UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
[GRADUADA/O EN INGENIERÍA DEL SOFTWARE]

Aplicación técnica de arquitectura basada en microservicios con GraphQL y tecnologías serverless en cloud

Technical application of microservice architecture with GraphQL and serverless technologies in cloud

Realizado por
Rafael Pernil Bronchalo

Tutorizado por
Inmaculada Ayala Viñas

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2023

Fecha defensa: Por determinar

Resumen

Este TFG surge de una idea para una aplicación destinada a compartir determinados recursos entre grupos de personas, creando colas de espera y notificando a los usuarios cuando una vacante de ese recurso está disponible. Un ejemplo práctico podría ser compartir una cuenta de Netflix entre diez personas cuando la cuenta solo permite cinco visualizaciones simultáneas. Pero esta idea se expande a cualquier tipo de recurso que se pueda compartir como listas de espera, personas dentro de una sala o cualquier idea que tenga el
usuario.

Este TFG está basado en la línea 22-16 cuyo concepto es “Implantación de sistemas con arquitectura de microservicios. Despliegue y monitorización de aplicaciones basadas en microservicios. Desarrollo de software basado en microservicios. Tecnologías Serverless, Docker, Kubernetes.”. Por tanto, el dominio de la aplicación es desarrollo web con arquitectura de microservicios.

Palabras clave: **Microservicios, Serverless, Kubernetes, Aplicación Web**

Abstract

This TFG arises from an idea for an application destined to share certain resources between groups of people, creating queues and notifying users when a slot of that resource is available. A practical example could be sharing a Netflix account between ten people when the account only allows five simultaneous views. But this idea expands to any type of resource that you can share such as waiting lists, people in a room or any idea that the user has. This TFG is based on line 22-16 whose concept is “Implementation of systems with microservices architecture. Deployment and monitoring of applications based on microservices. Microservices-based software development. Serverless, Docker, Kubernetes technologies.” Therefore, the domain of the application is web development with microservices architecture.

Keywords: **Microservices, Serverless, Kubernetes, Web Application**

Índice

1

Introducción

1.1 Motivación

La motivación de este trabajo surge de una idea para una aplicación, que se presenta como una opción interesante para combinar con esta línea de TFG centrada en el desarrollo de arquitecturas de microservicios utilizando Docker y tecnologías "serverless" como OpenFaaS. La aplicación por desarrollar se adapta muy bien al concepto serverless debido a que las acciones que realiza el usuario son atómicas y no se necesita procesamiento permanente con tareas en segundo plano por parte del backend. La aplicación escala horizontalmente según aparezcan más usuarios y salvo para métricas y notificaciones de sugerencias, es siempre el usuario el que inicia o reactiva los casos de uso.

A continuación, se adjunta un diagrama de contexto basado en el modelo C4 de diagramas de arquitectura de software (Brown, 2011):

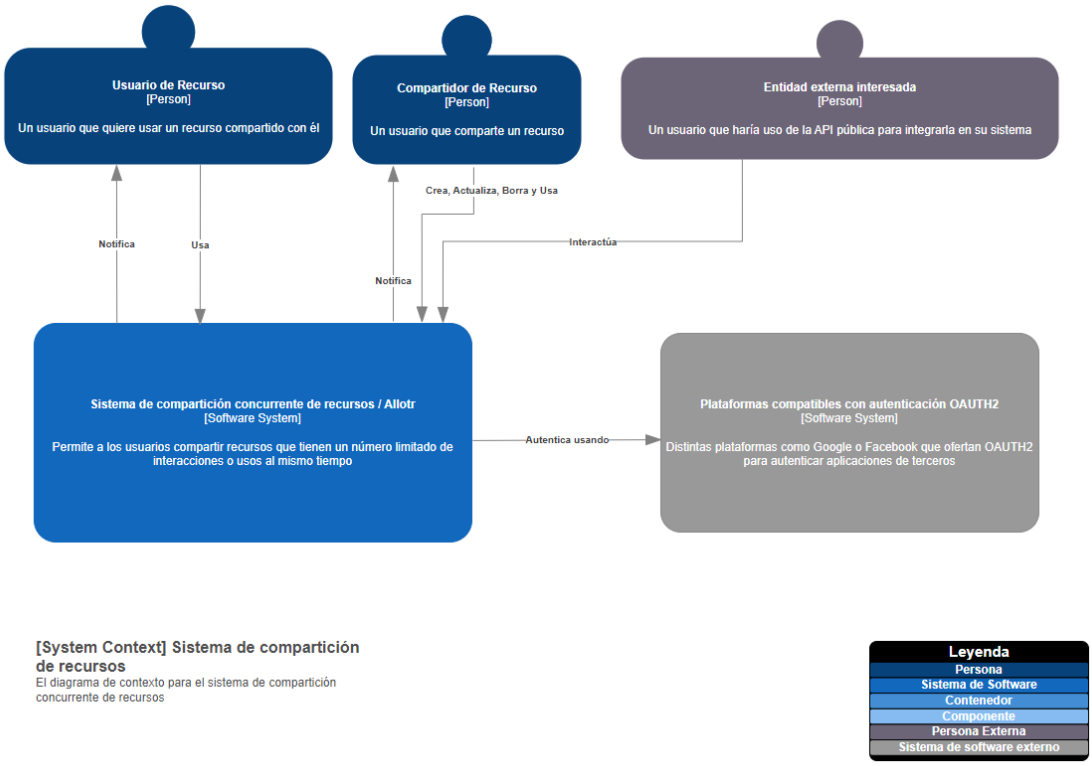


Figura 1.1 Diagrama de contexto de la aplicación propuesta; basado en el modelo C4 de diagramas de arquitectura de software (Brown, 2011)

1.2 Objetivos

El TFG consistirá en el diseño, implementación y prueba de una aplicación web y la documentación y publicación de una API GraphQL para compartir determinados recursos entre grupos de personas basado en el siguiente conjunto de tecnologías:

Base de datos: MongoDB
Middleware/Servicios web: Node.js con TypeScript, Express, uWebSockets.js, GraphQL y librerías que implementan WebSockets, integrado con la base de datos usando el controlador nativo de MongoDB y aplicando una arquitectura serverless con OpenFaaS
Frontend: React

Todas las capas tendrán su proyecto correspondiente Docker-izado que se desplegará usando Kubernetes, y en aquellos casos donde aplique, combinado con OpenFaaS.

La aplicación será desplegada usando su certificado SSL correspondiente y el frontend y middleware estarán segurizados usando OAUTH2 de Google para minimizar la recolecta de datos personales ya que no son necesarios para el perfecto desempeño de la aplicación.

Las funcionalidades de la aplicación a alto nivel son las siguientes:

• **Registro de usuario:** Si el usuario no tenía cuenta, se creará un usuario basado en la información que se puede solicitar y/o obtener a través de la autenticación por OAUTH2, una vez hecho eso, se realizará el proceso de inicio de sesión

• **Inicio de sesión de usuario:** El usuario accederá a la pantalla principal de la aplicación y podrá utilizar la aplicación al completo

• **Visualización de lista de ítems compartidos contigo:** En esta pantalla se verán todos los recursos que otros usuarios estén compartiendo contigo en una lista, se podrá

observar al menos el nombre del recurso, un contador de usuarios activos respecto al máximo de usuarios simultáneos y un bloque que muestre de forma clara e intuitiva el estado y permita interactuar en algunas situaciones:

-Recurso disponible con al menos una vacante libre y no solicitado por el usuario: Permitiría reservar una vacante que se daría inmediatamente al usuario.

-Recurso disponible con al menos una vacante libre y ya solicitado previamente por el usuario: Este estado se notificará mediante dos vías: Notificaciones Push y mediante un diálogo emergente en pantalla recibido mediante una Suscripción de GraphQL como conexión bidireccional implementada en WebSockets

-Recurso no disponible y ya solicitado por el usuario: Informa que ya has reservado y no puedes hacer nada más hasta que se liberen las suficientes vacantes como para que llegue tu puesto en la cola para utilizar el recurso

-Recurso no disponible y no solicitado por el usuario: Permitiría entrar en una lista de espera para obtener una vacante aleatoria cuando se encuentre disponible. La cola es una FIFO y según entren irán saliendo.

• **Creación de recurso**: En esta pantalla se crean los recursos a compartir añadiendo al menos el nombre, descripción y los usuarios entre los que se va a compartir

• **Visualización/Edición de recurso**: En esta pantalla puedes ver quién tiene bloqueado el recurso en detalle.
Si eres administrador del recurso, podrías añadir personas entre las que se compartirá el recurso, editar o eliminar el ítem. En la lista de usuarios, el administrador podría asignar roles a cada usuario (en principio solo administrador, es decir, el administrador puede hacer administradores a otros usuarios). La eliminación del recurso tendrá su pantalla de confirmación.

• **Ajustes**: En esta pantalla se podrá borrar la cuenta entera si así lo desea el usuario, esta operación eliminaría los recursos creados por él y se desvincularía de los que han creado otros usuarios y éste se incluye. Se añadirá una pantalla de confirmación ya que las implicaciones de borrar la cuenta son severas.

• **Sugerencias de usuario**: Se analizará el uso del usuario para poder mostrar notificaciones que se adapten a las costumbres del usuario

Como requisitos no funcionales, está la seguridad de la aplicación, autenticación de peticiones.
La aplicación web será responsiva y se adaptará a ordenadores y móviles, funcionando como una aplicación web progresiva. Por otro lado, la documentación de la API será clara y detallada, mostrando explícitamente el objetivo de cada función y la semántica de cada dato de entrada y de retorno.

1.3 Estructura de la memoria

- Estado del Arte: Investigación de las tecnologías a utilizar y sus implicaciones
- Metodología de trabajo
- Diseño: Requisitos, diseño del modelo de datos, casos de uso, casos de prueba (usando TDD), arquitectura…
- Implementación: Programación de todas las capas de la aplicación
- Pruebas: Implementación de las pruebas funcionales
- Documentación: Redactado de documentación de uso de la API en GraphQL

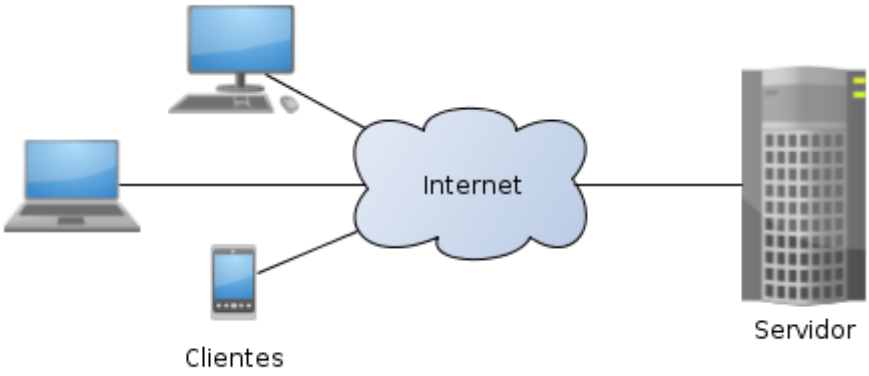
2

Estado del arte

Para este proyecto se han involucrado un amplio abanico de tecnologías que deben integrarse para conformar una arquitectura escalable.

Como punto de partida, antes de presentar el propio anteproyecto, se planteó una arquitectura web de cliente-servidor. El cliente se ofrece al usuario para que interactúe con la aplicación y el servidor provee la lógica y funcionalidad interna que usa el cliente.

> "La **arquitectura cliente-servidor** es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta."
> (Cliente-servidor - Wikipedia, la enciclopedia libre, s.f.)

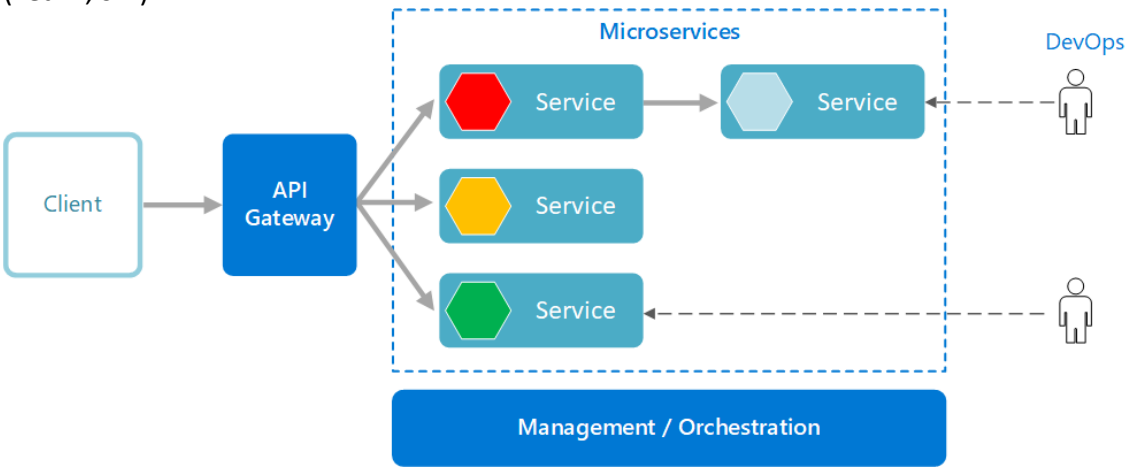


(Neves, 2013)

Esta arquitectura sirve como base inicial, pero para obtener una solución escalable es necesario dividir el código a nivel de servidor para poder escalar bajo demanda y reducir la complejidad.

Como solución, se presenta una arquitectura de microservicios:

> "Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de servicio es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado."
> (Learn, s.f.)



(Learn, s.f.)

Una arquitectura de microservicios proporciona un terreno sólido sobre el que añadir el resto de las tecnologías que presentaré a continuación, ordenadas desde lado servidor hasta lado cliente.

La parte fundamental de cualquier aplicación son los datos, la forma de accederlos y guardarlos, es decir, una **base de datos**. Para este proyecto se analizó si utilizar una

base de datos **relacional** o **no relacional**. Como parte de la investigación, se exponen las diferencias a continuación:

> "El principio de las bases de datos relacionales se basa en la organización de la información en trozos pequeños, que se relacionan entre ellos mediante la relación de identificadores.
> En el ámbito informático se habla mucho de **ACID**, cuyas siglas vienen de las palabras en inglés: **atomicidad, consistencia, aislamiento y durabilidad**. Son propiedades que las bases de datos relacionales aportan a los sistemas y les permiten ser **más robustos y menos vulnerables** ante fallos.
>
> La base de datos relacional más usada y conocida es MySQL junto con Oracle, seguida por SQL Server y PostgreSQL, entre otras.
>
> Las bases de datos no relacionales son las que, a diferencia de las relacionales, no tienen un identificador que sirva de relación entre un conjunto de datos y otros. Como veremos, la información se organiza normalmente mediante documentos y es muy útil cuando **no tenemos un esquema exacto de lo que se va a almacenar**.
>
> La indiscutible reina del reciente éxito de las bases de datos no relacionales es MongoDB seguida por Redis, Elasticsearch y Cassandra." (Lafuente, 2018)

Una vez claras las ventajas e inconvenientes de cada alternativa, la balanza se inclinó hacia el paradigma no relacional por dos razones: **rapidez de acceso** y **escalabilidad en el modelo de datos.**

El modelo de datos está adaptado a este paradigma como se verá en posteriores capítulos y podemos obtener la información que mostrar al usuario con una cantidad mínima de transformaciones. El estado íntegro del recurso está en un solo registro asegurando la atomicidad y aislamiento en la escritura, aspecto clave para gestionar estados del recurso a tiempo real y evitar condiciones de carrera sin sacrificar escalabilidad. Por otra parte, la base de datos debe permitir ser expandida para entidades externas que quieran extender la funcionalidad actual para su sistema.

Dentro del paradigma no relacional, la base de datos utilizada es **MongoDB** y el modelo de datos ha sido diseñado siguiendo el patrón de incrustación de datos o "embedding" en vez de usar referencias. A continuación, se citan las ventajas clave para este proyecto:

Incrustación para Atomicidad y Aislamiento

Otra preocupación que pesa a favor de la incrustación es el deseo de atomicidad y aislamiento en la escritura de datos. Cuando actualizamos datos en nuestra base de datos, queremos asegurarnos de que nuestra actualización sea exitosa o fracasa por completo, nunca tenga un "éxito parcial" y que cualquier otro lector de base de datos nunca vea una operación de escritura incompleta. Las bases de datos relacionales logran esto mediante el uso de transacciones de declaraciones múltiples. Por ejemplo, si queremos hacer DELETE a Jenny de nuestra base de datos normalizada, podríamos ejecutar un código similar al siguiente:

```
BEGIN TRANSACTION;
DELETE FROM contacts WHERE contact_id=3;
DELETE FROM numbers WHERE contact_id=3;
COMMIT;
```

El problema con el uso de este enfoque en MongoDB es que MongoDB está diseñado sin transacciones de documentos múltiples. Si intentáramos eliminar a Jenny de nuestro esquema MongoDB "normalizado", necesitaríamos ejecutar el siguiente código:

```
db.contacts.remove({'_id': 3})
db.numbers.remove({'contact_id': 3})
```

¿Por qué no hay transacciones?
MongoDB fue diseñado desde cero para ser fácil de escalar a múltiples servidores distribuidos. Dos de los mayores problemas en el diseño de bases de datos distribuidas son las operaciones distribuidas de unión y transacción. Ambas operaciones son complejas de implementar y pueden producir un rendimiento deficiente o incluso un tiempo de inactividad en caso de que no se pueda acceder a un servidor. Al "apostar" en estos problemas y no admitir uniones o transacciones de múltiples documentos, MongoDB ha podido implementar una solución de fragmentación automática con características de rendimiento y escalado mucho mejores de las que normalmente tendría si tuviera que tomar uniones relacionales y transacciones en cuenta.
Con este enfoque, presentamos la posibilidad de que Jenny se elimine de la colección de contactos pero que sus números permanezcan en la colección de números. También existe la posibilidad de que otro proceso lea la base de datos después de que se haya eliminado a Jenny de la colección de contactos, pero antes de que se hayan eliminado sus números. Por otro lado, si usamos el esquema incrustado, podemos eliminar a Jenny de nuestra base de datos con una sola operación:

```
db.contacts.remove ({'_ id': 3})
```

(Copeland, 2013)- Traducción por Google

Una vez determinada la base de datos a utilizar, el siguiente aspecto clave para la arquitectura de microservicios y tecnología con la que el servidor expondrá la información para el cliente. En este aspecto, inicialmente, antes de presentar el anteproyecto se barajó implementar una API REST

> "La transferencia de estado representacional (en inglés representational state transfer) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo."
> (Transferencia de Estado Representacional - Wikipedia, s.f.)

Pero tras un análisis exhaustivo de las opciones disponibles hoy en día, se decidió usar **GraphQL**

> "**GraphQL** es un lenguaje de consulta y manipulación de datos para APIs, y un entorno de ejecución para realizar consultas con datos existentes. GraphQL fue desarrollado internamente por Facebook en 2012 antes de ser liberado públicamente en 2015"
> (GraphQL - Wikipedia, la enciclopedia libre., s.f.)

¿Cuáles son las ventajas de GraphQL respecto a REST y qué ofrece?
GraphQL está diseñado para poder crear APIs declarativas donde el cliente especifica qué datos necesita exactamente, evitando crear múltiples endpoints con estructuras de datos distintas según la necesidad. Con GraphQL tienes un solo endpoint que responde a la información exacta que pide el cliente. (How To GraphQL. (s.f.), s.f.)

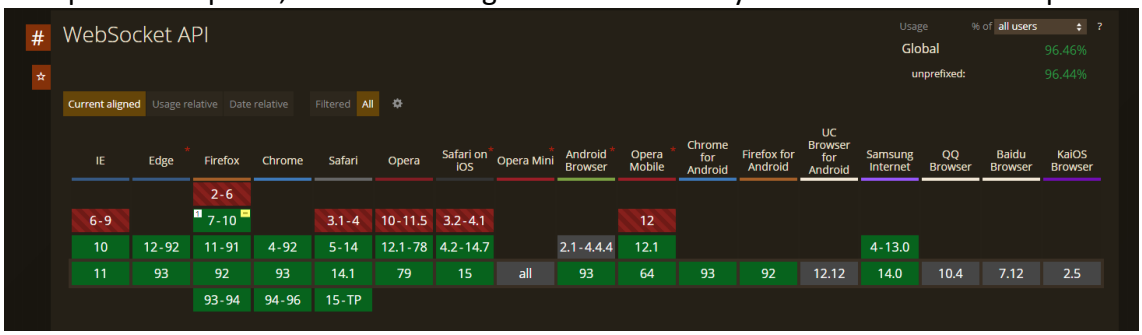
Por otra parte, GraphQL ofrece el concepto de suscripciones para hacer peticiones "servidor-cliente" mandando datos a clientes que escuchan. Este aspecto es clave para implementar las notificaciones de recurso disponible cuando llega tu turno en la cola. Otra ventaja clave es que permite reducir el costo de servidores y acceso a base de datos eliminando por completo el polling con peticiones HTTP. Sólo se refresca la información necesaria para el usuario y el resto del tiempo el único consumo es mantener una conexión WebSockets activa entre cliente servidor. Potencialmente reduce los accesos a base de datos y permite refrescar la información a tiempo real.

Esta decisión permite tener una arquitectura muy escalable y gracias a GraphQL la integración de WebSockets está estandarizada. Es una gran ventaja ya que WebSockets es un protocolo ligero sobre TCP y sería necesario implementar muchas herramientas de soporte si no fuera por GraphQL.

Detallando mejor en qué consiste WebSockets, esta es la definición que ofrece Mozilla:

> **WebSockets** es una tecnología avanzada que hace posible abrir una sesión de comunicación interactiva entre el navegador del usuario y un servidor. Con esta API, puede enviar mensajes a un servidor y recibir respuestas controladas por eventos sin tener que consultar al servidor para una respuesta.
> (WebSockets - Referencia de la API Web | MDN, s.f.)

Y respecto al soporte, todos los navegadores modernos y no tan modernos lo soportan



(WebSocket API | Can I use..., s.f.)

La API GraphQL se implementa en Node.js, Express como servidor HTTPS con Typescript y Passport.js como librería de autenticación OAUTH2.

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Desengancha la necesidad de tener un navegador web para ejecutar Javascript y permite crear código altamente escalable. (Node.js, s.f.)

Express es el estándar de facto a la hora de definir APIs REST, middlewares y servidores HTTP con Node.js y tiene una sencilla integración con GraphQL. (Express.js - Wikipedia, s.f.)

TypeScript es JavaScript con tipado fuerte que compila a JavaScript y fue creado por Microsoft. TypeScript aporta tres grandes ventajas:

- Seguridad de lenguajes con tipado fuerte
- Sintaxis moderna compatible con navegadores que sólo soportan versiones antiguas de JavaScript ya que se encarga de añadir implementaciones de las funciones que falten en el archivo JavaScript resultante
- Una experiencia de desarrollo muy superior gracias a las auto sugerencias eliminando muchos problemas comunes al desarrollar JavaScript.

(TypeScript: JavaScript With Syntax For Types, s.f.)

Passport es una biblioteca de autenticación para Node.js y Express que simplifica la autenticación OAUTH2 mediante sesión o JWT con gran soporte para muchas plataformas de autenticación y control granular de permisos. (Passport.js, s.f.)

Una vez definida la base de datos y API, vamos con la arquitectura y tecnología del cliente.

La interfaz de usuario será una **aplicación web**. En concreto, se decidió que fuera **progresiva** para poder adaptarse tanto a móviles como ordenadores y siendo instalable como aplicación y mostrar notificaciones push al usuario.

¿Cuáles son las diferencias entre aplicación web progresiva (PWA) y una aplicación web de una sola página (SPA)?

Una aplicación web progresiva es una aplicación web en la que se cumplen unas directrices particulares:

- Service Workers: Son scripts instalados en el navegador del cliente que ejecutan las tareas de caché de la página, refresco de caché y notificaciones push
- Conexión HTTPS: Todas las conexiones de la web deben estar seguras usando SSL (HTTPS o WSS en el caso de WebSockets) y con un certificado válido
- Manifiesto de la Aplicación Web: Proporciona la información necesaria para permitir que la web sea instalable como aplicación

La razón principal por la que se eligió que el cliente fuera una aplicación web progresiva fueron las notificaciones push con el service worker y ofrecer una mejor experiencia de usuario. Fue un camino natural ya que la mayoría de los puntos se cumplen con la arquitectura propuesta.

Por otro lado, otra decisión importante fue si se necesitaba renderizado en lado de cliente o en lado de servidor (CSR vs SSR).

A continuación, cito las diferencias y ventajas e inconvenientes de cada una:

La diferencia:

La principal diferencia entre CSR y SSR es dónde se representa la página. SSR representa la página en el lado del servidor y CSR representa la página en el lado del cliente. El lado del cliente administra el enrutamiento dinámicamente sin actualizar la página cada vez que el cliente solicita una ruta diferente.

Usar Renderizado en lado Servidor

- Si el SEO es su prioridad, normalmente cuando está creando un sitio de blog y desea que todos los que buscan en Google vayan a su sitio web, SSR es su elección.
- Si su sitio web necesita una carga inicial más rápida.
- Si el contenido de su sitio web no necesita mucha interacción del usuario.

Usar Renderizado en lado de Cliente

- Cuando el SEO no es tu prioridad
- Si su sitio tiene interacciones enriquecidas
- Si está creando una aplicación web

(Alain2020, 2020)

En mi caso, la decisión está clara, usaré **CSR o renderización de lado de cliente** la interacción con el usuario es bastante dinámica ya que hay que reflejar en todo momento el estado actual del recurso y las acciones cambian respecto a ese estado y

el SEO (optimización para motores de búsqueda) ni siquiera es una consideración para este proyecto.

Una vez decidida la arquitectura, aprovechando los conocimientos previos en Javascript y Typescript adquiridos por propia experiencia laboral, se decidió utilizar **React**, una biblioteca de JavaScript para construir aplicaciones web de una sola página interactivas de forma sencilla mediante una arquitectura de componentes web.

Al igual que GraphQL, **React** fue desarrollado por Facebook y en este caso, es mantenido por Facebook y por la comunidad de software libre. Tiene gran soporte por la comunidad de desarrolladores y se adapta sin fricciones a los requisitos de este proyecto.
(React - Una biblioteca de JavaScript para construir interfaces de usuario, s.f.)

Hasta aquí se define el conjunto de tecnologías básico para la aplicación, pero todavía quedan aquellas decisiones relacionadas con los requisitos de la línea del proyecto como son Docker, Kubernetes y OpenFaaS.

Comenzando por Docker, esta tecnología se basa en el concepto de contenedores de software ofreciendo automatización a la hora de desplegar dichos contenedores.

Los contenedores de software agrupan los elementos y configuraciones necesarios de una aplicación para poder desplegarlos en cualquier sistema operativo. Este concepto puede resultar similar a la virtualización, pero a diferencia de ésta, aquí no se virtualiza hardware, tan solo se porta el software necesario para ser ejecutado, de esta forma el rendimiento es mucho mayor y tiene menor consumo de memoria y almacenamiento.

(Pernil, 2018)

Docker ha sido utilizado en el proyecto como estándar de facto como gestor de contenedores de software, pero el objetivo no es que la aplicación dependa de Docker, el único objetivo es crear imágenes de contenedores que sigan la Open Container Initiative (OCI), un proyecto de la Fundación Linux para diseñar un estándar abierto para los contenedores de software (Open Container Initiative - Wikipedia, la enciclopedia libre., s.f.)

Docker aclara que Docker V2 es la base del estándar OCI

> "El objetivo era proporcionar una especificación confiable y estandarizada, por lo que Docker contribuyó con runc, un entorno de ejecución de contenedor simple, como base del trabajo de especificación de entorno de ejecución, y luego contribuyó con la especificación de imagen de Docker V2 como base para el trabajo de especificación de imagen OCI.
>
> Los desarrolladores de Docker como Michael Crosby y Stephen Day han sido colaboradores clave desde el comienzo de este trabajo, asegurando que la experiencia de Docker alojando y ejecutando miles de millones de imágenes de contenedores se traslade a la OCI. Cuando el grupo de trabajo de certificación complete su trabajo, Docker llevará sus productos a través del proceso de certificación OCI para demostrar la conformidad con OCI."
>
> (Demystifying the Open Container Initiative (OCI) Specifications., 2017)

Con este punto se enlazaría la siguiente tecnología utilizada en el proyecto que determina en enorme medida cómo se orquesta y estructura toda la arquitectura de contenedores de software. Se trata de **Kubernetes**

> Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento. El soporte, las herramientas y los servicios para Kubernetes están ampliamente disponibles.
>
> Google liberó el proyecto Kubernetes en el año 2014. Kubernetes se basa en la experiencia de Google corriendo aplicaciones en producción a gran escala por década y media, junto a las mejores ideas y prácticas de la comunidad.
>
> (Documentación de Kubernetes, s.f.)

Kubernetes está centrado en la administración de contenedores, pero es una herramienta que ofrece control sobre muchos más aspectos como la infraestructura de red y almacenamiento. Posee gran soporte y hay muchos operadores (Extensiones de software a Kubernetes que hacen uso de recursos personalizados

(Documentación de Kubernetes, s.f.)) y permite desplegar software listo para producción y escalable horizontalmente según demanda.

Ofrece una solución completa para montar una arquitectura de clúster y está integrado en muchas plataformas en la nube, así, que una vez definida la arquitectura se puede migrar de plataforma de forma sencilla.

En este proyecto Kubernetes se utiliza como:

- Entorno de ejecución de contenedores
- Red privada entre contenedores
- Gestión de certificados SSL con LetsEncrypt y protección de todos los endpoints públicos de la aplicación con Ingress y CertManager
- Balanceo de carga entre despliegues con MetalLB
- Almacenamiento de base de datos distribuido (MongoDB y Redis)
- Entorno de ejecución de OpenFaaS
- Cronjobs o tareas programadas

La siguiente tecnología clave de este proyecto es OpenFaaS, una herramienta que permite crear y desplegar funciones serverless de forma sencilla en Kubernetes.
(Ellis, s.f.)

Esta herramienta introduce el concepto de FaaS o Function as a Service que ofrece una plataforma en la que desplegar funcionalidades sin tener que considerar la complejidad de construir o mantener la infraestructura asociada con el despliegue de una aplicación.
(Function as a service - Wikipedia, s.f.)

Esto casa perfectamente con la filosofía serverless, donde el proveedor de nube da recursos en demanda, ofreciendo un menor costo para el cliente y para el proveedor de nube.
(Serverless computing - Wikipedia, s.f.)

OpenFaaS y la filosofía serverless tienen inmensas ventajas a nivel de costo y escalabilidad, sólo se paga por el uso que hacen los usuarios, ya que, si no hay demanda, se escala a 0 instancias, y según aumente la demanda, se replica dicha función hasta cubrir demanda.
(Ellis, s.f.)

Se plantea como una tecnología que se adapta de forma muy adecuada a esta aplicación, ya que es el usuario quien dispara todas las interacciones y en un uso normal de la aplicación, el usuario rara vez va a estar largos periodos de tiempo seguidos utilizando la aplicación. Esta arquitectura y filosofía que se obtiene con OpenFaaS permite escalar la aplicación a producción con costes mínimos.

Como último punto para finalizar esta sección, durante el desarrollo del proyecto, surgió la necesidad de añadir un broker de mensajería para gestionar las suscripciones de GraphQL por WebSockets e implementar un mecanismo de Publicador-Suscriptor escalable para la arquitectura propuesta en Kubernetes.

> Redis es un almacenamiento de estructuras de datos en memoria usado como base de datos, caché y broker de mensajería. (Redis, s.f.)

Esta integración se realiza con una implementación específica para Redis de las suscripciones de GraphQL en Node.js

Adicionalmente Redis se utiliza en los microservicios de GraphQL como una capa de caché para reducir las llamadas a base de datos cuando los datos no varían, reducir costos y mejorar la experiencia de usuario con respuestas prácticamente inmediatas.

3

Metodología de trabajo

La metodología aplicada en el desarrollo del trabajo es la experimentación ya que la idea es una propuesta personal. Se aplica de forma
práctica la arquitectura planteada, investigando sobre la teoría y uso de las tecnologías implicadas.

3.1 Objetivos

- Implementar la aplicación propuesta y analizar cómo encaja en la arquitectura propuesta, qué problemas resuelve y qué inconvenientes presenta, tanto funcionalmente como técnicamente
- Evaluar la arquitectura de microservicios con GraphQL
- Evaluar la capacidad de orquestación de Kubernetes para alojar una aplicación en su totalidad
- Evaluar el valor de una arquitectura Serverless con OpenFaaS, qué soluciones aporta y qué inconvenientes presenta
- Evaluar la escalabilidad de la aplicación propuesta
- Evaluar el uso del protocolo WebSockets para comunicación bidireccional
- Evaluar la aplicación de WebPush para mandar notificaciones en una aplicación web progresiva
- Evaluar ventajas e inconvenientes de una aplicación web progresiva

3.2 Metodología

Apodada como “Allotr”, la aplicación propuesta, en un inicio sólo tenía la pretensión de ser un proyecto relativamente sencillo para poner a prueba una arquitectura compleja con tecnologías como GraphQL, WebPush, WebSockets, microservicios GraphQL y funciones como servicio (FaaS).

A lo largo del desarrollo, “Allotr”, ha presentado más retos y oportunidades de los previstos y ha incitado a investigar en mucho más detalle.

3.2.1 GraphQL y WebSockets

Como antesala al anteproyecto, se realizó una prueba de concepto con GraphQL y WebSockets para evaluar que la arquitectura funcionaría correctamente. En local todas las pruebas resultaron exitosas.

3.2.2 Kubernetes

La investigación del proyecto comienza con Kubernetes, una plataforma compleja que realiza muchas tareas de orquestación más allá de gestionar contenedores.

Kubernetes permite desplegar software en contenedores, escalar automáticamente, limitar los recursos que utiliza cada recurso, crear y gestionar redes, hacer balanceo de carga, ejecutar tareas programadas, generar automatizaciones complejas a nivel de infraestructura… El poder de Kubernetes es inmenso, pero para este proyecto, estos son los componentes con los que más he interactuado:

- Deployment -> Permiten definir planes de despliegue de aplicaciones en contenedores
- Service -> Define la exposición de una aplicación (Deployment/Pod/ReplicaSet) a la red interna del clúster, abriendo un puerto a la máquina o generando una IP con un balanceador de carga
- Pod -> Son las instancias donde se ejecutan esas imágenes de los contenedores. Se pueden definir de forma independiente sin un Deployment, con un ReplicaSet
- ReplicaSet -> Es un recurso que asegura que se ejecuten una cantidad determinada de pods iguales para escalar. Está incluido dentro de los Deployment
- Ingress -> Este recurso permite crear redes con balanceo de carga a partir de dominios, rutas e incluso permite añadir certificación SSL en los dominios si se combina con CertManager

- CronJob -> Permite ejecutar tareas de forma programada con una programación temporal (e.g. cada 20 minutos -> */20 * * * *)
- Operador -> Herramienta que orquesta la creación/gestión de recursos en el clúster Kubernetes
- Secret -> Recurso para guardar datos de configuración de forma confidencial

(Documentación de Kubernetes, s.f.)

Una vez comprendidos los conceptos base, es necesario decidir qué implementación instalar. Como servidor de desarrollo se utilizará una Raspberry Pi 4 de 8GB de RAM y en base a las limitaciones de rendimiento se decidió usar k3s, que es más ligero que k8s (Proyecto original de Kubernetes por Google) y no presenta incompatibilidades con los recursos de k8s.
(Slingerland, 2022)

Adicionalmente es necesario instalar un balanceador de carga en Kubernetes como MetalLB que permita generar IPs dentro del rango de IPs privadas del router para poder exponer en una sola IP todo el tráfico de aquellos servicios que queramos exponer.

Para exponer los servicios se usará Ingress, cuya configuración resulta sencilla y junto al balanceador de carga proporciona una IP única que se podrá exponer a internet abriendo un solo puerto, el 443.

A nivel de Kubernetes, no he encontrado ninguna limitación, permite implementar al completo la aplicación propuesta.

3.2.3 OpenFaaS

Se continúa la investigación con OpenFaaS, que presenta un concepto muy útil y atractivo: desplegar código sin pensar en el servidor, funciona y escala automáticamente.

Pese a que la premisa es realmente sencilla, en la práctica, no se adapta por defecto a la aplicación propuesta. OpenFaaS funciona con plantillas adecuadas a distintos lenguajes de programación para que sea más sencillo el desarrollo y despliegue.
(OpenFaaS Classic templates, s.f.)

El problema surge cuando no existe una plantilla que se adapte a las tecnologías de la aplicación propuesta, como TypeScript, Express y uWebSockets.js.

Es por esta razón que se implementaron plantillas personalizadas para solventar el problema:

Véase:

- https://github.com/rafaelpernil2/openfaas-template-node-typescript-express
- https://github.com/rafaelpernil2/openfaas-template-node-typescript-uwebsockets

Una vez resuelto este problema surgió otro más con su falta de compatibilidad con WebSockets.
(Websockets and HTTP mode, s.f.)

OpenFaaS no es compatible con WebSockets porque las conexiones con WebSockets son de larga duración y conservan estado. En el paradigma serverless, las conexiones deben ser de corta duración y sin estado para permitir completa escalabilidad.

En este punto decidí excluir de OpenFaaS las suscripciones de GraphQL e incluirlas en un nuevo servicio web expuesto directamente desde Kubernetes. Esto resolvió el problema.

Como último punto reseñable en la investigación, para que OpenFaaS exponga las funciones en un endpoint con certificado SSL, es necesario aplicar una configuración especial con el operador Ingress para OpenFaaS y con el operador cert-manager
(IngressOperator for OpenFaaS, s.f.)
(cert-manager, s.f.)
(TLS on Kubernetes, s.f.)

3.2.4 cert-manager

Enlazando con el punto anterior, fue necesario investigar sobre cert-manager para la generación de certificados SSL en el dominio para la aplicación, https://allotr.eu

Esta herramienta es un operador de Kubernetes que solicita certificados “Let’s Encrypt” al proveedor de DNS asociado al dominio y en caso de autorizarlo, estos certificados se guardan y auto renuevan en el clúster.
(Certificados SSL/TLS Gratuitos - Let's Encrypt, s.f.)

La configuración inicial resulta algo complicada si el proveedor de DNS no está en la lista de los proveedores estándar. Con Cloudflare tan solo es necesario proveer un token y automáticamente cert-manager se encarga de verificar que eres el dueño del dominio y generar los certificados.
(Cloudflare - cert-manager Documentation, s.f.)

Estos certificados son gratuitos y totalmente válidos en cualquier navegador, sin tener que instalar nada más.

3.2.5 Gestión de dominio y DNS

Para poder solicitar los certificados, fue necesario comprar un dominio y migrar el proveedor de DNS a Cloudflare.

Se crea un registro A que enlaza con la IP de externa del router, se expone por el puerto 443 la IP generada por el controlador Ingress de Kubernetes, y cualquier recurso Ingress generado con esos dominios tendrá certificado y se enrutará correctamente a dentro del clúster.

3.2.6 MongoDB en cloud/en Kubernetes

En este punto ya se encuentran resueltos los mayores puntos críticos de la configuración. Ahora hay que analizar si es posible y factible instalar la base de datos MongoDB en el propio clúster Kubernetes.

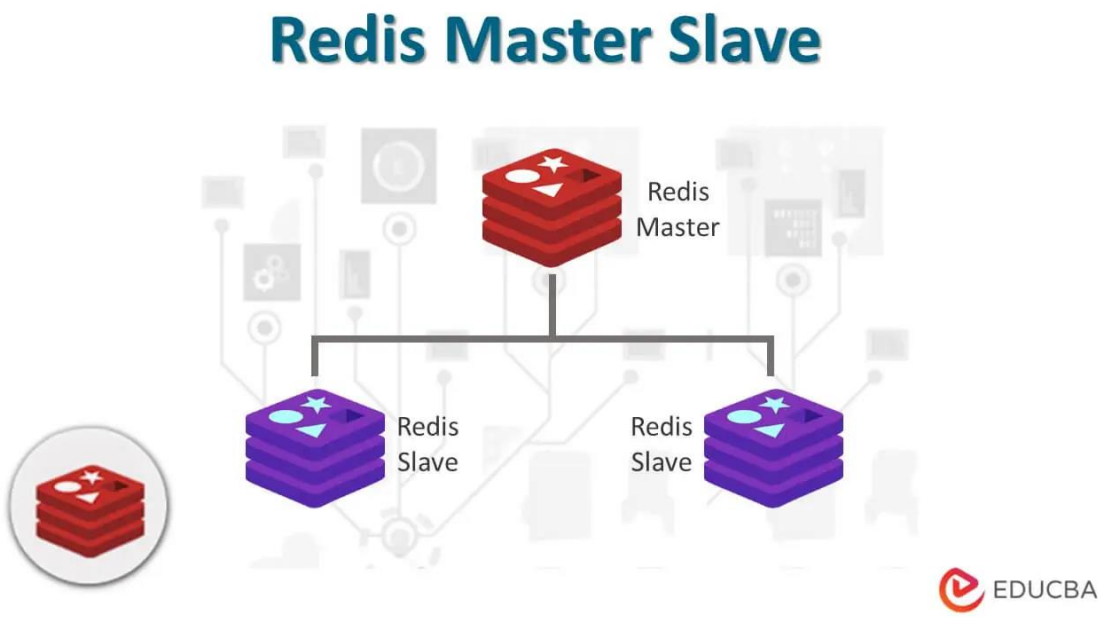
Si se tratara de un clúster con arquitectura x86_64, sin mayor dificultad se podría usar el operador de Kubernetes “MongoDB Community Kubernetes Operator” (MongoDB Community Kubernetes Operator, s.f.)

Este operador permite crear una base de datos con un número determinado de réplicas para mejorar la fiabilidad y rendimiento.

Pero en una Raspberry Pi, con arquitectura ARM, este operador no se puede instalar. Por esa razón se opta por crear una base de datos en la nube con MongoDB Atlas (MongoDB Atlas Database | Multi-Cloud Database Service, s.f.)

3.2.7 Redis

Para la base de datos Redis, se ha optado por desplegar una configuración maestro-esclavo mediante “bitnami-redis”, un conjunto de recursos de Kubernetes que orquestan y escalan automáticamente una base de datos Redis.
(Helm Charts to deploy Redis® in Kubernetes, s.f.)



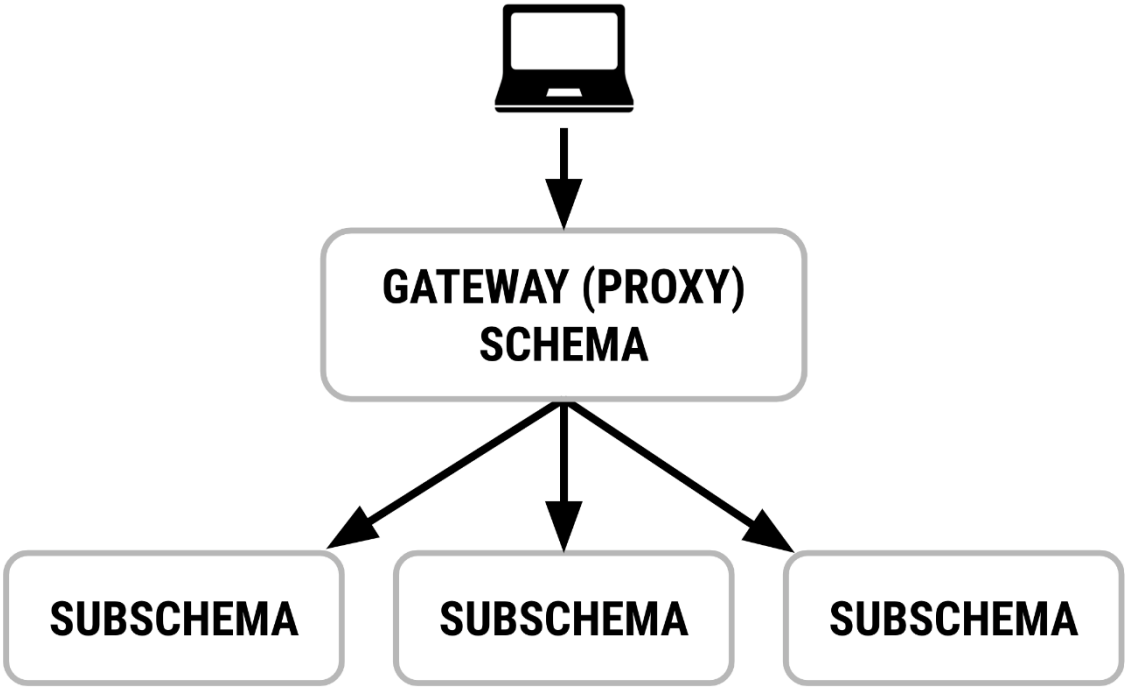
(How to Setup Redis Master and Slave Replication? - EDUCBA)

La arquitectura maestro-esclavo mantiene los datos más seguros al repartirse entre las distintas réplicas y también se agiliza la lectura gracias al reparto de carga.

3.2.8 GraphQL, microservicios y caché

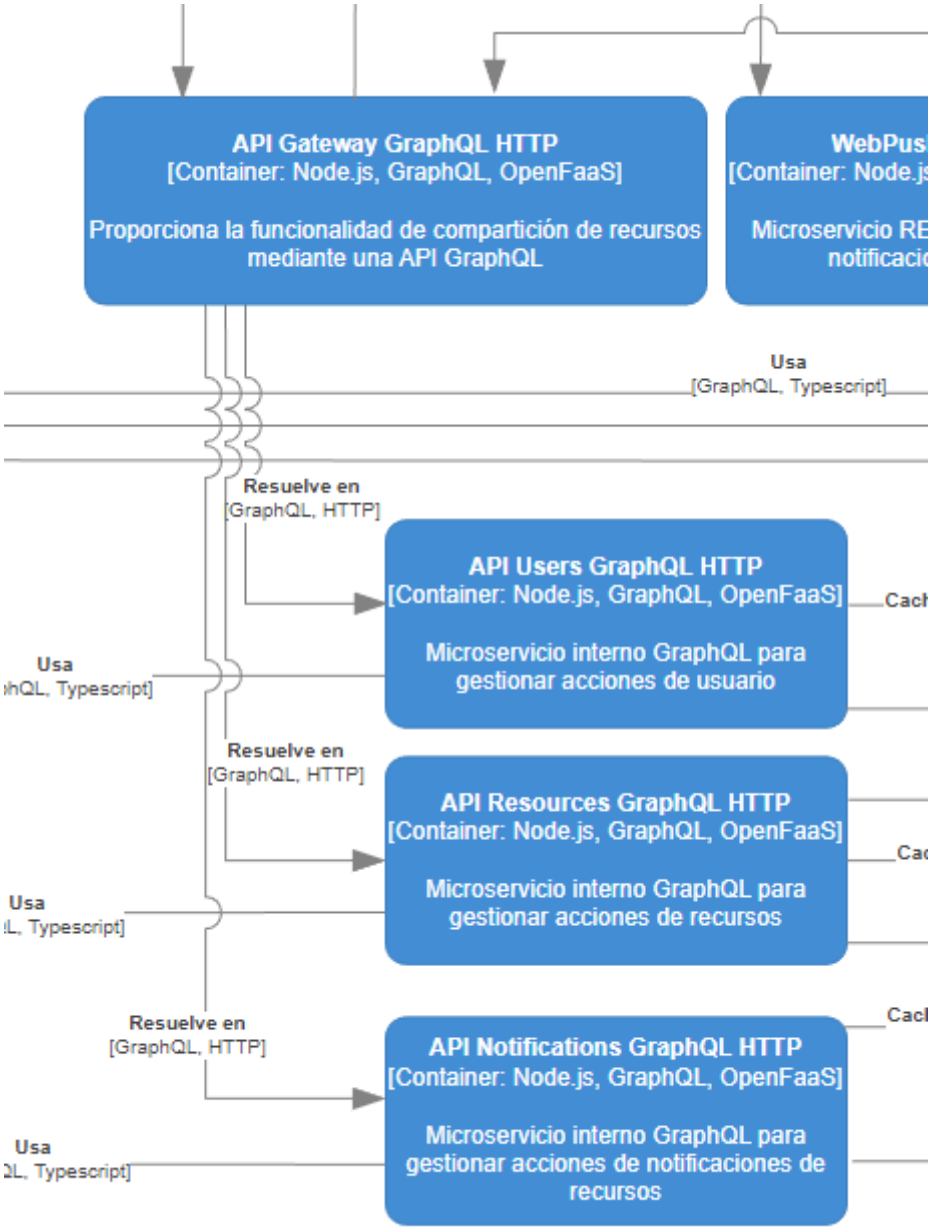
Para hacer que GraphQL sea escalable, es necesario separar el código que resuelve las peticiones con acceso a base de datos y lógica de negocio de el código que realiza el procesamiento de la petición GraphQL y enruta.

Esto se logra mediante la arquitectura de microservicios. Debe existir un servicio de tipo API Gateway que se encarga de enrutar las peticiones GraphQL según a qué esquema hagan referencia. Esto se logra mediante GraphQL Stitching, que resultó relativamente sencillo de configurar.
(Home – GraphQL Stitching, s.f.)



(Introduction – GraphQL Stitching, s.f.)

Los microservicios GraphQL reciben las peticiones del API Gateway GraphQL y no deberían estar expuestos a internet, ya que el punto de entrada debe ser el API Gateway siempre.



(Diagrama de arquitectura Allotr, 2023)

Adicionalmente, se ha investigado cómo añadir una caché en las respuestas para agilizar las llamadas cuando los datos no cambian y así reducir las llamadas a base de datos. Esto se realiza con GraphQL Yoga con distintas librerías y una conexión con la base de datos Redis.

Presentó complejidad encontrar una forma efectiva de invalidar la caché entre procesos para que siempre se visualice la información actualizada. Para que sea escalable, se genera una clave de “bloqueo” en la base de datos Redis y se lee siempre que la librería intente introducir en caché la respuesta. Si está bloqueado en este momento invalidando datos previos, no permite que se introduzca en caché.

La generación de la mencionada clave de bloqueo requiere un proceso asíncrono y la librería utilizada sólo proporciona una función síncrona para evaluar si se debe introducir en caché la respuesta. Por esa razón, ha sido necesario modificar la librería y usarla como reemplazo.

Otro problema adicional surge con la autenticación. Es necesario crear un servicio de autenticación y una forma de obtener la sesión del usuario en los microservicios.

3.2.9 Autenticación OAUTH2

Para la autenticación OAUTH2 fue necesario investigar qué librerías ofrecen integraciones sencillas, en este caso con la autenticación de Google.

Se empleó la librería Passport.js en conjunto con Node.js y Express.

Esta librería se encarga de generar los usuarios en la base de datos en el caso de registro y verifica los usuarios en caso de autenticación.

Para la autenticación, el servicio inyecta una cookie en el navegador para que se reciban en todas las peticiones que se realizan desde la interfaz web.

Luego, el resto de los microservicios deben desencriptar la cookie y encontrar la sesión a la que hacen referencia en la base de datos.

Todas las sesiones se guardan en la base de datos MongoDB, al igual que los datos de la aplicación

3.2.10 WebPush

Las notificaciones WebPush permiten que una aplicación web progresiva mande notificaciones al dispositivo del usuario.
(Aplicaciones web progresivas - web.dev, s.f.)
(Push API, s.f.)

La aplicación propuesta necesita las notificaciones para avisar al usuario de que ya puede usar el recurso que estaba esperando.

Esto funciona solo en aplicaciones web progresivas porque la web necesita estar instalada en el dispositivo y tener un service worker ejecutándose de fondo para escuchar las notificaciones.
(Service Worker API, s.f.)

Cuando se registra un usuario mediante WebPush, el servicio recibe un endpoint que se debe guardar en base de datos para invocarlo cuando se quiera enviar una notificación.
(Push API, s.f.)

3.2.11 Aplicación Web Progresiva

Para que una aplicación web se considere progresiva debe cumplir varios criterios entre ellos incluir una política de caché para disponer de modo offline.

- Debe tener un manifiesto con los datos necesarios (como un icono de aplicación)
- Debe estar bajo HTTPS
- Debe tener un icono para representar la aplicación
- Debe tener un service worker

(How to make PWAs installable, s.f.)

Fue necesario investigar los requisitos para adaptar la web y que fuera instalable.

3.2.12 React

Para el desarrollo de una aplicación web con React, es necesario incluir otras dependencias que ayuden con el desarrollo en local. Como servidor local se usó Vite, que permite recarga en caliente de la página mientras se programa, incluyendo TypeScript.

A la hora de añadir el service worker fue necesario investigar qué librería añadía soporte al proceso de minificación y empaquetamiento de la web para su despliegue en producción.

Por otra parte, se implementó la aplicación web siguiendo la documentación oficial sin mayores bloqueos.

3.2.13 Express

Este es un framework de facto para crear servidores HTTP con Node.js. La experiencia e integración con éste fue sin mayor problema, siguiendo la documentación oficial de las distintas librerías todo funcionó perfectamente.

3.2.14 GraphQL y uWebSockets.js

Evaluando los tiempos de respuesta de GraphQL cuando se aplica stitching para implementar la arquitectura de microservicios, la respuesta tarda 1.5 segundos por petición cuando debería tardar unos 200-300ms como mucho. Para agilizar y reducir cuellos de botella, en los servicios de GraphQL se utiliza como servidor HTTP la librería uWebSockets.js, que es un servidor HTTP implementado en C++ optimizado y enlazado como una librería de Node.js

3.2.15 Docker y compilación cruzada

La instalación de Docker para compilar los proyectos y generar los contenedores es bastante sencilla siguiendo la guía oficial
(Install Docker Engine on Ubuntu - Docker Docs, s.f.)

Para realizar compilación multiplataforma y poder generar imágenes para ARM y x86 es necesario instalar Docker buildx y descargar todos los intérpretes siguiendo la guía oficial
(Multi-platform images - Docker Docs, s.f.)

4

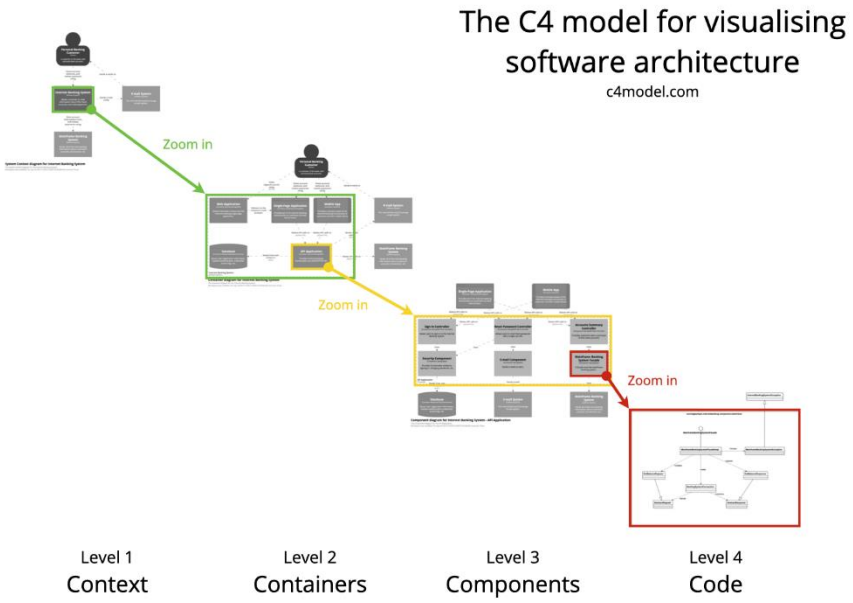
Diseño

El diseño de la aplicación se ha realizado a distintos niveles:

- Diagrama de arquitectura
- Wireframes
- Casos de uso
- Casos de prueba
- Modelo de datos
- Esquema GraphQL
- Máquina de estados de boletos del recurso

4.1 Diagrama de arquitectura

Para este diagrama he empleado el modelo C4



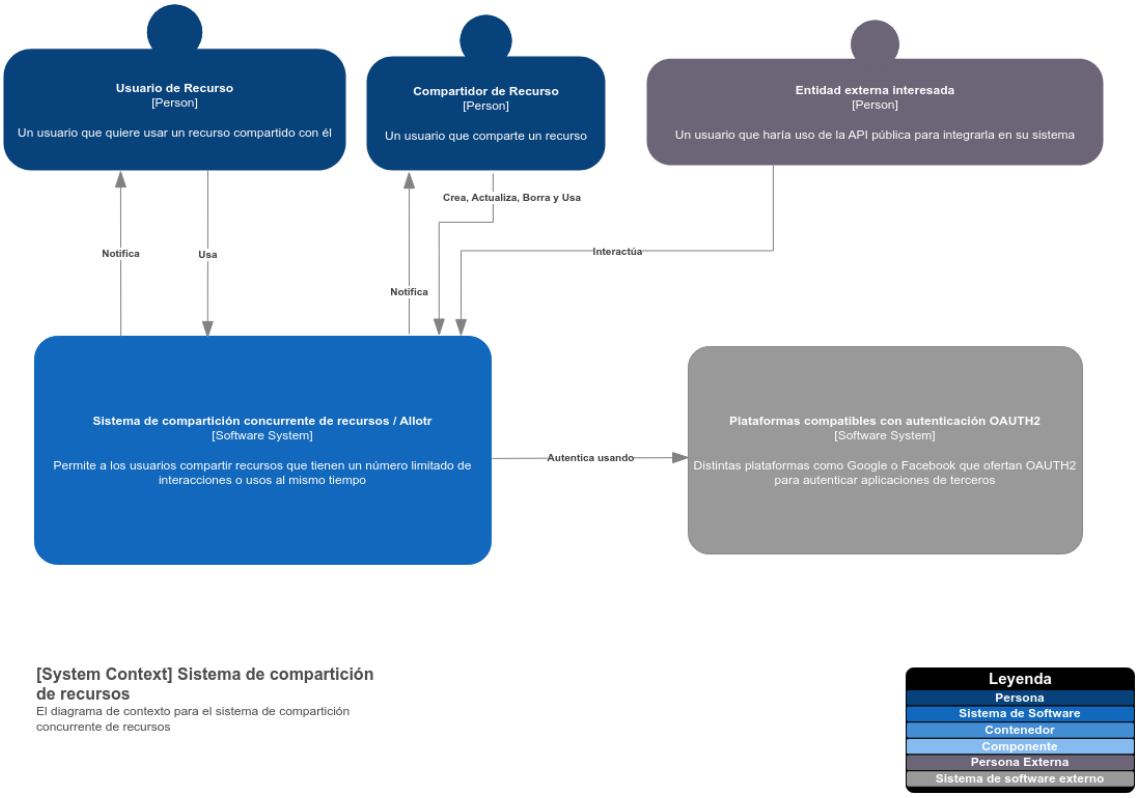
(Brown, 2011)
Describe la arquitectura a distintos niveles de abstracción: Sistema, Contenedores y Componentes

No se incluye el nivel de código ya que se emplea programación funcional y no es necesario incluir UML.

También se puede consultar en https://graphql-docs.allotr.eu/ArquitecturaTFG.drawio.html

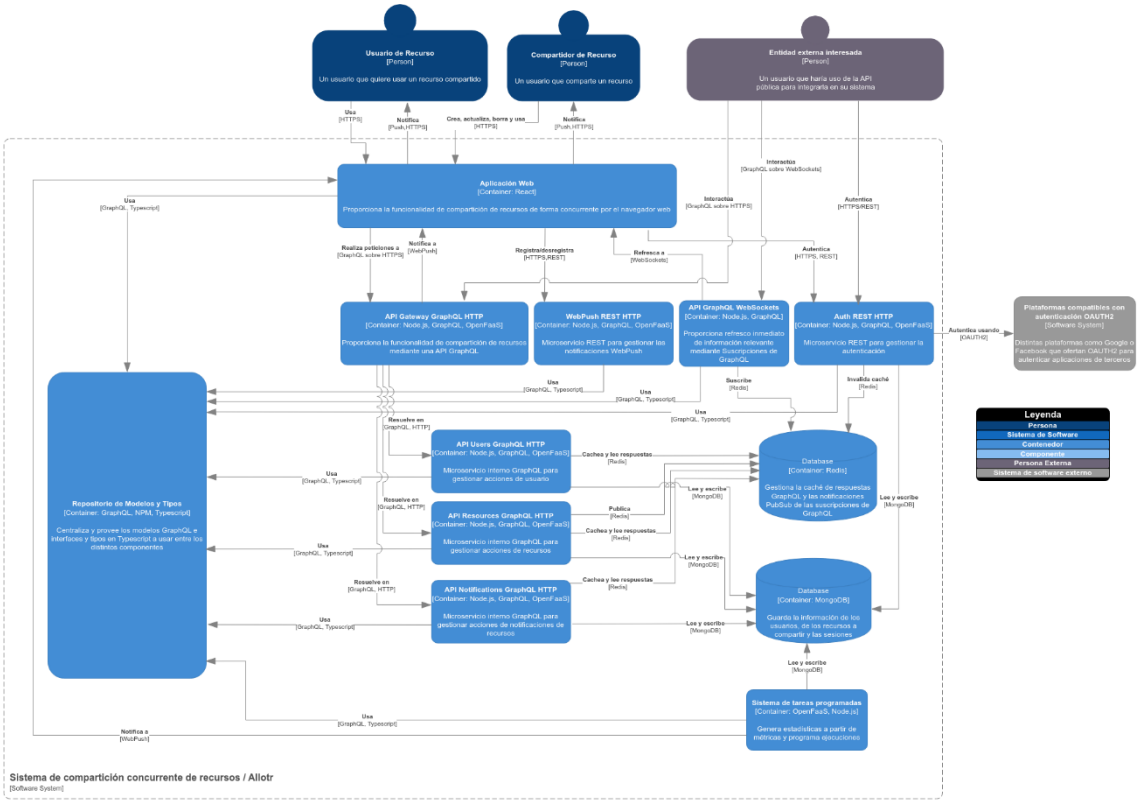
Y se adjunta a la memoria como ArquitecturaTFG.drawio

4.1.1 Sistema



Aquí se visualizan a alto nivel las interacciones del sistema y sus conexiones

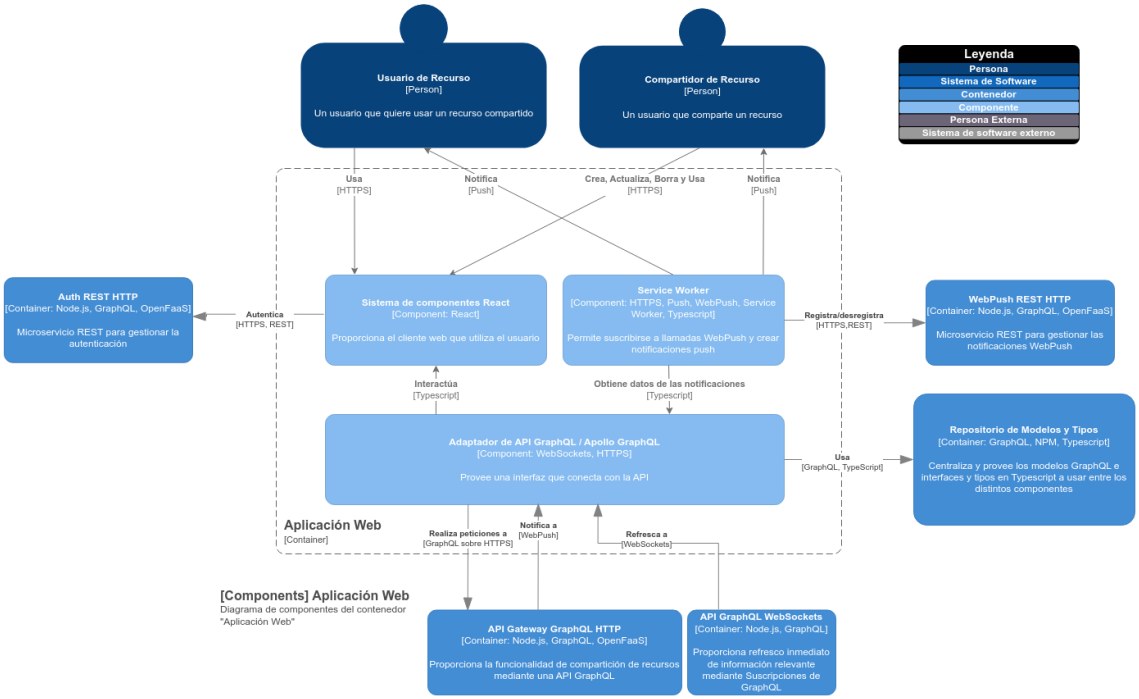
4.1.2 Contenedores



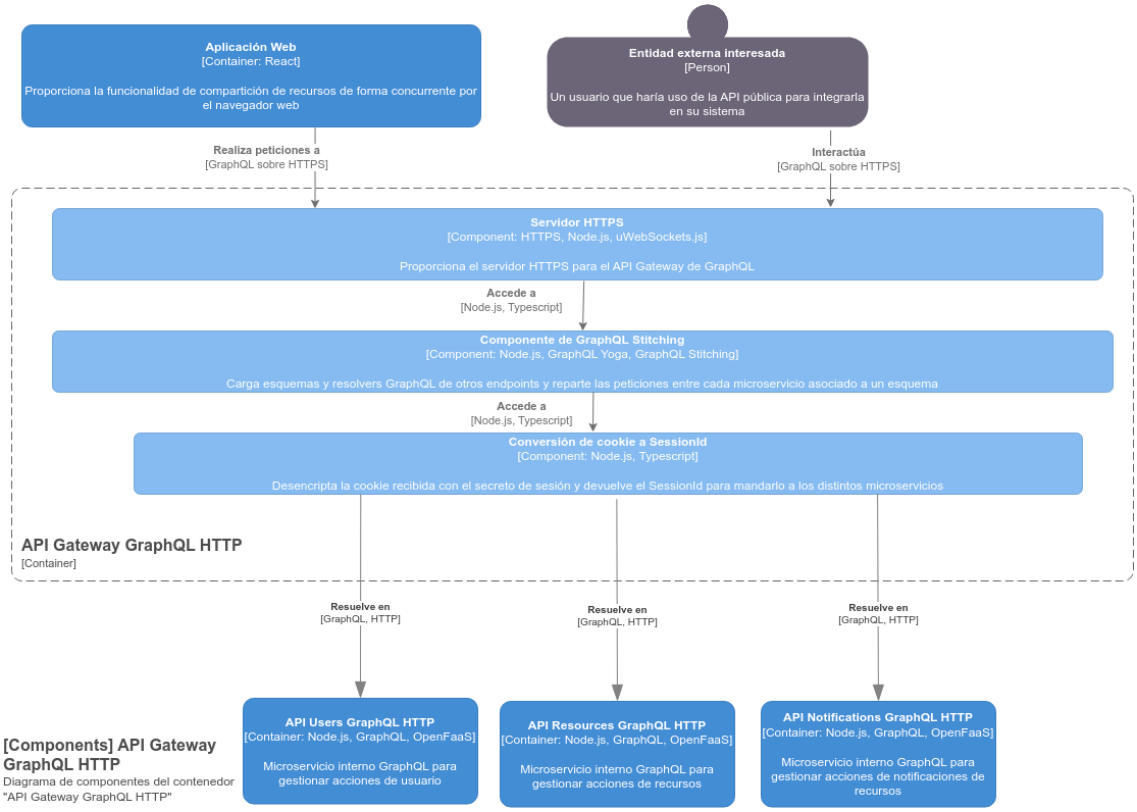
[Containers] Sistema de compartición concurrente de recursos / Allotr
Diagrama de contenedores del sistema de compartición concurrente de recursos

Aquí se puede ver en más detalle todos los contenedores que componen la aplicación. Todos ellos se desplegarían en Kubernetes

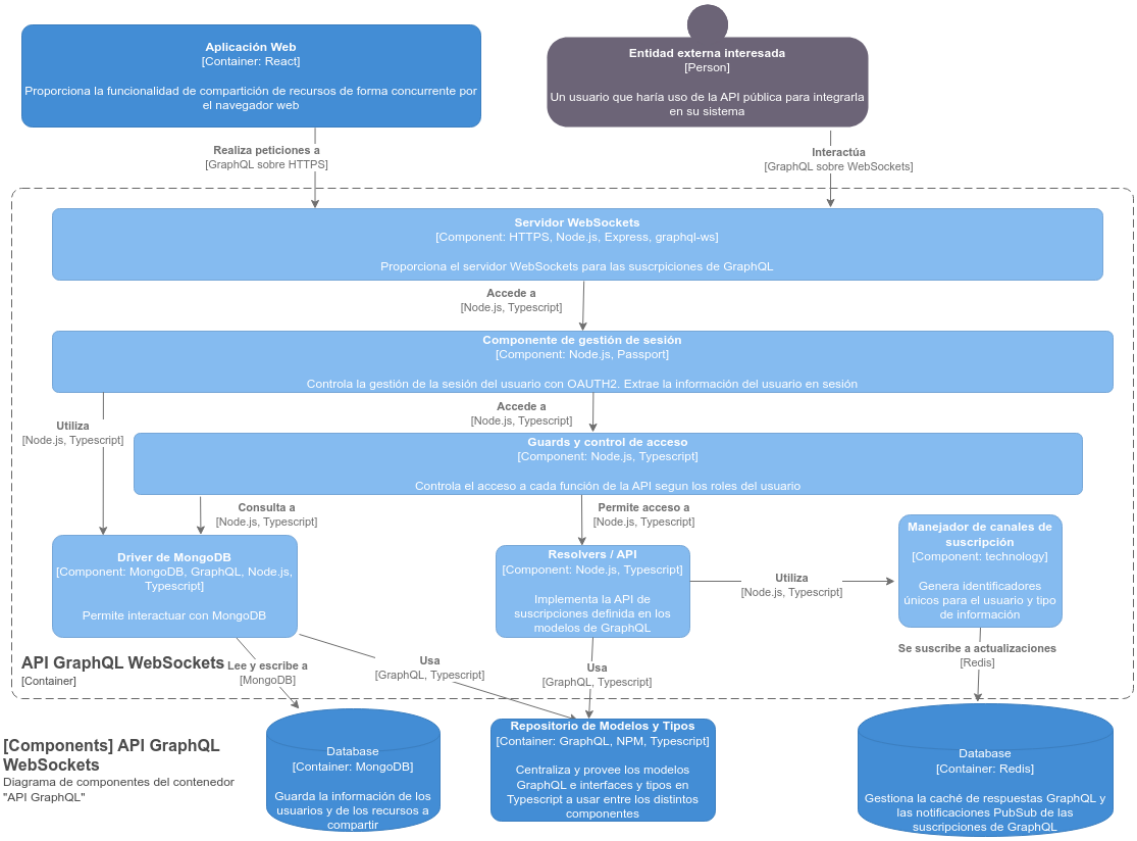
4.1.3 Componentes



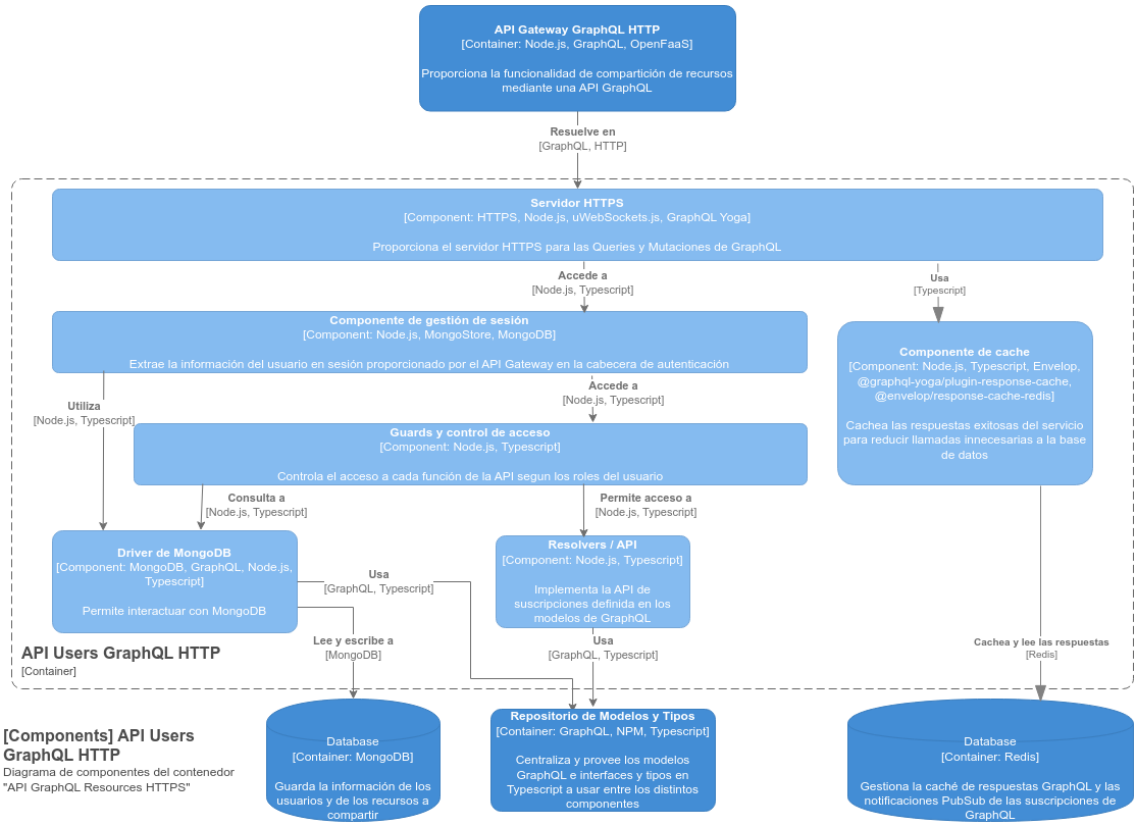
Aplicación Web



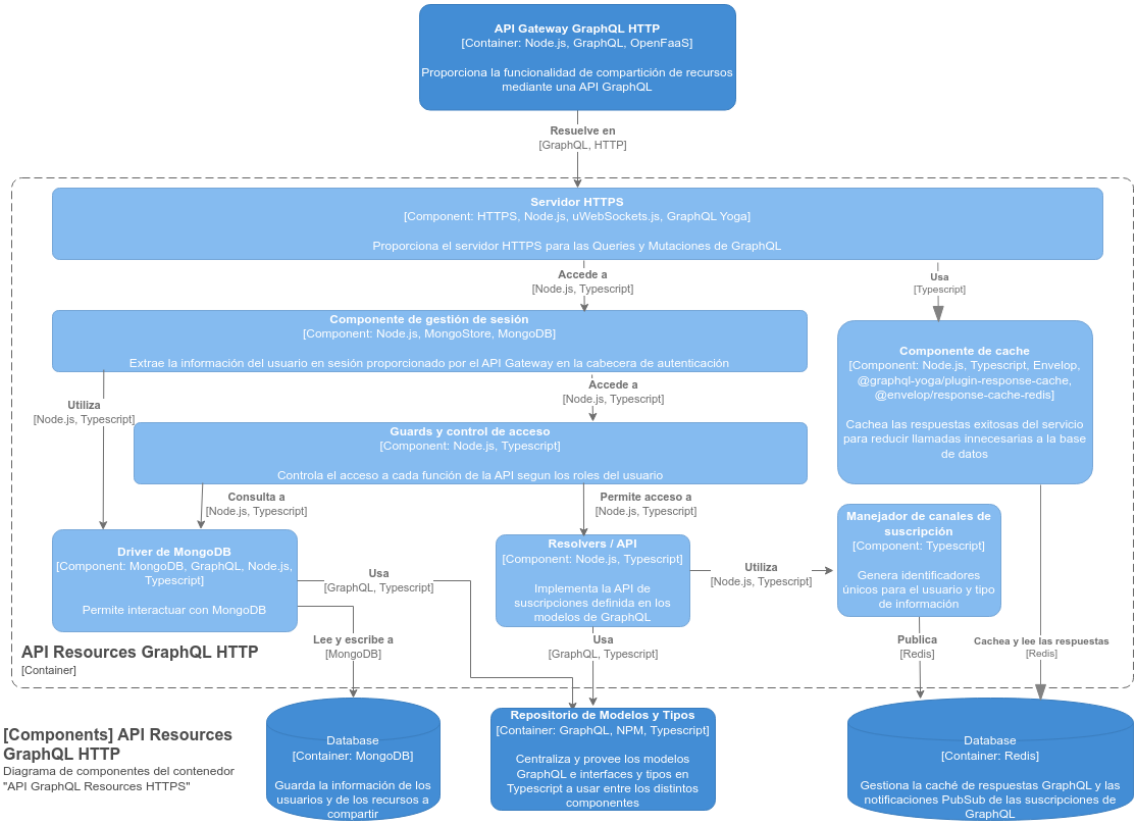
API Gateway GraphQL HTTP



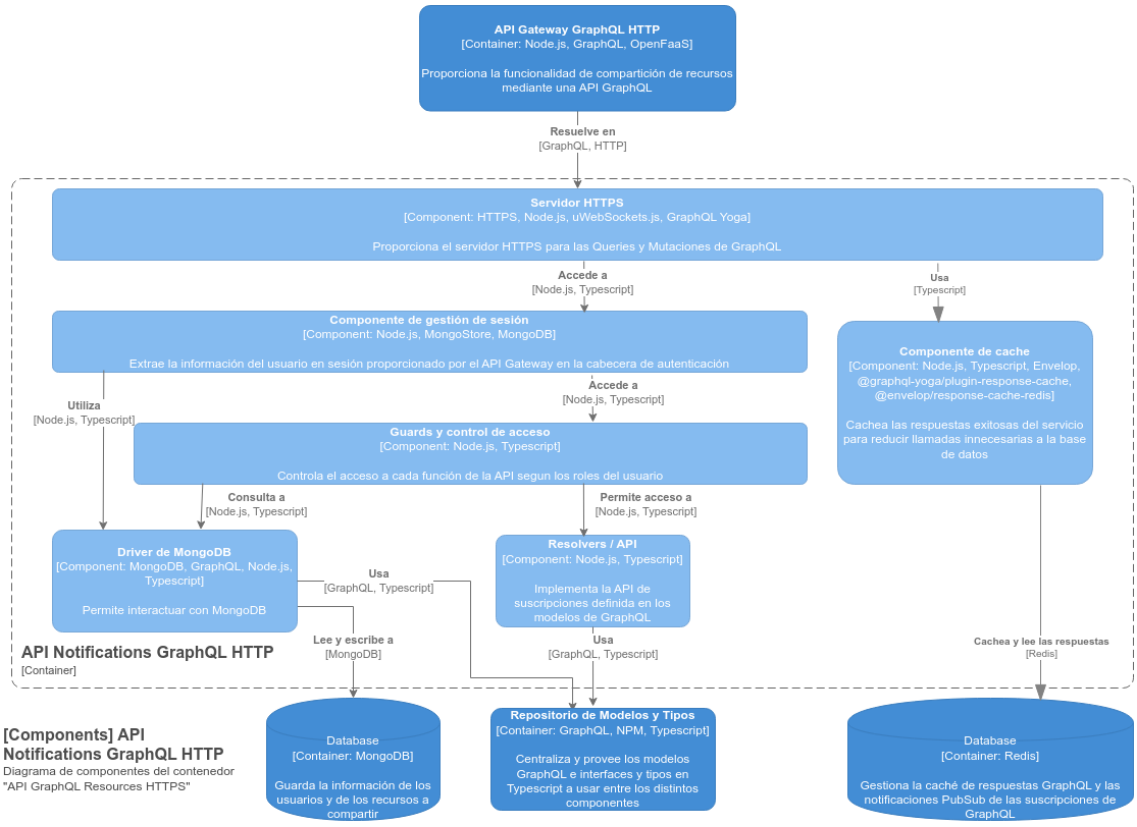
API Gateway GraphQL WebSockets



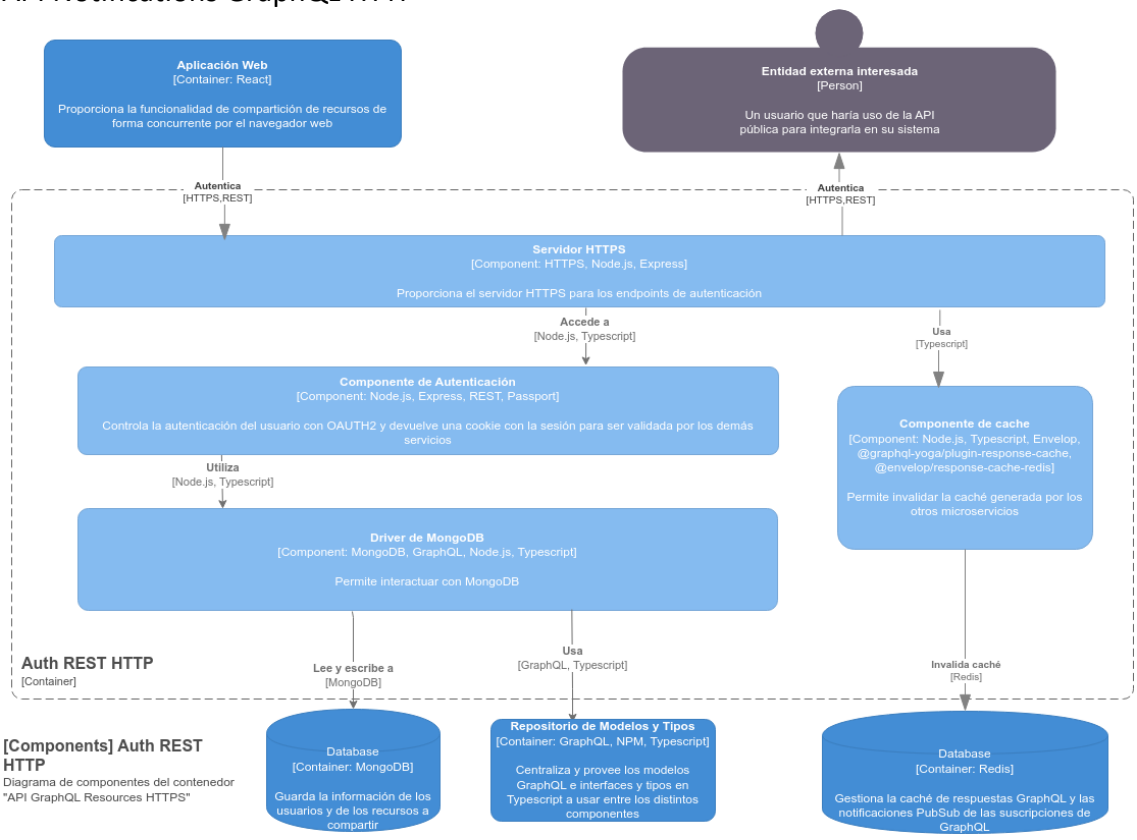
API Users GraphQL HTTP



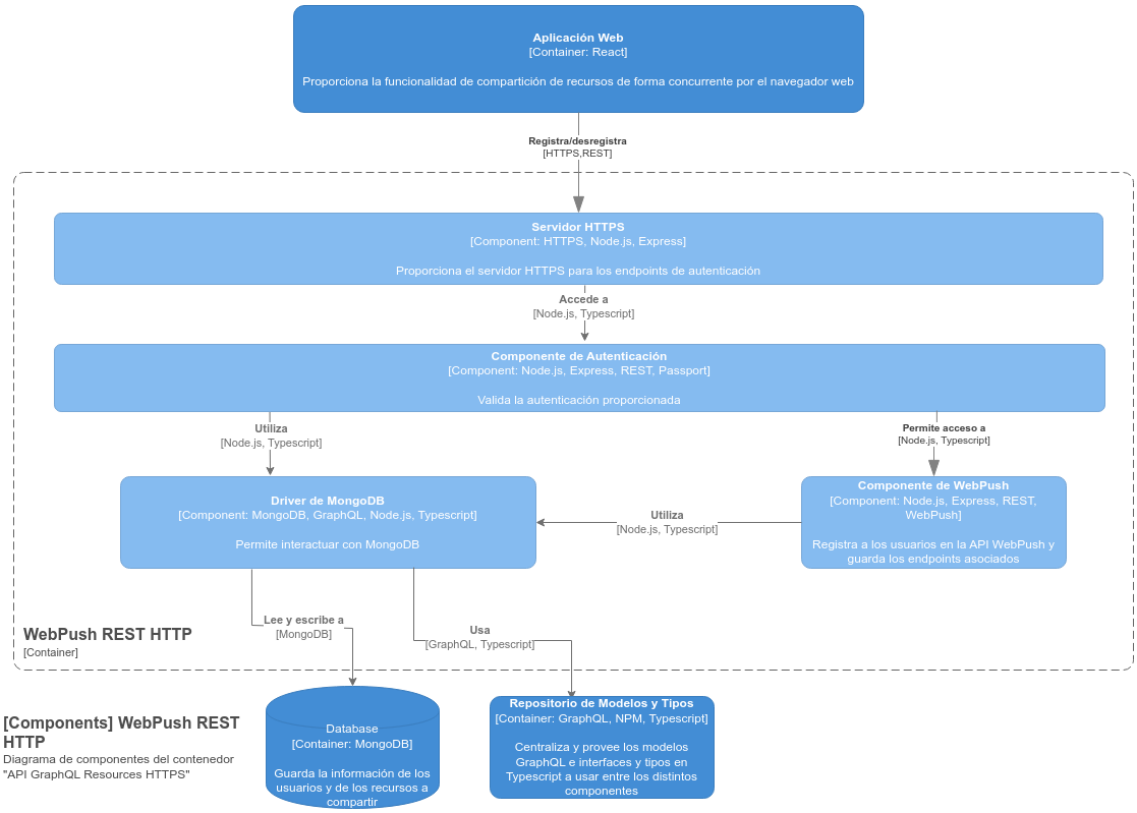
API Resources GraphQL HTTP



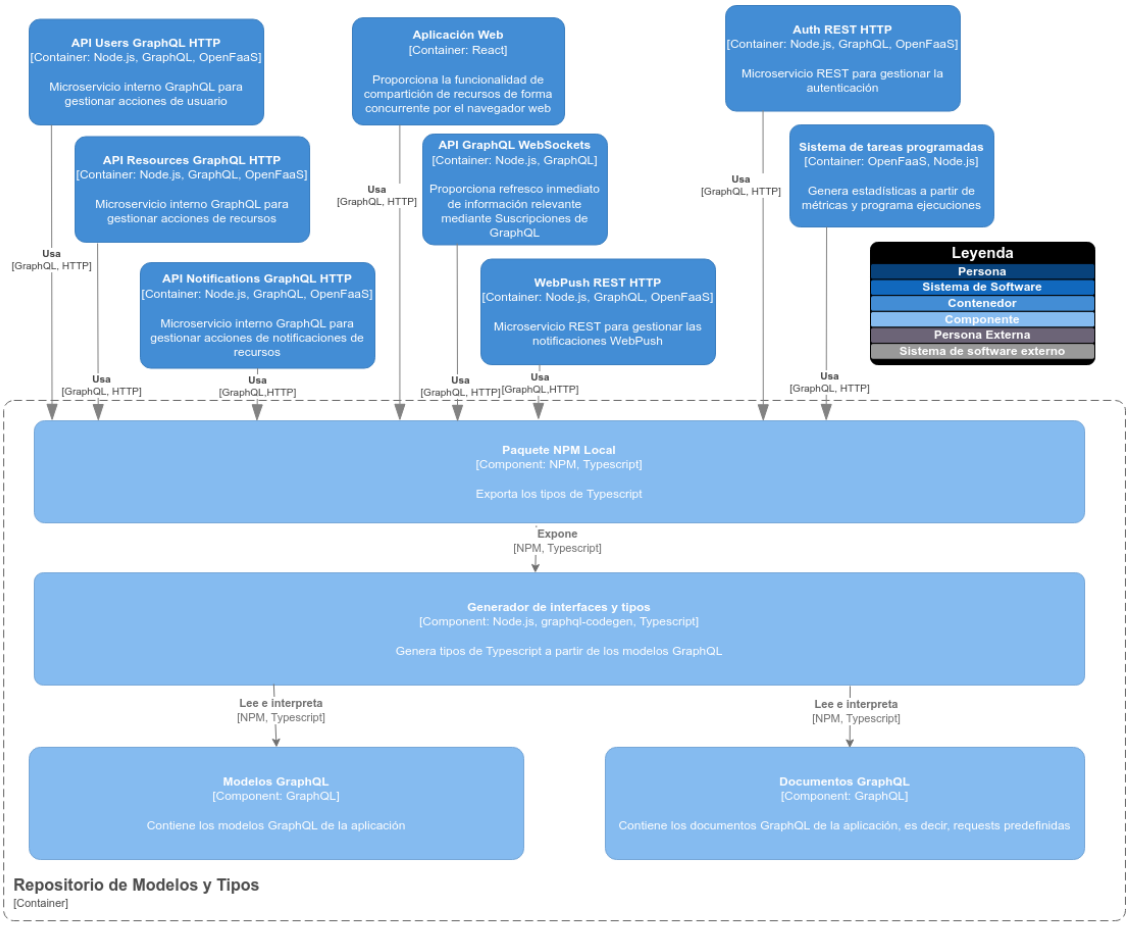
API Notifications GraphQL HTTP



Auth REST HTTP

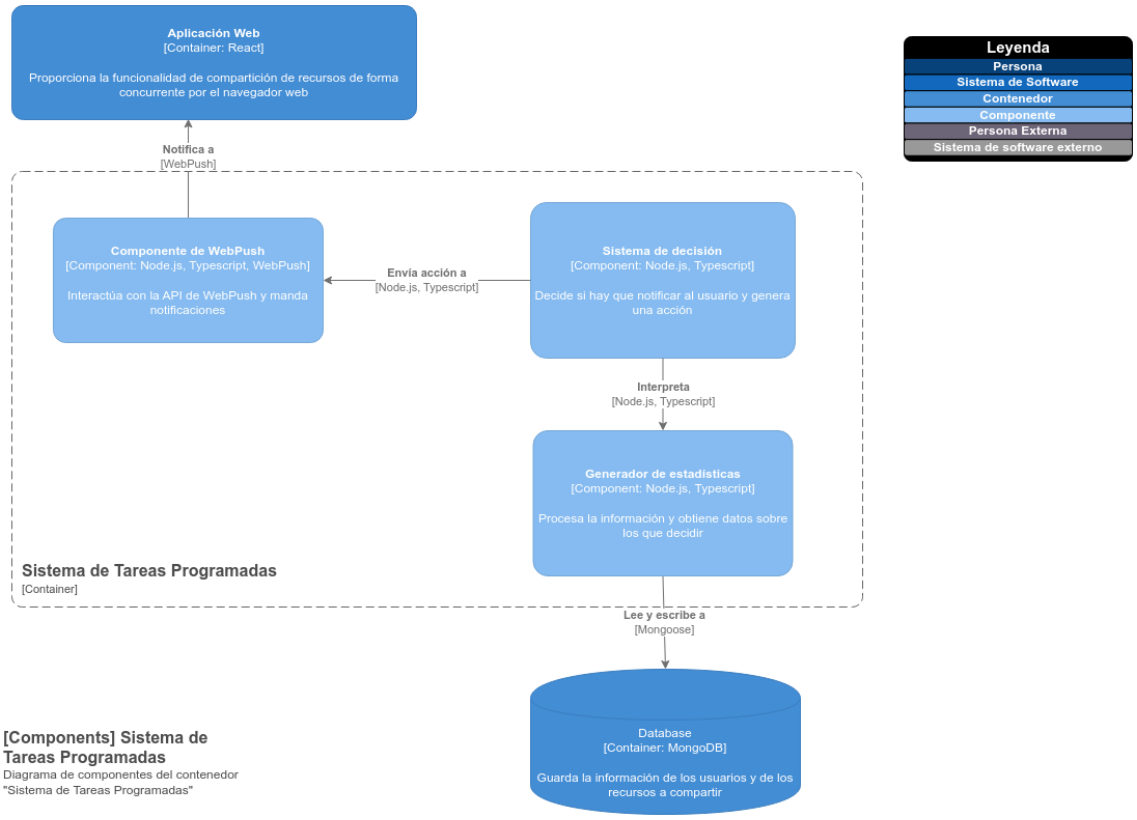


WebPush REST HTTP



[Components] Repositorio de Modelos y Tipos
Diagrama de componentes del contenedor "Repositorio de modelos y tipos"

Repositorio de Modelos y Tipos



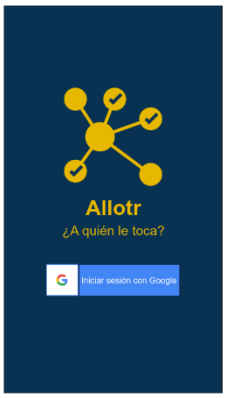
Sistema de Tareas Programadas

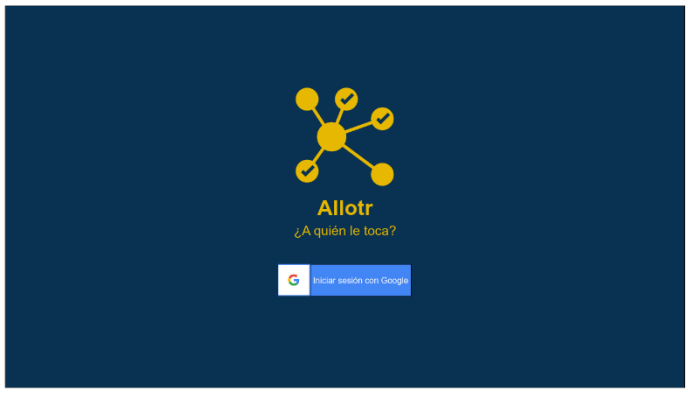
4.2 Wireframes

Los wireframes sirven como una referencia del diseño de la aplicación web.

La paleta de colores ha sido escogida mediante Adobe Colors y se ha ajustado para que cumpla con el criterio de accesibilidad WCAG 2.1 con nivel de conformidad AA
(Adobe Color, s.f.)
(Web Content Accessibility Guidelines (WCAG) 2, s.f.)

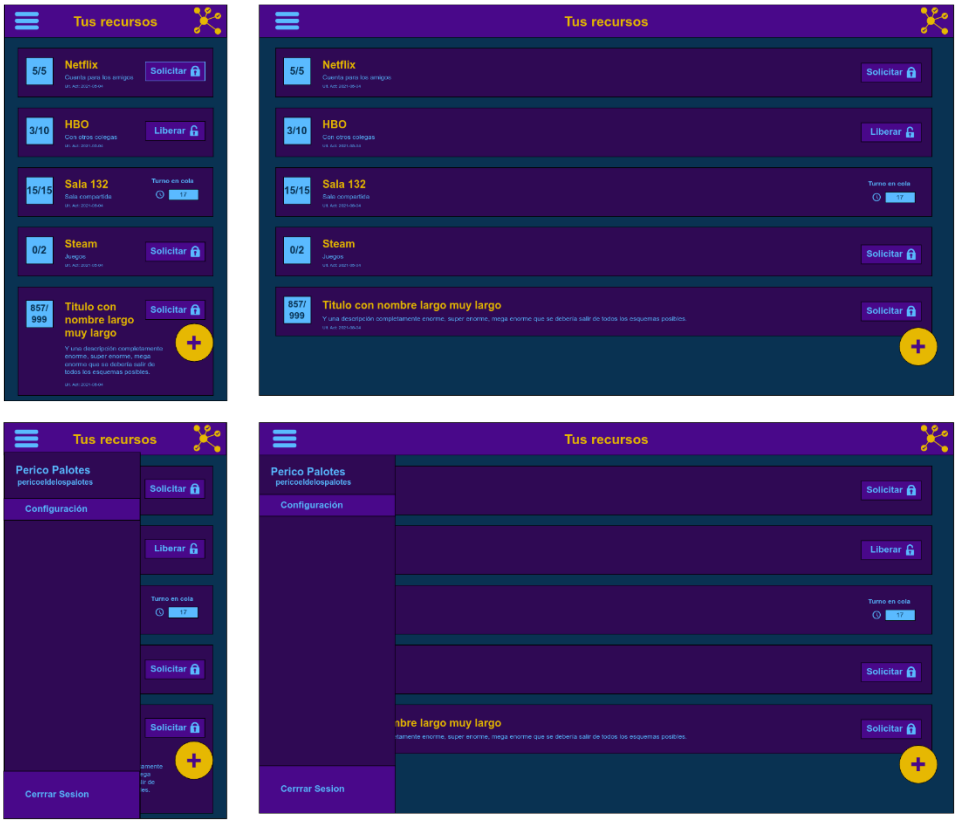
También se pueden visualizar en el archivo “wireframes.drawio”



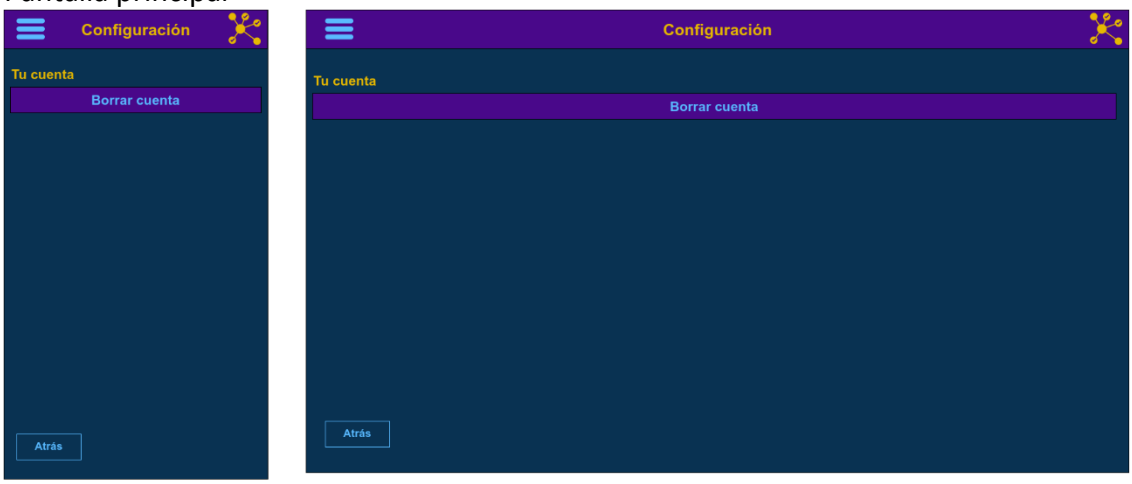


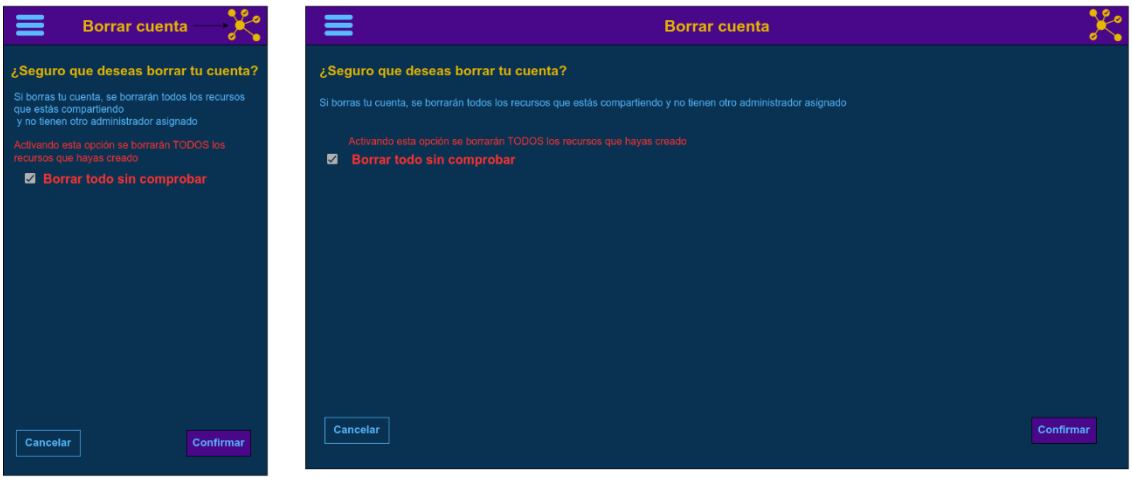
Inicio de sesión

El recuadro azul aparece cuando haces hover (sencillito)



Pantalla principal

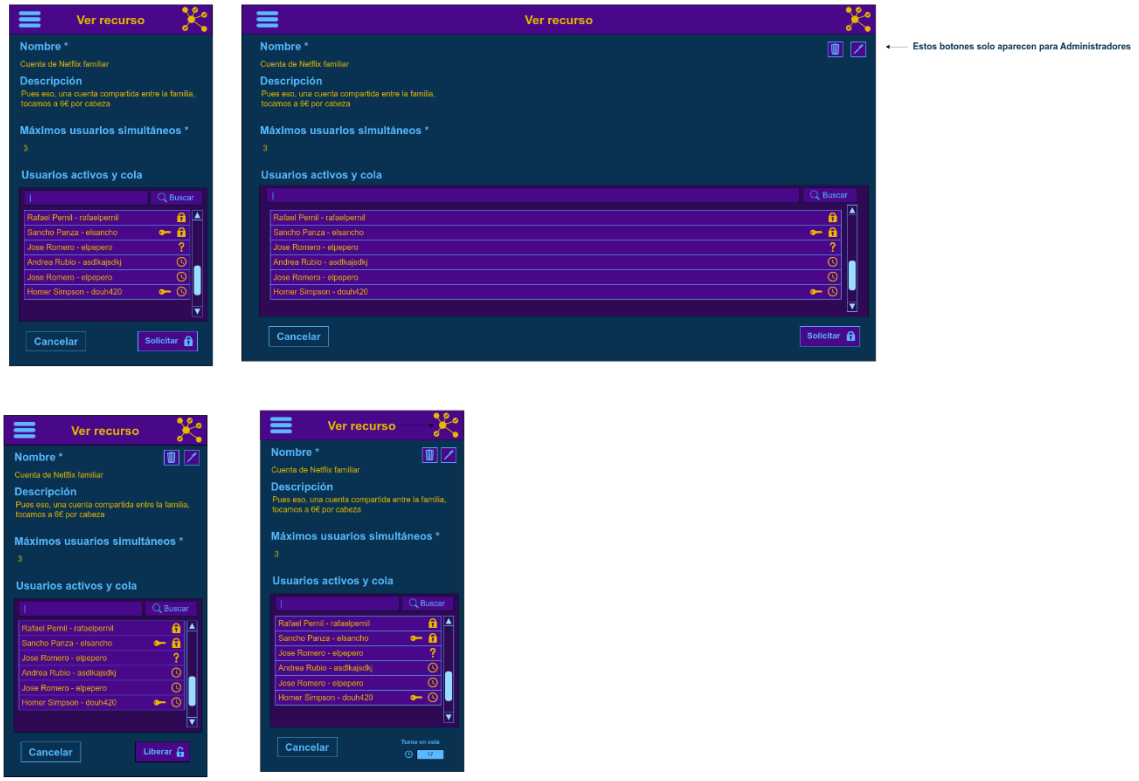




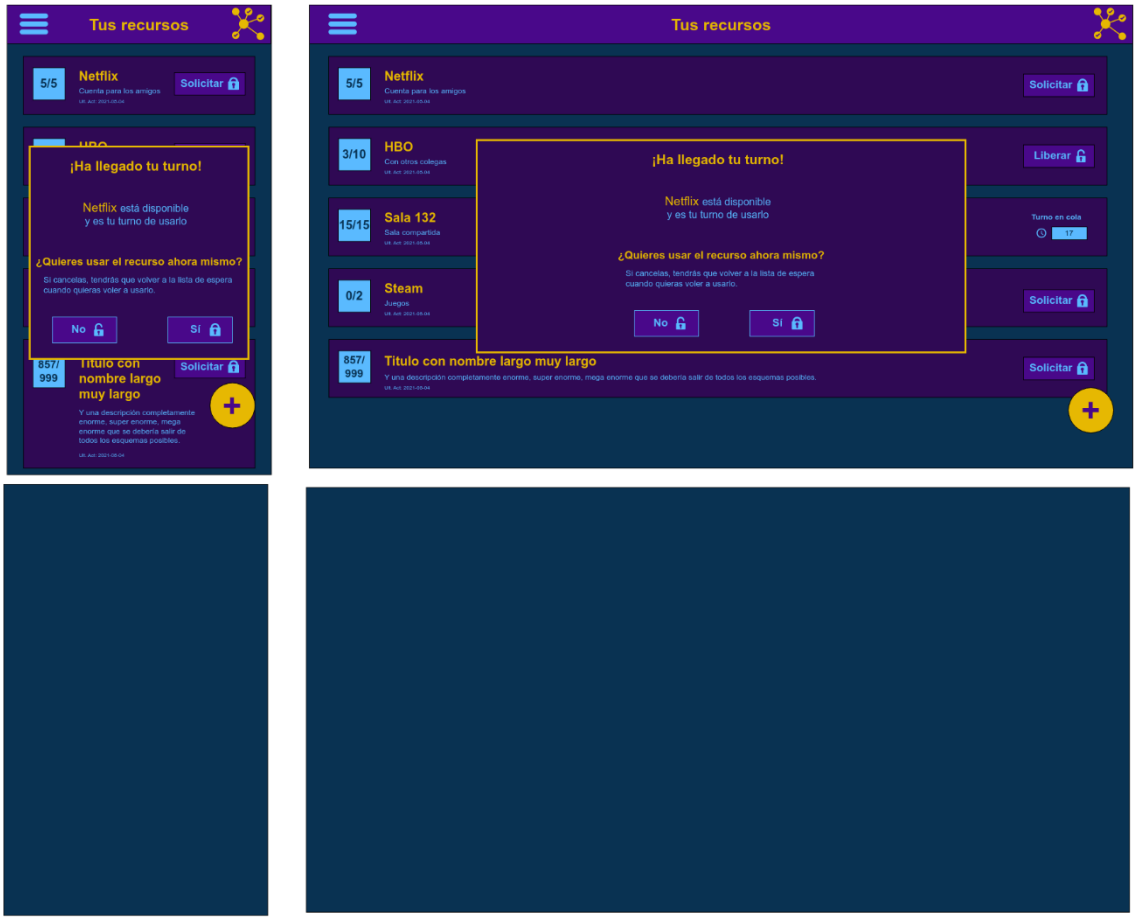
Borrar cuenta

Crear recurso

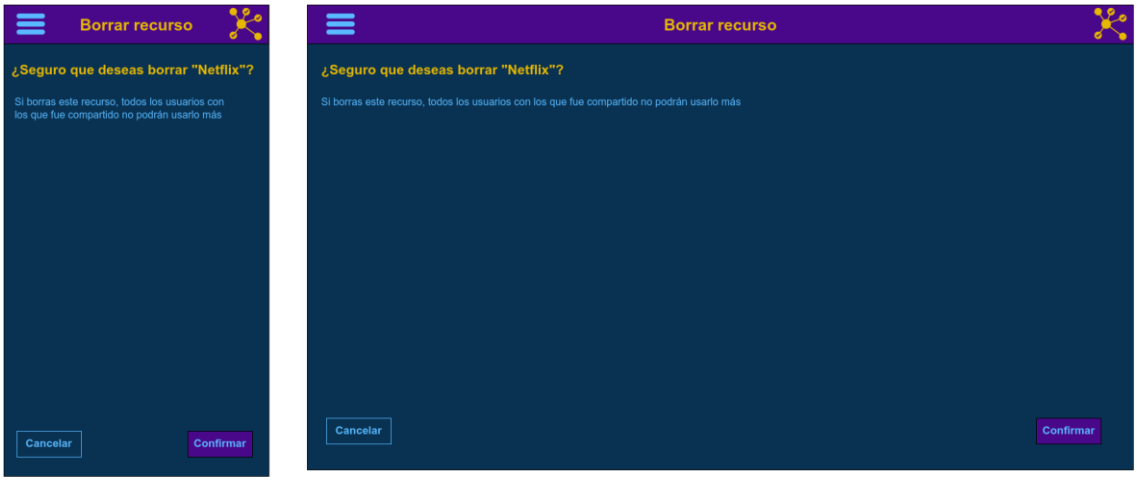
Editar recurso



Ver recurso



Diálogo emergente confirmación recurso



Borrar recurso

4.2 Casos de uso

Para definir la funcionalidad de la aplicación propuesta se han utilizado casos de uso.

Están en el archivo “casosdeuso.docx”

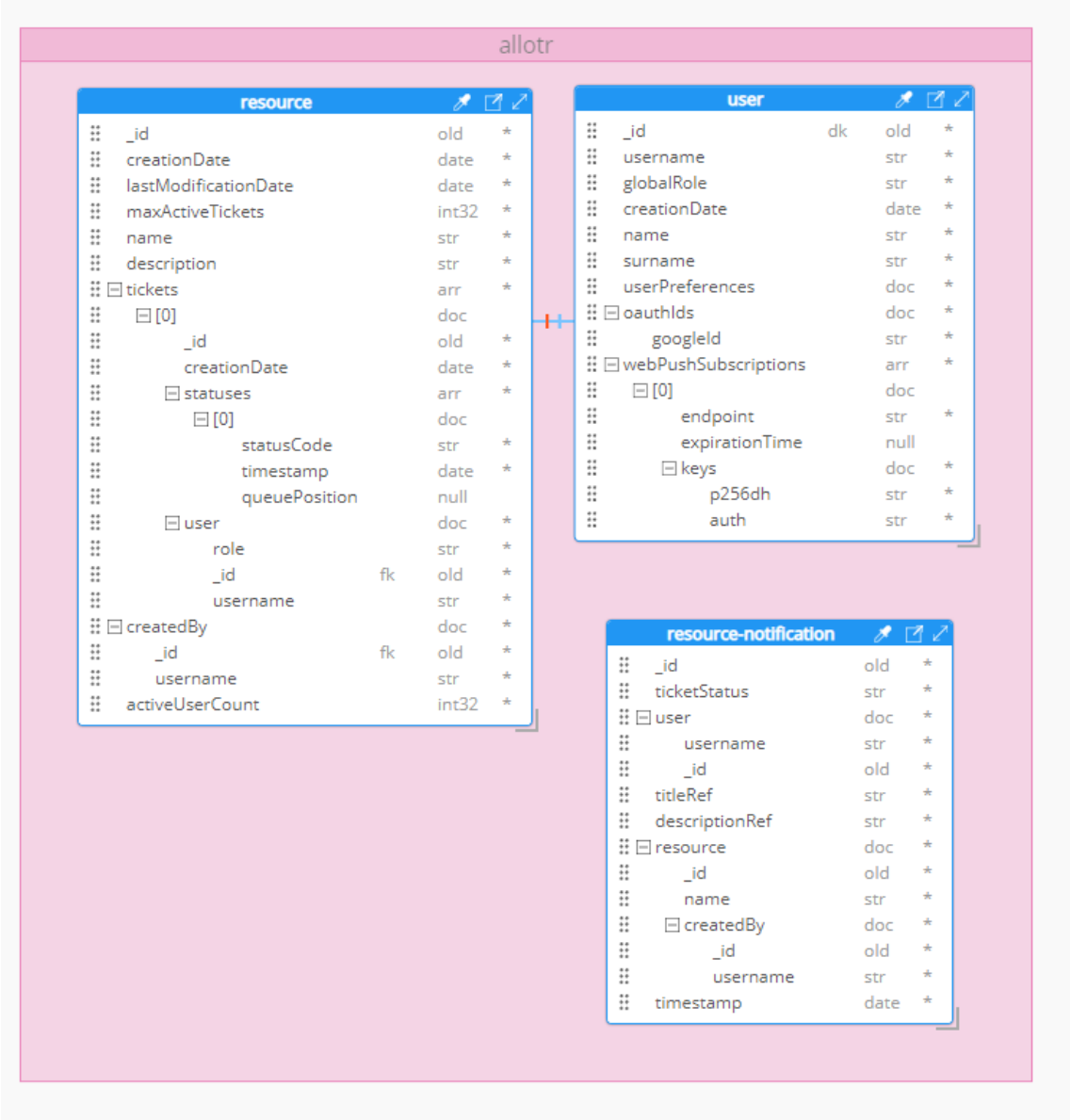
4.3 Casos de prueba

Todas las pruebas de la aplicación se encuentran en el conjunto de casos de prueba. Son pruebas funcionales.

Están en el archivo “casosdeprueba.docx”

4.4 Modelo de datos

El modelo de datos se ha diseñado con la herramienta Hackolade para diseñar modelos bases de datos no relacionales, más en concreto, de MongoDB



Nota: no se pueden visualizar todas las relaciones en el diagrama por ser la versión gratuita. Faltan 2 relaciones de clave foránea de “resource-notification.resource._id” con “resource._id” y de “resource-notification.user._id” con “user._id”.

Las relaciones de clave no se hacen automáticamente, se hacen al crear los datos, es una propiedad de las bases de datos no relacionales, no existen los “join”.

Por otra parte, el modelo de sesión no se encuentra porque se genera automáticamente por la librería “connect-mongo”
(connect-mongo - npmjs.com, s.f.)

4.5 Esquema GraphQL

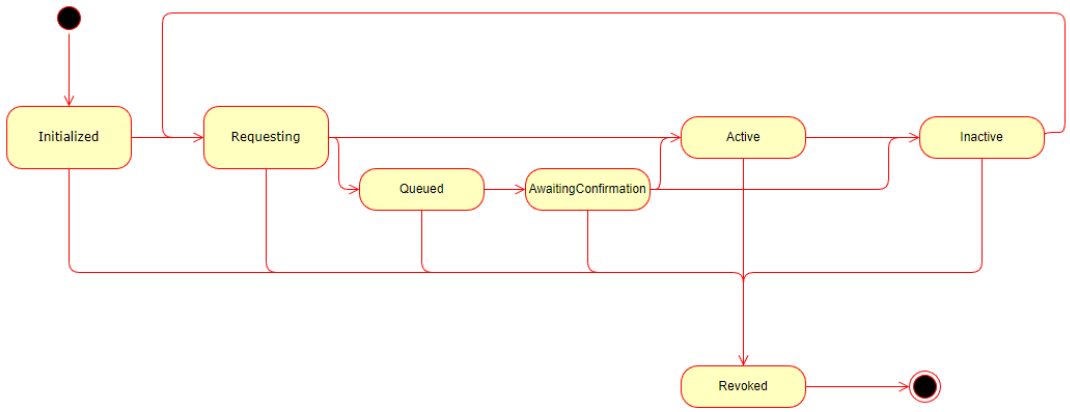
Los modelos de datos de MongoDB se han generado desde el esquema de GraphQL utilizando unas directivas especiales de GraphQL Codegen
(typescript-mongodb – GraphQL Code Generator, s.f.)

Las colecciones se crean con ese formato en cuanto se hace una inserción, es el poder de las bases de datos no relacionales.

Se puede consultar toda la documentación del esquema GraphQL en https://graphql-docs.allotr.eu/

4.6 Máquina de estados de boletos del recurso

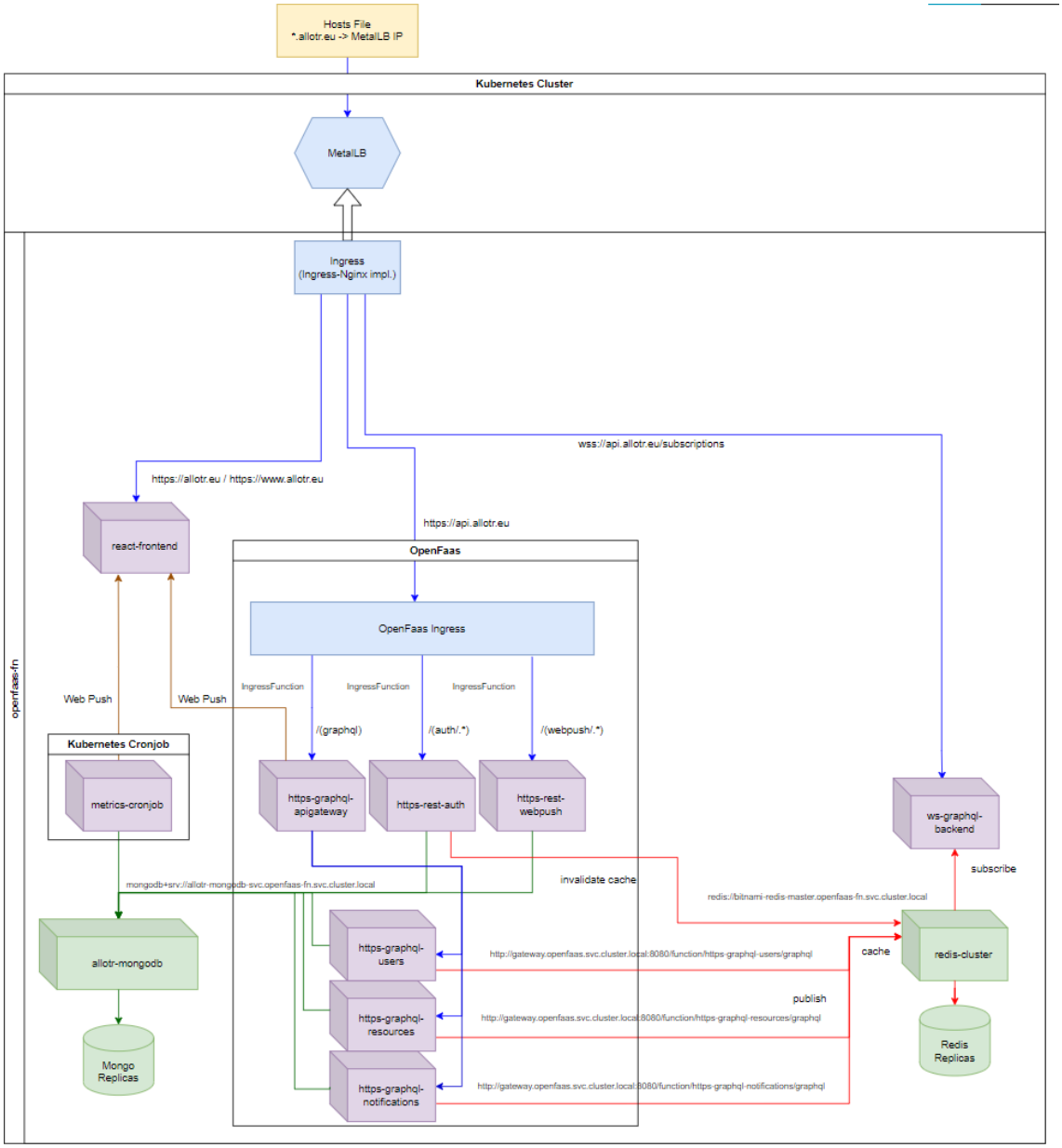
Un boleto de un usuario para un recurso presenta un conjunto limitado de transiciones válidas. Se define mediante esta máquina de estados:



También está disponible en el fichero “statemachine.drawio”

4.7 Esquema de conexiones Ingress

Este esquema muestra cómo se exponen los distintos componentes a internet y los endpoints asociados:



También se puede visualizar en el archivo “kubeconnections.drawio”

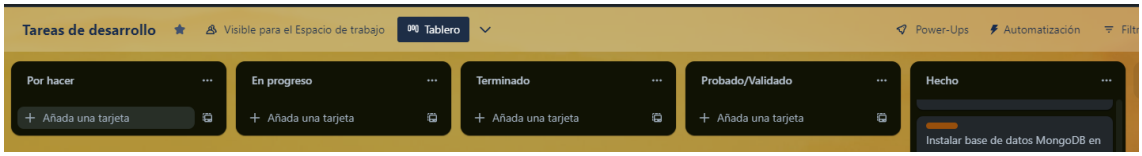
5

Implementación

5.1 Organización

Para la fase de implementación y organización de las tareas, se ha utilizado un tablero de Trello en el que se aplica la metodología Kanban.
(Metodología Kanban: revoluciona tu manera de trabajar, s.f.)

Se destinan las siguientes columnas: “Por hacer”, “En progreso”, “Terminado”, “Probado/Validado” y “Hecho”



Las tareas tienen las siguientes etiquetas por color:

Estas etiquetas identifican el tipo de tarea.

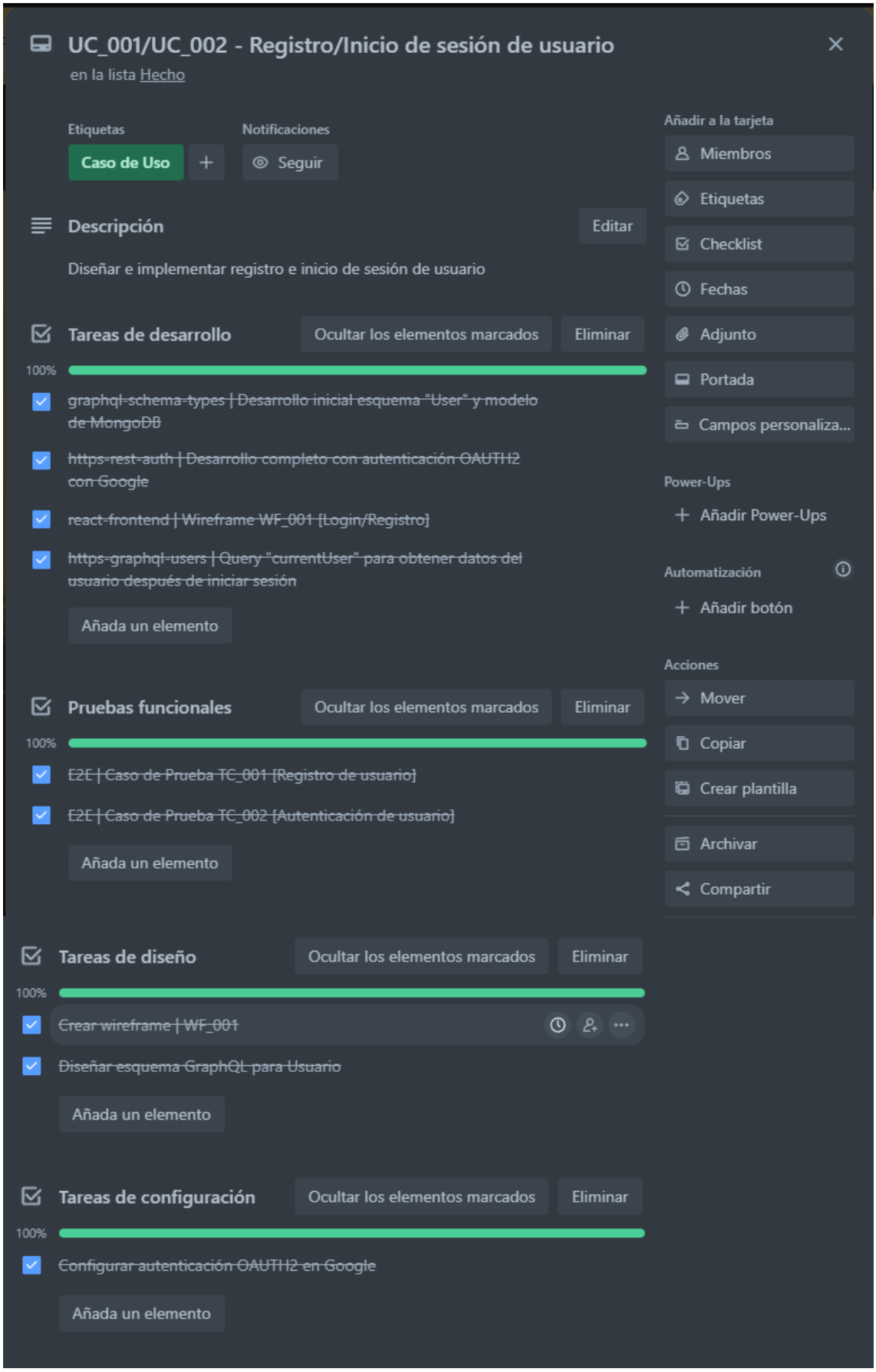
Las tareas de infraestructura son las primeras que se realizan debido a que sin ellas no hay entorno de despliegue.

Después, los casos de uso implementan la funcionalidad de la aplicación.

Las tareas de casos de uso se agrupan si tienen sentido a nivel de implementación, como es el caso del inicio de sesión de usuario que es exactamente igual que el registro.

Poseen distintas listas con casillas de verificación:

- Tareas de diseño -> Es el primer paso de la tarea, se diseña según se va implementando
- Tareas de configuración -> Estas tareas se crean cuando es necesario preparar algo externo a la tarea para realizarla
- Tareas de desarrollo -> Artefactos a modificar con breve descripción de la tarea a realizar
- Pruebas funcionales -> Enlace a casos de prueba y se marca Ok si la ejecución es correcta


UC_001/UC_002 - Registro/Inicio de sesión de usuario
en la lista Hecho
Etiquetas
Caso de Uso
Notificaciones
Seguir
Descripción
Editar
Diseñar e implementar registro e inicio de sesión de usuario
Tareas de desarrollo
Ocultar los elementos marcados
Eliminar
100%
graphql-schema-types | Desarrollo inicial esquema "User" y modelo de MongoDB
https-rest-auth | Desarrollo completo con autenticación OAUTH2 con Google
react-frontend | Wireframe WF_001 [Login/Registro]
https-graphql-users | Query "currentUser" para obtener datos del usuario después de iniciar sesión
Añada un elemento
Pruebas funcionales
Ocultar los elementos marcados
Eliminar
100%
E2E | Caso de Prueba TC_001 [Registro de usuario]
E2E | Caso de Prueba TC_002 [Autenticación de usuario]
Añada un elemento
Tareas de diseño
Ocultar los elementos marcados
Eliminar
100%
Crear wireframe | WF_001
Diseñar esquema GraphQL para Usuario
Añada un elemento
Tareas de configuración
Ocultar los elementos marcados
Eliminar
100%
Configurar autenticación OAUTH2 en Google
Añada un elemento
Añadir a la tarjeta
Miembros
Etiquetas
Checklist
Fechas
Adjunto
Portada
Campos personaliza...
Power-Ups
Añadir Power-Ups
Automatización
Añadir botón
Acciones
Mover
Copiar
Crear plantilla
Archivar
Compartir

Se puede visualizar aquí el tablero: https://trello.com/b/vgVCh2IJ/tareas-de-desarrollo

5.2 Repositorios y versionado de código

A la hora de subir el código, se ha creado una organización de GitHub llamada Allotr https://github.com/Allotr

Por ende, el versionado del código se hace mediante Git y se almacena en GitHub.

Esta es la lista de proyectos en uso para el proyecto:

- react-frontend – Aplicación web en React
- https-graphql-apigateway – Punto de entrada a API GraphQL de la aplicación
- https-graphql-users – Microservicio GraphQL privado para esquema de Usuarios
- https-graphql-resources – Microservicio GraphQL privado para esquema de Recursos
- https-graphql-notifications – Microservicio GraphQL privado para esquema de Notificaciones
- https-rest-auth – Microservicio REST de autenticación con Express y Passport.js
- https-rest-webpush – Microservicio REST para registrar service worker y guardar los endpoints con los que enviar notificaciones push
- ws-graphql-backend – Servicio GraphQL con WebSockets para suscripciones. Se separa del API Gateway porque OpenFaaS no soporta WebSockets y este proyecto debe ser diferente.
- metrics-cronjob – Tarea programada que analiza aquellos recursos que se solicitan varios días seguidos a la misma hora y notifica a aquellos usuarios que no lo han solicitado hoy
- config-files – Repositorio centralizado con pasos de instalación y todos los ficheros de configuración con secretos de configuración
- graphql-schema-types – Paquete NPM con esquemas GraphQL y tipos Typescript para interactuar tanto en los microservicios, como en la aplicación web de forma unificada
- allotr.github.io – Página web de documentación de la API GraphQL

https-graphql-backend está obsoleto. Es el backend de la primera versión de este proyecto realizado en 2021 que no se llegó a presentar. Ese proyecto no implementa una arquitectura de microservicios y por eso se ha descartado

5.3 Generación de imagen Docker e integración con Kubernetes

Todos los proyectos desplegables contienen un archivo Dockerfile que define cómo se deben empaquetar para generar la imagen a desplegar. A su vez, también incluyen un script llamado “buildMultiArch.sh” con el que se compila y sube una versión para arquitectura ARM y x86_64.

Las imágenes Docker están subidas a https://hub.docker.com/u/rafaelpernil

Para las credenciales y endpoints, se usan variables de entorno en OpenFaaS y secretos de Kubernetes para ws-graphql-backend y metrics-cronjob

Aquellos empaquetados con OpenFaaS, incluyen un manifiesto de despliegue donde se especifica la plantilla y las variables de entorno (véase sección 4.7)

```yaml
version: 1.0
provider:
  name: openfaas
  gateway: https://openfaas.allotr.eu
functions:
  https-graphql-apigateway:
    lang: node-typescript-uwebsockets
    handler: ./https-graphql-apigateway
    image: rafaelpernil/https-graphql-apigateway:latest
    environment:
      SESSION_SECRET: ...
```

Como caso especial, para react-frontend, se compila la imagen Docker con las variables de entorno incrustadas ya que este código se ejecuta en lado de cliente. Desde el servidor sólo se sirven los archivos de la web.

Todas las credenciales se guardan en el repositorio privado config-files.

5.4 Despliegue en Kubernetes

Todos los proyectos desplegables incluyen un script llamado “deploy.sh” que fuerza la regeneración del despliegue para que Kubernetes descargue la nueva imagen.

Por facilitar, hay cuatro scripts extra en el repositorio config-files:

- copy_env_dev.sh -> Copia y aplica todos los archivos de configuración con credenciales a cada proyecto para el entorno de desarrollo/máquina virtual
- copy_env_prod.sh -> Copia y aplica todos los archivos de configuración con credenciales a cada proyecto para el entorno de producción/Raspberry Pi
- deploy_frontend.sh -> Despliega la aplicación web react-frontend

- deploy_backend.sh -> Despliega todos los servicios de backend en el orden correcto

5.5 Generación de documentación GraphQL

La documentación de la API GraphQL se genera con la herramienta Graphdoc (Graphdoc, s.f.) y se aloja en GitHub Pages de forma gratuita

5.6 Acceso como entidad externa a la aplicación

Como funcionalidad trasversal, todas las queries y mutaciones de GraphQL dependientes de la sesión del usuario, permiten acceso administrador global, proporcionando el identificador de un usuario en nombre del cual se realiza la tarea.

Es decir, un administrador global puede realizar tareas en nombre de otro usuario.

De esta manera, una entidad externa podría desarrollar un panel de control donde gestionar las peticiones sin problema.

Y la autenticación está preparada para funcionar en cualquier subdominio de https://allotr.eu

Es decir, cualquier entidad externa que se quiera adherir, tendrá que solicitar que se redirija el tráfico de un nuevo subdominio de “allotr.eu”, como podría ser “external-app.allotr.eu” a su cliente web

5.7 Acceso a base de datos seguro respetando transacciones

Para lograr este objetivo y proteger el uso del recurso de condiciones de carrera, se utilizan las transacciones de MongoDB
(Transactions — MongoDB Manual, s.f.)

Las transacciones aseguran que sólo se escriben los cambios cuando todas las operaciones de la transacción han funcionado correctamente

5.8 Implementación de caché con Redis

Los microservicios https-graphql-users, https-graphql-resources y https-graphql-notifications cuentan con una capa de caché en las respuestas.

Esto se logra gracias a un plugin de GraphQL Yoga llamado @graphql-yoga/plugin-response-cache y adicionalmente a @envelop/response-cache-redis

```
plugins: [
  useGraphQlJit(),
  useParserCache(),
  useResponseCache({
    // cache based on the authorization header
    session: request => {
      return request.headers.get('authorization')
    },
    shouldCacheResult: async ({ result }) => {
      // If cache writes are locked, don't cache
      const isWriteLocked = await isCacheWriteLocked();
      if (isWriteLocked) {
        return false;
      }

      const functionBlacklist = [
        // Add functions to blacklist
      ]

      const data = result?.data as any;
      // Check that result is not an error
      const hasOkValue = !_.isEmpty(result?.data) && _.isEmpty(result?.errors);
      // Check only fields in data
      const fieldsAvailable = queryNames.filter(field => field in (data ?? {}));
      // Check value is not empty
      const isValidValue = fieldsAvailable.every(query => !_.isEmpty(data?.[query]));
      // Check function is valid
      const isValidFunction = functionBlacklist.every(key => data?.[key] == null);

      return hasOkValue && isValidValue && isValidFunction
    },
    cache
  })
```

Ambas librerías en conjunto guardan correctamente todo en caché. El problema surge a la hora de invalidar la caché.

Todas aquellas respuestas vacías o con error, la librería decide insertarlas en la caché y como no tienen ningún objeto relacionado, no es capaz de encontrar la clave donde se encuentra el valor en caché y eliminarlo, provocando que la aplicación muestre permanentemente la misma información.

La solución implementada ignora respuestas vacías, con error o aquellas cuya operación esté en una lista negra.

Por otra parte, es importante que el microservicio REST de autenticación invalide la caché al cerrar sesión para que la API detecte que la sesión no está iniciada.

```
app.get("/auth/google/logout", async (req, res, next) => {
    // Clean cached resources
    clearCache();
```

Si hay una respuesta en caché para la sesión proporcionada, los microservicios no comprueban que la sesión siga siendo válida, se limitan a devolver el valor en caché.

5.9 Funcionamiento de Aplicación Web Progresiva

Para que el navegador considere que la aplicación web es una aplicación web progresiva debe cumplir unos criterios de calidad.

- Debe tener un manifiesto con los datos necesarios como el nombre de la aplicación y una serie de iconos
- Debe estar bajo HTTPS para garantizar tráfico seguro
- Debe tener un service worker

(How to make PWAs installable, s.f.)

El punto más interesante de los tres es el service worker. Es un proceso que se ejecuta en segundo plano y que se encarga de cachear aquellos recursos que se le indiquen, para dar funcionalidad offline y reducir en gran medida la carga inicial de datos.

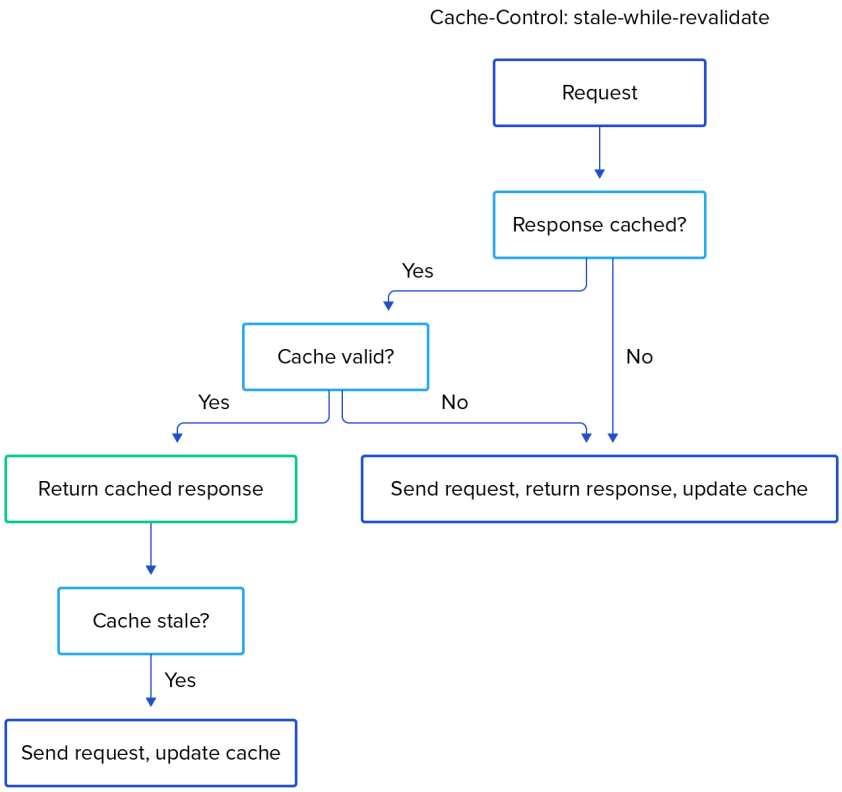
En este caso almacena en caché imágenes, recursos estáticos y la clave pública para registrar el endpoint WebPush

```
registerRoute(/\.(?:js|css)$/,
    new StaleWhileRevalidate({
        cacheName: 'static-resources',
        plugins: [
            new ExpirationPlugin({
                maxAgeSeconds: 24 * 60 * 60, // 24 hours
            }),
        ],
    })
);

registerRoute(/\.(?:png|gif|jpg|svg)$/,
    new CacheFirst({
        cacheName: 'images-cache',
        plugins: [
            new ExpirationPlugin({
                maxAgeSeconds: 24 * 60 * 60, // 24 hours
            }),
        ],
    })
);

registerRoute('https://api.allotr.eu/webpush/vapidPublicKey',
    new CacheFirst({
        cacheName: 'vapidPublicKey',
        plugins: [
            new ExpirationPlugin({
                maxAgeSeconds: 24 * 60 * 60, // 24 hours
            }),
        ],
    })
);
```

Para los archivos .js y .css utiliza la técnica StaleWhileRevalidate



(Aryan)

Esta técnica se asegura que los datos en caché se mantengan relativamente frescos.

Trae los datos de caché si existen y siguen siendo válidos, pero paralelamente actualiza el valor de la caché con la última versión para futuras peticiones.

6

Pruebas

Para esta aplicación se han realizado pruebas funcionales que se pueden ver en el documento de casos de prueba llamado “casosdeprueba.docx”

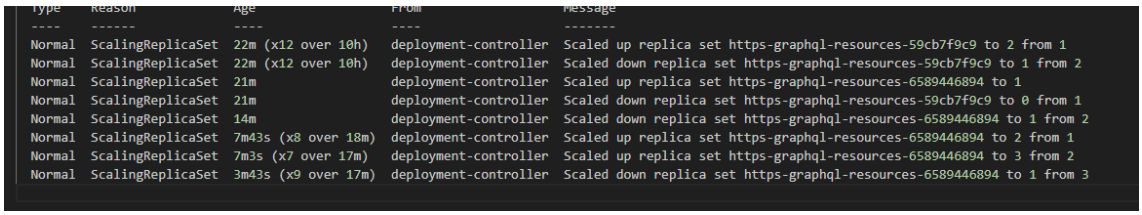
La aplicación se puede utilizar al completo de manera pública en https://allotr.eu

De forma adicional a los casos de prueba, se ha probado de forma exhaustiva la aplicación en busca de condiciones de carrera a la hora de solicitar y liberar recursos.

Hoy, bajo todas las pruebas realizadas, debería ser completamente seguro el estado de un recurso.

A nivel de interfaz web, se han realizado optimizaciones y correcciones para obtener una puntuación de 100/100 en Google Lighthouse

Por último, se ha probado la escalabilidad de OpenFaaS y cuando supera cierto umbral de consumo de recursos en el clúster, se generan más réplicas:



Type	Reason	Age	From	Message
Normal	ScalingReplicaSet	22m (x12 over 10h)	deployment-controller	Scaled up replica set https-graphql-resources-59cb7f9c9 to 2 from 1
Normal	ScalingReplicaSet	22m (x12 over 10h)	deployment-controller	Scaled down replica set https-graphql-resources-59cb7f9c9 to 1 from 2
Normal	ScalingReplicaSet	21m	deployment-controller	Scaled up replica set https-graphql-resources-6589446894 to 1
Normal	ScalingReplicaSet	21m	deployment-controller	Scaled down replica set https-graphql-resources-59cb7f9c9 to 0 from 1
Normal	ScalingReplicaSet	14m	deployment-controller	Scaled down replica set https-graphql-resources-6589446894 to 1 from 2
Normal	ScalingReplicaSet	7m43s (x8 over 18m)	deployment-controller	Scaled up replica set https-graphql-resources-6589446894 to 2 from 1
Normal	ScalingReplicaSet	7m3s (x7 over 17m)	deployment-controller	Scaled up replica set https-graphql-resources-6589446894 to 3 from 2
Normal	ScalingReplicaSet	3m43s (x9 over 17m)	deployment-controller	Scaled down replica set https-graphql-resources-6589446894 to 1 from 3

7

Documentación de API

La API se encuentra completamente documentada con acceso directo al diagrama de arquitectura y consola GraphiQL en la siguiente página:

https://graphql-docs.allotr.eu/

8

Conclusiones

Se ha conseguido demostrar la viabilidad y escalabilidad de una arquitectura de microservicios con GraphQL y desplegada como FaaS en Kubernetes.

Como mejoras a futuro, se plantea añadir pruebas automatizados en todas las capas.

Referencias

Adobe Color. (s.f.). Obtenido de https://color.adobe.com/es/

Alain2020. (23 de Diciembre de 2020). *SSR vs CSR - dev.to*. Obtenido de https://dev.to/alain2020/ssr-vs-csr-2617

Aplicaciones web progresivas - web.dev. (s.f.). Obtenido de https://web.dev/i18n/es/progressive-web-apps/

Aryan, A. (s.f.). *Stale-while-revalidate Data Fetching with React Hooks: A Guide.* Obtenido de https://www.toptal.com/react-hooks/stale-while-revalidate

Brown, S. (2011). *The C4 model for visualising software architecture*. Obtenido de https://c4model.com/

Certificados SSL/TLS Gratuitos - Let's Encrypt. (s.f.). Obtenido de https://letsencrypt.org/es/

cert-manager. (s.f.). Obtenido de https://cert-manager.io/

Cliente-servidor - Wikipedia, la enciclopedia libre. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Cliente-servidor

Cloudflare - cert-manager Documentation. (s.f.). Obtenido de https://cert-manager.io/docs/configuration/acme/dns01/cloudflare/

connect-mongo - npmjs.com. (s.f.). Obtenido de https://www.npmjs.com/package/connect-mongo

Copeland, R. (2013). *MongoDB Applied Design Patterns. En R. Copeland, MongoDB Applied Design Patterns (págs. 9-10). O'Reilly*. Obtenido de https://www.oreilly.com/library/view/mongodb-applied-design/9781449340056/ch01.html

Demystifying the Open Container Initiative (OCI) Specifications. (19 de Julio de 2017). Obtenido de https://www.docker.com/blog/demystifying-open-container-initiative-oci-specifications/

Diagrama de arquitectura Allotr. (Septiembre de 2023). Marbella, Málaga, España.

Documentación de Kubernetes. (s.f.). Obtenido de https://kubernetes.io/es/docs

Ellis, A. (s.f.). *OpenFaaS - Serverless Functions Made Simple*. Obtenido de https://www.openfaas.com/

Express.js - Wikipedia. (s.f.). Obtenido de https://en.wikipedia.org/wiki/Express.js

Function as a service - Wikipedia. (s.f.). Obtenido de https://en.wikipedia.org/wiki/Function_as_a_service

Graphdoc. (s.f.). Obtenido de https://github.com/2fd/graphdoc#readme

GraphQL - Wikipedia, la enciclopedia libre. (s.f.). Obtenido de https://es.wikipedia.org/wiki/GraphQL

Helm Charts to deploy Redis® in Kubernetes. (s.f.). Obtenido de https://bitnami.com/stack/redis/helm
Home – GraphQL Stitching. (s.f.). Obtenido de https://the-guild.dev/graphql/stitching/docs
How To GraphQL. (s.f.). (s.f.). Obtenido de https://www.howtographql.com/
How to make PWAs installable. (s.f.). Obtenido de https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Installable_PWAs
How to Setup Redis Master and Slave Replication? - EDUCBA. (s.f.). Obtenido de https://www.educba.com/redis-master-slave/
IngressOperator for OpenFaaS. (s.f.). Obtenido de https://github.com/openfaas/ingress-operator
Install Docker Engine on Ubuntu - Docker Docs. (s.f.). Obtenido de https://docs.docker.com/engine/install/ubuntu/
Introduction – GraphQL Stitching. (s.f.). Obtenido de https://the-guild.dev/graphql/stitching/docs
Lafuente, A. (1 de Agosto de 2018). *Bases de datos relacionales vs. no relacionales: ¿qué es mejor? Obtenido de Aukera*. Obtenido de https://aukera.es/blog/bases-de-datos-relacionales-vs-no-relacionales/
Learn, M. (s.f.). *Estilo de arquitectura de microservicios*. Obtenido de https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices
Metodología Kanban: revoluciona tu manera de trabajar. (s.f.). Obtenido de https://blog.trello.com/es/metodologia-kanban
MongoDB Atlas Database | Multi-Cloud Database Service. (s.f.). Obtenido de https://www.mongodb.com/atlas/database
MongoDB Community Kubernetes Operator. (s.f.). Obtenido de https://github.com/mongodb/mongodb-kubernetes-operator
Multi-platform images - Docker Docs. (s.f.). Obtenido de https://docs.docker.com/build/building/multi-platform/
Neves, T. d. (15 de Junio de 2013). *Un diagrama cliente-servidor vía Internet*. Obtenido de https://es.wikipedia.org/wiki/Cliente-servidor#/media/Archivo:Cliente-Servidor.png
Node.js. (s.f.). Obtenido de https://nodejs.org/es/about/
Open Container Initiative - Wikipedia, la enciclopedia libre. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Open_Container_Initiative
OpenFaaS Classic templates. (s.f.). Obtenido de https://github.com/openfaas/templates
Passport.js. (s.f.). Obtenido de http://www.passportjs.org/features/
Pernil, R. (4 de Junio de 2018). Presentación Kubernetes - Calidad de Software. Málaga, Málaga, España.
Push API. (s.f.). Obtenido de https://developer.mozilla.org/en-US/docs/Web/API/Push_API
React - Una biblioteca de JavaScript para construir interfaces de usuario. (s.f.). Obtenido de https://es.reactjs.org/
Redis. (s.f.). Obtenido de https://redis.io/
Serverless computing - Wikipedia. (s.f.). Obtenido de https://en.wikipedia.org/wiki/Serverless_computing

Service Worker API. (s.f.). Obtenido de https://developer.mozilla.org/es/docs/Web/API/Service_Worker_API

Slingerland, C. (20 de July de 2022). *K3s Vs K8s: What's The Difference? (And When To Use Each)*. Obtenido de https://www.cloudzero.com/blog/k3s-vs-k8s

TLS on Kubernetes. (s.f.). Obtenido de https://docs.openfaas.com/reference/ssl/kubernetes-with-cert-manager/

Transactions — MongoDB Manual. (s.f.). Obtenido de https://www.mongodb.com/docs/manual/core/transactions/

Transferencia de Estado Representacional - Wikipedia. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional

TypeScript: JavaScript With Syntax For Types. (s.f.). Obtenido de https://www.typescriptlang.org/

typescript-mongodb – GraphQL Code Generator. (s.f.). Obtenido de https://the-guild.dev/graphql/codegen/plugins/typescript/typescript-mongodb

Web Content Accessibility Guidelines (WCAG) 2. (s.f.). Obtenido de https://www.w3.org/WAI/WCAG2AAA-Conformance

WebSocket API | Can I use... (s.f.). Obtenido de https://caniuse.com/mdn-api_websocket

WebSockets - Referencia de la API Web | MDN. (s.f.). Obtenido de https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

Websockets and HTTP mode. (s.f.). Obtenido de https://github.com/openfaas/of-watchdog/issues/20

Apéndice A

Manual de Instalación

Requerimientos:

- Máquina virtual/local con Ubuntu u otro sistema Linux
- Arquitectura x86_64
- Acceso al repositorio de configuración y despliegue de la aplicación en https://github.com/Allotr/config-files (se proporciona en el código fuente)

Pasos:

1. Iniciar la máquina con Ubuntu u otro sistema Linux
2. Abrir una terminal y lanzar “sudo su”
3. Introducir credenciales del usuario
4. Copiar archivo “install_vm.sh” del repositorio “config-files” a una carpeta vacía
5. Otorgarle permiso de ejecución con “chmod +x install_vm.sh”
6. Ejecutar script con “./install_vm.sh”
7. Esperar unos 20 minutos

Nota: Si falla el despliegue de los servicios de OpenFaas la primera vez, volver a iniciar sesión y desplegar:

PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)
echo -n $PASSWORD | faas-cli login --username admin --password-stdin --gateway https://openfaas.allotr.eu/

Y dentro del repositorio “config-files”, lanzar “./deploy_backend.sh”

Como alternativa, ofrezco una máquina virtual completamente configurada con la aplicación corriendo en la última instantánea:
https://drive.google.com/drive/folders/1TkJ7bmJmDNyugr6ZR1nH2AHU1a1jh91L?usp=sharing