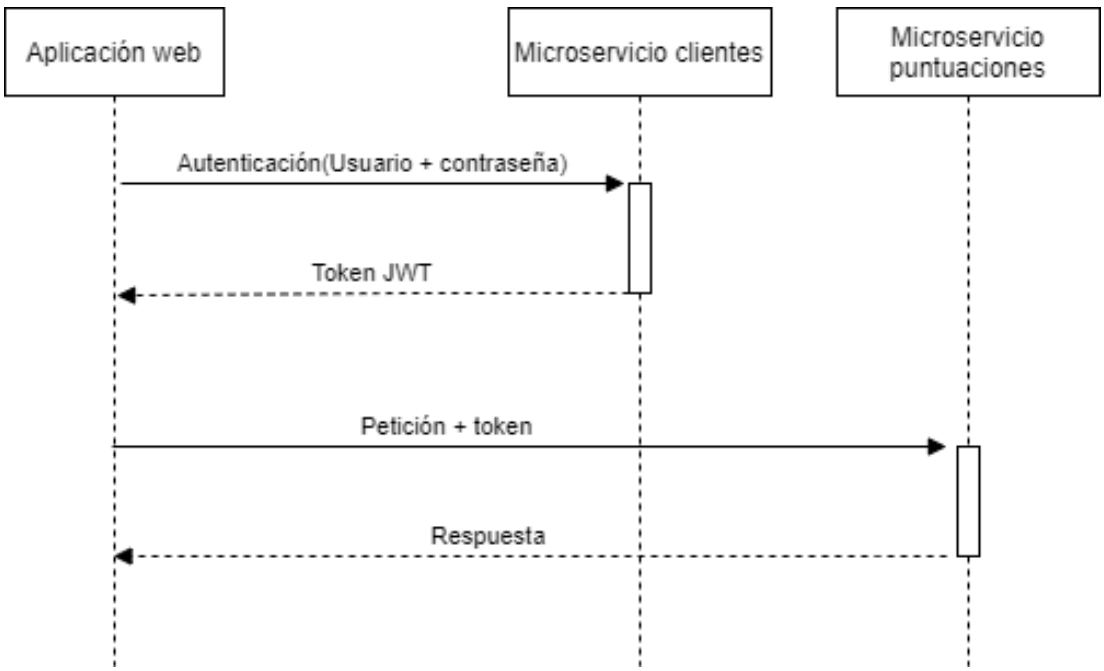


Figura 7.11: Autenticación con JWT

US03 - Cerrar sesión

Es una operación sencilla que no requiere implementación en el backend al no almacenar el token en este. De este modo, simplemente se tiene que borrar del estado de la aplicación web cuando el usuario pulse el botón cerrar sesión.

US04 - Cambiar contraseña

En esta operación se recibe la contraseña antigua y la nueva. El backend comprueba que la contraseña es válida y si no hay ningún error la cambia por la nueva, en caso de ser incorrecta, se mostrará un mensaje de error.

US05- Actualizar datos

Se mostrará en el frontend un formulario autocompletado con los datos existentes en base de datos. Una vez modificados, se validará en el propio frontend al pulsar el botón actualizar y se realizará la actualización en el backend.

7.4 Sprint 3: Componentes de la arquitectura

En esta iteración se implementarán y se configurarán los componentes necesarios para que el sistema tenga una arquitectura basada en microservicios. Los componentes que trataremos serán Eureka y Zuul.

7.4.1 Tareas

Eureka
TAREAS
Implementación del microservicio Eureka
Actualizar los microservicios de usuario y trabajador para integrarlos con eureka

Tabla 7.6: Eureka

Zuul
TAREAS
Implementación del microservicio Zuul
Crear filtro para extender en las peticiones JWT

Tabla 7.7: Zuul

7.4.2 Eureka

Server

La implementación de este microservicio es muy sencilla. Se crea un proyecto nuevo de SpringBoot, se añade en el POM la dependencia de Spring Cloud spring-cloud-starter-netflix-eureka-servery se le añade la anotación correspondiente.

```
@EnableEurekaServer
@SpringBootApplication
public class EurekaserverApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaserverApplication.class, args);
    }

}
```

Figura 7.12: Anotación Eureka server

Client

De la misma manera que con el servidor, se añade la anotación @EnableEruekaClient pertinente a todos los clientes que quieran ser descubiertos por eureka y además en la configuración de los proyectos clientes se facilitará la dirección de eureka para que este pueda indexarlos.

A partir de este sprint, todos los microservicios que se creen llevarán la anotación.

7.4.3 Zuul

La implementación de Zuul es similar a la de Eureka. Se crea un proyecto nuevo y se añade la dependencia spring-cloud-starter-netflix-zuul

La anotación de Zuul incluye el registro a Eureka, por lo tanto solo habrá que facilitar la dirección del Eureka.

Zuul permite implementar filtros. Su finalidad es ejecutar una pieza de código cada vez que llega una petición. Esto puede usarse para establecer reglas, enrutar de una manera determinada, etc.. En concreto, se implementará un filtro para extender en todas las peticiones el token JWT en caso de que existiera.

```java
@EnableZuulProxy
@SpringBootApplication
public class ZuulServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ZuulServerApplication.class, args);
    }

}
```

Figura 7.13: Anotación de Zuul

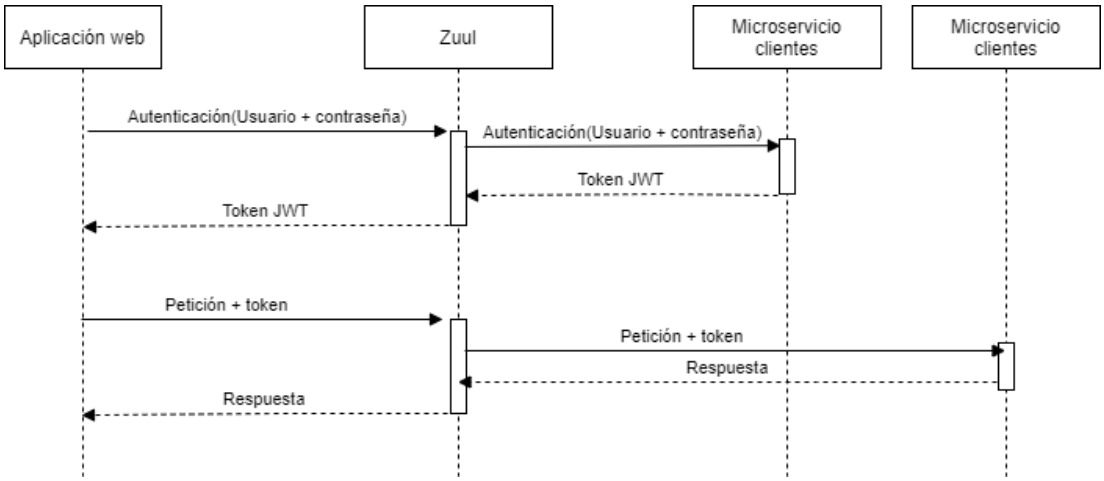


Figura 7.14: Autenticación JWT con Zuul

7.5 Spring 4: Gestión de servicios

En este sprint serán creados 3 microservicios: uno de categorías (core), uno core para los servicios y uno de composición para la gestión de los servicios.

7.5.1 Tareas

US06 - Crear Servicio
Como trabajador quiero poder crear un servicio para poder vender mi trabajo
TAREAS
Implementar crear servicio en el servicio rest del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Crear el formulario de crear servicio en el frontend

Tabla 7.8: US06 - Crear Servicio

US07 - Ver servicio
Como usuario y trabajador quiero poder ver el detalle de un servicio
TAREAS
Implementar ver servicio en el servicio rest del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Crear las vistas en las aplicaciones web

Tabla 7.9: US07 - Ver servicio

US08 - Buscar servicios por categoría
Como usuario quiero buscar los servicios, ordenados por puntuación, por categoría.
TAREAS
Implementación de la operación bulk en el servicio REST del microservicio Core
Integración del microservicio de composición con los microservicios necesarios
Implementación de la operación en el servicio REST del microservicio de composición
Crear la vista en el frontend

Tabla 7.10: US08 - Buscar servicios por categoría

US09 - Ver servicios propios
Como trabajador quiero ver todos los trabajos que ofrezco.
TAREAS
Implementación de la operación bulk en el servicio REST del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Crear la vista en el frontend

Tabla 7.11: US09 - Ver servicios propios

7.5.2 Implementación

Esta iteración comienza con la creación de 3 proyectos; uno por cada microservicio a implementar.

Una vez creados los microservicios core principales para abordar las historias de usuario escogidas para este Sprint, se procede a la implementación del microservicio de composición y a sus vistas en el frontend.

Para abordar las implementaciones de estas funcionalidades es importante implementar los Feing Client de los microservicios core de categorías y servicios. Las siguientes funcionalidades quedarán expuestas en el API REST del microservicio de composición.

US06- Crear servicio

Cuando esta operación sea invocada, se comprobará que la categoría existe, se llamará al microservicio core de servicios para que cree el servicio y guardará los datos necesarios en su base de datos.

US07 - Ver Servicio

En esta operación, se verá como se recogen los datos de varios microservicios y se le proporcionan al usuario. Cuando esta petición llega al microservicio de composición, este busca las referencias y mediante los correspondientes feign client obtiene la información, la monta y se la devuelve al usuario. Este flujo queda refljeado en la figura 7.15

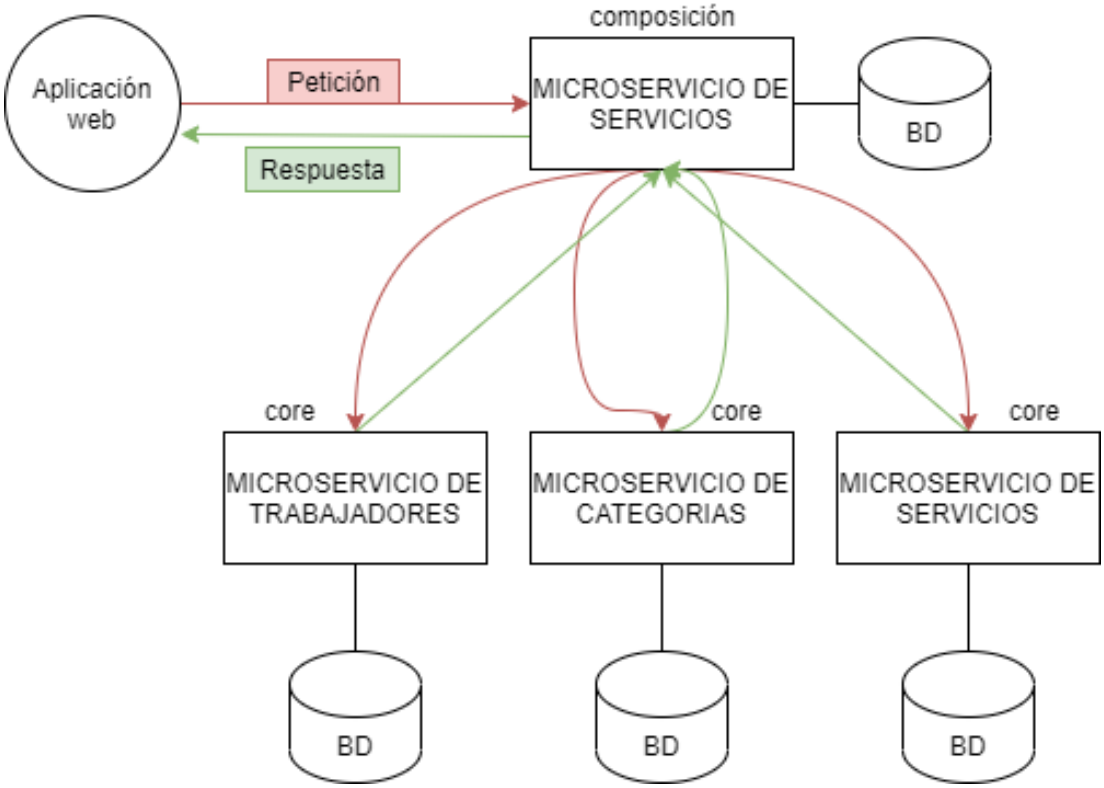


Figura 7.15: Flujo de comunicaciones

US08 y US09

Estas dos funcionalidades son prácticamente iguales que la anterior, con la diferencia de que se usan listas paginadas de servicios. El cambio principal es que se emplearán operaciones Bulk contra los microservicios core para conseguir un mejor tiempo de respuesta.

Las operaciones bulk son empleadas para gestionar datos de manera conjunta en lugar de hacerlo de manera individual. Para ejemplificar esto usaremos la figura 7.15.

La figura 7.15 representa el flujo de una búsqueda para un único servicio(US07). Por lo tanto, una búsqueda cuyo resultado sea una lista de servicios (US08 y US09) deberá ejecutar ese flujo por cada servicio de la lista.

Con una operación bulk, en lugar de pedir individualmente cada servicio, se hace mediante una lista. Como consecuencia, en vez de realizar una petición por cada servicio, se realiza una única petición para todos los servicios.

7.6 Sprint 5: Gestión de contrataciones

En este sprint serán creados 2 microservicios; uno core para las contrataciones y uno de composición para la gestión de las contrataciones.

7.6.1 Tareas

US10 - Solicitar Contratación
Como usuario autenticado quiero solicitar la contratación de un servicio
TAREAS
Implementación de la operación en el servicio REST del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Crear el botón contratar en la vista de US07 y el formulario de contratación en el frontend

Tabla 7.12: US10 - Solicitar Contratación

US15 - Aceptar y rechazar solicitud de servicio
Como trabajador quiero poder aceptar o rechazar la solicitud de un servicio.
TAREAS
Implementación de la operación en el servicio REST del microservicio de composición
Crear la vista en el frontend

Tabla 7.13: US15 - Aceptar y rechazar solicitud de servicio

7.6.2 Implementación

Después de implementar el microservicio core, es el turno del microservicio de composición que expondrá en su API REST las funcionalidades:

US16 - Finalizar o cancelar un servicio aceptado
Como trabajador, una vez acepte el servicio, quiero poder cancelar o dar por finalizado el servicio aceptado.
TAREAS
Implementación de la operación en el servicio REST del microservicio de composición
Crear la vista en el frontend

Tabla 7.14: US16 - Finalizar o cancelar un servicio aceptado

US10 - Solicitar contratación

Cuando esta operación sea llamada, se hará uso del feign client contra el microservicio core para crear la solicitud y, acto seguido, guardará los datos pertinentes.

US15 y US16

Cuando alguno de estos dos casos de uso son llamados, el microservicio de composición cambiará el estado de esa solicitud. Esta operación no necesita de ninguna comunicación con otros microservicios

7.7 Sprint 6: Visualización de contrataciones

En esta iteración no hará falta crear nuevos microservicios, la funcionalidad principal recae en el microservicio de composición que gestiona las contrataciones.

7.7.1 Tareas

US11 - Ver historial de contrataciones de un usuario
Como usuario quiero ver un historial de contrataciones filtradas por estado
TAREAS
Implementación de la operación bulk en el servicio REST del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Crear la vista en el frontend

Tabla 7.15: US11 - Ver historial de contrataciones de un usuario

7.7.2 Implementación

Estas funcionalidades son prácticamente idénticas a las implementadas en el sprint 4. Sin embargo, afectan a diferentes microservicios y son expuestas por el microservicio de composición de contrataciones.

US12 - Ver detalle de contratación
Como usuario quiero ver el detalle de una solicitud de contratación.
TAREAS
Implementación de la operación en el servicio REST del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Crear la vista en el frontend

Tabla 7.16: US12 - Ver detalle de contratación

US14 - Ver historial de contrataciones de un servicio
Como trabajador quiero ver el historial de contrataciones filtradas por estado.
TAREAS
Implementación de la operación bulk en el servicio REST del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Crear la vista en el frontend

Tabla 7.17: US14 - Ver historial de contrataciones de un servicio

US12 - Ver detalle de contratación

Esta operación se implementó de manera similar a la descrita en la figura 7.15. Cuando se realiza esta operación, se buscan los datos de referencia en el microservicio de composición y se recoge la información necesaria mediante los Feign client de los microservicios de contrataciones (core), servicios (composición) y categoría.

US11 y US14

Estas operaciones buscarán las referencias (usuario o servicios) y se dispondrán, mediante los feign clients anteriores, a realizar las bulk operations para su posterior montaje y entrega al usuario.

7.8 Sprint 7: Puntuaciones

En este Sprint será necesario la creación de dos microservicios: uno core para las puntuaciones y uno de composición para gestionarlas.

US13 - Puntuar servicio realizado
Como usuario quiero poder puntuar un servicio realizado.
TAREAS
Implementación de la operación en el servicio REST del microservicio Core
Integrar la operación en el microservicio de composición y exponerla en el servicio Rest
Integrar con los microservicios necesarios
Crear la vista en el frontend

Tabla 7.18: US13 - Puntuar servicio realizado

7.8.1 Tareas

7.8.2 Implementación

US13 - Puntuar servicio realizado

Antes de comenzar el desarrollo de esta funcionalidad es necesario hacer un cambio en el modelo de datos del microservicio de composición de servicios. Este cambio será añadir los parámetros media y puntuaciones totales, ya que, desde ese microservicio se gestionará la puntuación media de los servicios. Además de ese cambio en el modelo, se expondrá una funcionalidad a través del API REST que permitirá actualizar esta información.

Cada vez que un usuario realice una puntuación, se hará uso del feign client del microservicio core de puntuaciones para guardarla y, a su vez, se guardarán los datos necesarios en el microservicio de composición y se actualizará la puntuación media del servicio mediante el feign client contra el microservicio de composición de servicios que se ha mencionado anteriormente.

Buscar puntuación por ID

En este punto, igualmente se ve modificada otra funcionalidad que estaba incompleta hasta que se implementó la puntuación de servicios. La funcionalidad afectada es US12 por parte del trabajador. Por tanto, se modificará el servicio de composición de contrataciones para que incluya en su respuesta, la puntuación (si la hubiere) del servicio prestado.

Buscar puntuaciones por lista de IDs

La funcionalidad afectada es US07. Se modificará el servicio de composición de servicios para que incluya en su respuesta, la lista paginada de las valoraciones ordenadas por fecha.

7.9 Docker

Cada microservicio, componentes y base de datos serán ejecutados en contenedores Docker.

Para los microservicios y componentes que son implementados en Java, se partirá de la imagen pública openjdk:11 (alojada en el repositorio oficial de docker, llamado dockerhub [24]). Además se creará una nueva imagen añadiendo el ejecutable (.jar) con la ayuda de un dockerfile[25]. Un dockerfile es un archivo de texto plano que contiene instrucciones para crear una imagen.

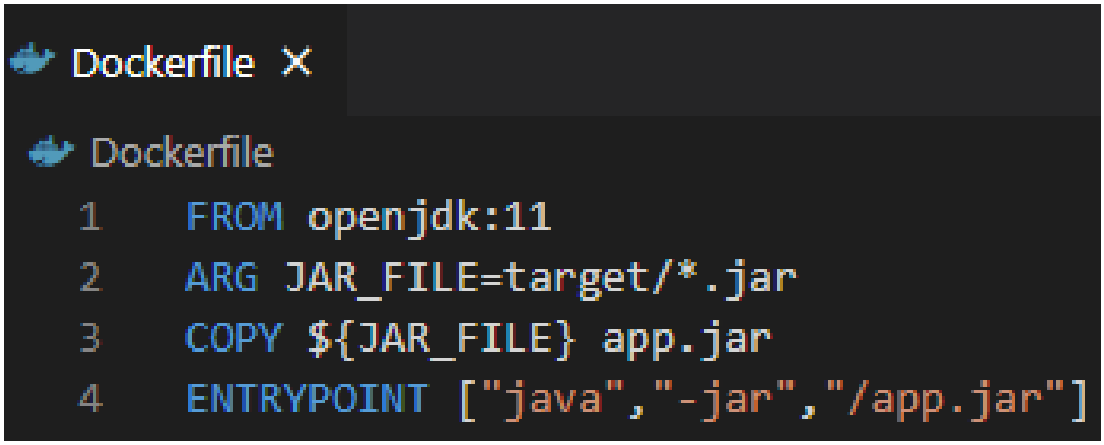


Figura 7.16: Ejemplo de dockerfile

Para las bases de datos, se usará directamente la imagen oficial de MongoDB alojada en dockerhub.

Para ejecutar la nube de microservicios se hará uso de docker-compose[26], que es una herramienta de docker para correr de manera simultanea varias imagenes. Esto será posible, al registrar en un fichero .yml las imagenes que se dean ejecutar.

docker-compose.yml

```yaml
version: "2"
services:
  eureka:
    image: eurekaserver
    container_name: "eureka"
    ports:
      - 9091:9091
  microserviceworker:
    image: microserviceworker
    container_name: "microserviceworker"
    links:
      - mongodbwork
      - eureka
  mongodbwork:
    image: 'mongo:latest'
    container_name: "mongodbwork"
```

Figura 7.17: Ejemplo de docker-compose.yml

Capítulo 8

Conclusiones y trabajo futuro

8.1 Conclusiones

El objetivo principal de este proyecto es experimentar con las arquitecturas orientadas a microservicios y aprender lo máximo posible de estas. Se considera que el objetivo ha sido alcanzado.

Desde el punto de vista del aprendizaje, considero que este proyecto me dió una base para poder iniciarme en el ecosistema del desarrollo basado en microservicios.

En lo personal, este proyecto me ha ayudado a mi desarrollo como ingeniero. En poco tiempo, he conseguido aprender sobre una nueva arquitectura, sus fundamentos tecnológicos y llevarla a cabo. Considero que en el mundo del software adaptarse de manera rápida a nuevos entornos es de las habilidades más importantes y las lecciones aprendidas enfrentándome a este reto sin duda alguna serán de utilidad en mi futuro laboral.

Como conclusión final, me siento más que satisfecho con este proyecto y siento que los objetivos fueron alcanzados

8.2 Trabajo futuro

Desde el punto de vista de la plataforma

- Rol administrador: Crear una aplicación web para administrar, tanto las categorías como a los trabajadores.
- Reclamaciones: crear un sistema de reclamaciones para que el cliente, si no está satisfecho con el servicio, pueda reclamar al trabajador.
- Perfiles de usuarios y trabajadores: Con los perfiles se crearía un entorno más social, que genera mayor confianza en cliente como en trabajadores.

Desde el punto de vista de la arquitectura

- Kubernetes: Investigar y familiarizarse con kubernetes, la plataforma de gestión de contenedores más popular .[27]
- Despliegue: Considero interesante poder probar el despliegue de la nube de microservicos en provedores públicos como AWS[28] o Azure[29].
- También sería interesante poder añadir más patrones de diseño a la nube de microservicios, como por ejemplo, la comunicación basada en eventos y la gestión distribuida de transacciones.

Apéndices

Apéndice A

Glosario de términos

API

Acrónimo de Application Programming Interface.Conjunto de funciones que ofrece una aplicación.

Backend

Parte invisible de una aplicación, donde se recoge la lógica de esta.

Frontend

Parte visible de una aplicación, es con la que se interactúa directamente.

Diagrama de Gantt

Es una gráfica donde se representa una planificación de tarear en un tiempo determinado.

Mockup

Maqueta a escala del diseño visual de un sistema.

NoSql

Concepto que engloba a las bases de datos que no comparten el modelo relacional.

Framework

Marco o entorno de trabajo. En software, normalmente un conjunto de librerías.

Bibliografía

[1] "Página web de microservicios." [En línea]. Disponible en: https://microservices.io

[2] "Eureka server." [En línea]. Disponible en: https://github.com/Netflix/eurekar

[3] "Spring cloud." [En línea]. Disponible en: https://spring.io/cloud

[4] "Netflix open source." [En línea]. Disponible en: https://github.com/Netflix

[5] "Hystrix." [En línea]. Disponible en: https://github.com/Netflix/Hystrix/wiki

[6] "Zuul server." [En línea]. Disponible en: https://github.com/Netflix/zuul

[7] "Spring security oauth." [En línea]. Disponible en: https://spring.io/projects/spring-security-oauth

[8] "Feign." [En línea]. Disponible en: https://github.com/OpenFeign/feign

[9] "Guía de scrum." [En línea]. Disponible en: https://www.scrumguides.org/

[10] "Java." [En línea]. Disponible en: https://www.oracle.com/es/java/technologies/java-ee-glance.html

[11] "Spring framework." [En línea]. Disponible en: https://spring.io/

[12] "Apache maven." [En línea]. Disponible en: https://maven.apache.org/

[13] "Mongodb." [En línea]. Disponible en: https://www.mongodb.com/es

[14] "Eclipse ide." [En línea]. Disponible en: https://www.eclipse.org/

[15] "Javascript." [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[16] "React." [En línea]. Disponible en: https://es.reactjs.org/

[17] "Bootstrap." [En línea]. Disponible en: https://getbootstrap.com/

[18] "Visual studio code." [En línea]. Disponible en: https://code.visualstudio.com/

[19] "Docker." [En línea]. Disponible en: https://www.docker.com/

[20] "Git." [En línea]. Disponible en: https://git-scm.com/

[21] "Microservice architecture." [En línea]. Disponible en: https://microservices.io/

[22] J. Carnell, *Spring Microservices in Action*, 1st ed. Manning, 2017.

[23] "Spring initializr." [En línea]. Disponible en: https://start.spring.io/

[24] "Docker hub." [En línea]. Disponible en: https://hub.docker.com/

[25] "Dockerfile." [En línea]. Disponible en: https://docs.docker.com/engine/reference/builder/

[26] "Docker compose." [En línea]. Disponible en: https://docs.docker.com/compose/

[27] "Kubernetes." [En línea]. Disponible en: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

[28] "Aws." [En línea]. Disponible en: https://aws.amazon.com/es/

[29] "Azure." [En línea]. Disponible en: https://azure.microsoft.com/es-es/