|  |  |
| --- | --- |
|  | **2020** |
|  | IES San Vicente  Francisco José Ferrer Rodríguez |

|  |
| --- |
| **[GestlINE]** |
| GestLine es una aplicación web que simula un área de clientes de una compañía de telecomunicaciones. |

Contenido

[1. Introducción 3](#_Toc40018040)

[2. Antecedentes 5](#_Toc40018041)

[3. Análisis 6](#_Toc40018042)

[4. Diseño 8](#_Toc40018043)

[4.1. Angular (Frontend) 8](#_Toc40018044)

[4.2. Microservicios (Stack Environment) 9](#_Toc40018045)

[4.3. Microservicios (APIs) 10](#_Toc40018046)

[4.4. Flujos de trabajo 10](#_Toc40018047)

[5. Resultados 11](#_Toc40018048)

[6. Conclusiones 11](#_Toc40018049)

[7. Bibliografía 11](#_Toc40018050)

# 1. Introducción

Hoy en día vivimos en la sociedad 2.0 donde las plataformas digitales dominan la forma de interactuar sobre el mundo, pero, realmente, ¿Esto siempre ha sido así? La respuesta es no, por ello han tenido que evolucionar para que sus servicios escalen a toda la demanda bajo un coste sostenible, además de que el mantenimiento y el desarrollo de las mismas sea algo sencillo.

En un corto espacio de tiempo nuestros teléfonos pasaron a ser inteligentes, nuestros coches, incluso electrodomésticos se conectan a internet. Gran parte de esta evolución la Web, que ha ido creciendo con el paso del tiempo. Parece que Internet lleva con nosotros mucho tiempo, pero nada más lejos de la realidad: apenas llevamos un par de décadas conviviendo juntos. Un periodo de tiempo breve pero intenso, que ha servido para que Internet haya experimentado una evolución imparable, y donde el diseño web se ha convertido en un pilar imprescindible para cualquier sitio web.

Venimos trabajando de manera monolítica. En nuestras aplicaciones, lo más común es encontrarnos con una arquitectura por capas, en la cual tenemos una única base de código con múltiples módulos. Normalmente, está la capa de Presentación (UI), la de lógica de negocio (Business Logic), y la de Acceso a Datos (Data Access). Si se hace un cambio en alguno de estos módulos, por más chico que sea, debemos redeployar la aplicación completa. De igual forma, sabemos que la escalabilidad se convierte en un reto, ya que todos los módulos deben ir a la par de manera horizontal para escalar el sistema completo, creando un riesgo de acoplamiento inherente a esta arquitectura. Por lo visto, se vuelve muy difícil cambiar la tecnología, lenguaje, o Framework, ya que toda la aplicación está fuertemente acoplada y los componentes son dependientes entre sí.

La arquitectura por microservicios viene dada por una colección de servicios vagamente acoplados, donde un proceso se divide en varios (microservicios) y cada uno debe tener solo una única responsabilidad.

A diferencia de las aplicaciones monolíticas, que deben escalar completas, desarrollar bajo la arquitectura de microservicios nos permite escalar cada microservicio independientemente. De esta manera, podemos cubrir la demanda o aumentar la funcionalidad de cada área que lo necesite, sin afectar a las demás. La composición de aplicaciones por microservicios permite mejorar la integración y entrega continua de paquetes, ya que podemos hacer deploy parcial de un servicio puntual. De igual manera, cada microservicio puede estar asilado bajo su propio entorno y tecnología que se adecue a la responsabilidad particular de este.

Gestline pretende simular un área de clientes de una compañía de telecomunicaciones. Una vez dentro del aplicativo, el cliente dispondrá de un acceso vía Login (mediante su documento de identidad o email asociado). Una vez verificado el acceso y dentro del sistema, podrá gestionar sus líneas y tarifas contratadas, visualizar sus consumos en llamadas, datos, bonos, etc. Gestionar sus servicios contratados, visualizar y descargar sus facturas o modificar su información personal, entre otras cosas.

Para el desarrollo del proyecto, se ha decidido implantar un sistema distribuido para la parte backend basándonos en una arquitectura de microservicios utilizando la solución desarrollada por Netflix “Spring Cloud Netflix” (Netflix OSS). Dicho Framework está construido sobre Spring Boot, el cual proporciona auto-configuraciones básicas para facilitar el desarrollo de los microservicios, además de proporcionar ya un contenedor de aplicaciones (Tomcat, Jetty, Undertow) embebido. Spring Cloud le ofrece al desarrollador las herramientas necesarias para la construcción de patrones comunes en sistemas distribuidos.

Cada microservicio, siguiendo uno de sus principios, es dueño de sus propios datos de manera independiente. Es por esto que cada microservicio dispone de su propia base de datos. En este caso para el desarrollo del proyecto he decidido utilizar bases de datos H2.

El propósito de H2 es agilizar el proceso de desarrollo. H2 puede crear la estructura de las tablas basándose en nuestras entidades JPA que permite cargar datos iniciales para nuestras pruebas, nos permitirá hacer transacciones CRUD como cualquier otra base datos con la diferencia que serán temporales, es decir los datos persistirán durante la ejecución, pero regresarán al estado original respetando el script inicial de carga.

Y para la parte frontal, se ha utilizado el Framework Angular en la versión 8, por lo que desarrollaremos nuestra aplicación utilizando el concepto SPA (Single Page Application) lo cual nos permite una serie de ventajas ya que la carga de datos se realiza de manera dinámica, casi instantánea, asíncronamente haciendo llamadas al servidor (backend con un API REST) y sobre todo sin tener que refrescar la página en ningún momento. Es decir, las aplicaciones web que podemos hacer con Angular son reactivas y no recargan el navegador.

# 2. Antecedentes

La arquitectura de microservicios, es un distintivo sistema de desarrollo de software que ha crecido en popularidad en los últimos años.

Gracias a su sencilla escalabilidad, este método de arquitectura se considera especialmente adecuado cuando se tiene que procurar la compatibilidad con un amplio sector de diferentes plataformas (IoT, web, móvil, wearables…) o simplemente cuando no sabemos a ciencia cierta hacia qué tipo de dispositivos estamos orientando nuestro trabajo.

No hay mejor forma de conocer el alcance que ha tenido este método de desarrollo que ver quiénes lo han implementado. Multitud de webs que sirven aplicaciones a gran escala han decidido invertir en la evolución hacia los microservicios en vistas de un futuro donde el mantenimiento y escalabilidad de sus productos es mucho más simple, efectivo y rápido. Vamos a destacar algunas de estas compañías, que lo mismo hasta os suenan:



* **Netflix**: Esta plataforma tiene una arquitectura generalizada que desde hace ya un par de años (coincidiendo con su “boom” en U.S.A.) se pasó a los microservicios para el funcionamiento de sus productos. A diario recibe una media de mil millones de llamadas a sus diferentes servicios (se dice que es responsable del 30% del tráfico de Internet) y es capaz de adaptarse a más de 800 tipos de dispositivos mediante su API de streaming de vídeo, la cual, para ofrecer un servicio más estable, por cada solicitud que le pedimos, ésta realiza cinco solicitudes a diferentes servidores para no perder nunca la continuidad de la transmisión.



* **Amazon**: No soporta tantos dispositivos como Netflix, pero tampoco es que sea fundamental para cubrir su sector. Migró hace tres años a la arquitectura de microservicios siendo una de las primeras grandes compañías que la implementaban en producción. No hay cifra aproximada de la cantidad de solicitudes que pueden recibir a diario, pero no son pocas. Entre éstas encontramos multitud de aplicaciones, las API del servicio web que ofrecen o la propia web de Amazon, cuyos ingenieros reconocen que habría sido imposible sobre la arquitectura monolítica con la que trabajaban previamente.



* **Ebay**: Cómo no, una de las empresas con mayor visión de futuro, siendo pionera en la adopción de tecnologías como Docker o ésta que nos ocupa. Su aplicación principal comprende varios servicios autónomos, y cada uno ejecutará la lógica propia de cada área funcional que se ofrece a los clientes.

# 3. Análisis

La aplicación permitirá a los clientes gestionar su área de cliente. Como hemos comentado anteriormente, el backend encargado de gestionar los datos externos de la aplicación utilizará una arquitectura de microservicios. Por lo que cada módulo del aplicativo será gestionado por un microservicio específico.

El primer paso para acceder a la aplicación será mediante el módulo de Login. Como requisito específico, los usuarios no podrán registrarse ya que será la compañía la encargada de dar de alta a los clientes en sus sistemas. Para el módulo de login y todo lo relativo a los accesos, dispondremos de un microservicio encargado de ello llamado **ms-authentication**.

Una vez logueados y dentro de la aplicación, accederemos directamente al módulo de "Mis líneas" donde podremos recuperar la información de los distintos contratos y líneas que tiene el cliente contratados. Además, mostrará información adicional como la tarifa contratada, consumo de datos o una pequeña vista al componente de facturas. Para recuperar los contratos del cliente se invocará al microservicio **ms-contract**.

La segunda opción de menú será el módulo de "Consumos", en el cual el cliente podrá visualizar mediante diferentes dashboard los consumos de llamadas nacionales, llamadas internacionales, SMS, consumo de datos móviles o el consumo de sus bonos contratados de sus diferentes líneas. Para obtener los consumos del cliente dispondremos de un microservicio llamado **ms-consumption**.

La tercera opción de menú será el módulo de "Servicios" donde el cliente podrá visualizar, activar o desactivar servicios como podrían ser "SMS" o "Llamadas internacionales", entre otros. El API encargado de gestionar los servicios del cliente es el microservicio **ms-service**.

La cuarta opción de menú será el módulo de "Facturas" donde el cliente podrá visualizar o descargar sus últimas 6 facturas referentes al año actual. El API encargado de gestionar dicho módulo será el **ms-invoice**.

La última opción desarrollada será la de "Ajustes" donde el cliente podrá modificar sus datos personales, con alguna excepción como su documento de identidad. El microservicio encargado de trabajar con estos datos será el **ms-settings**.

# 4. Diseño

Tras la fase de análisis, y los requerimientos planteados, se lleva a cabo la fase de diseño técnico del aplicativo. En esta fase se pretende buscar soluciones tecnológicas que den una solución adecuada al problema expuesto.

## 4.1. Angular (Frontend)

El desarrollo de la parte visual de la aplicación está desarrollado con el Framework Angular en su versión 8. Pero, ¿Qué ofrece Angular y por qué he decidido utilizarlo?

* **Angular:** El Framework de por sí solo ofrece muchas más opciones y funcionalidades de serie que una simple biblioteca. Con otro software similar, lo más común es tener que hacer uso de varias bibliotecas de terceros a la hora de desarrollar una aplicación. Lo más probable es que necesites algunas adicionales para hacer el routing, para la gestión de dependencias, para realizar llamadas a APIs REST, etc... También hay muchas decisiones que tomar sobre cómo organizar el código, la arquitectura de la aplicación... Con Angular, todo esto ya lo tienes solucionado.
* **TypeScript:** Aunque se puede programar en ECMAScript "puro", el equipo de Angular decidió que haría todo el desarrollo con el lenguaje TypeScript, y casi toda la documentación y los ejemplos que encuentras por ahí utilizan este lenguaje lo cual ofrece muchas ventajas.

Una de las primeras es la consistencia en la documentación. Si navegas por la Web intentando encontrar ejemplos y tutoriales de otras bibliotecas de JavaScript vas a ver de todo, pero la única constante es la inconsistencia que existe. Por ejemplo, ES6 (o sea, ECMAScript 2015) ofrece varias formas diferentes de declarar un objeto, lo cual puede confundir a muchos.

Con TypeScript esto no pasa, y toda la sintaxis y la manera de hacer las cosas en el código es la misma, lo que añade coherencia a la información y a la forma de leer el código.

Aunque Angular no te obliga a usar TypeScript, el equipo del core de Angular sí que lo ha adoptado y en la documentación sugiere usar TypeScript por defecto. Esto implica que los ejemplos relacionados y los proyectos de código abierto parezcan más familiares y consistentes. Angular ya ofrece ejemplos claros que enseñan cómo usar el compilador TypeScript

Esta consistencia debería ayudar a evitar la confusión y la sobrecarga en la toma de decisiones derivadas de empezar con Angular.

* **Componentes Web:** Un componente en Angular es una porción de código que es posible reutilizar en otros proyectos de Angular sin apenas esfuerzo, lo que permite un desarrollo de aplicaciones mucho más ágil, pasando de un "costoso" MVC a un juego de puzles con nuestros componentes.

El diseño de Angular adopta el estándar de los componentes web. Se trata de un conjunto de APIs que te permiten crear nuevas etiquetas HTML personalizadas, reutilizables y auto-contenidas, que luego puedes utilizar en otras páginas y aplicaciones web. Estos componentes personalizados funcionarán en navegadores modernos y con cualquier biblioteca o Framework de JavaScript que funcione con HTML.

Los componentes que creas en Angular son fáciles de convertir en componentes web nativos. A largo plazo esto es una gran ventaja pues te permitirá reutilizar componentes que crees en Angular en otro tipo de aplicaciones.

## 4.2. Microservicios (Orquestación)

A continuación, se describirán los módulos de los que se compone la solución de Spring Cloud Netflix (Netflix OSS) que hemos implantado para desarrollar nuestra arquitectura de microservicios. Hay que tener en cuenta que el proyecto está muy vivo y en evolución constante.

* **Zuul:** Actúa como gateway sobre nuestros microservicios. Será la puerta de entrada de nuestro ecosistema de microservicios, disponiendo de capacidad para añadir filtros y seguridad.
* **Eureka:** Encargado del autodescubrimiento con el patrón vía cliente, los microservicios se registran con su nombre en el servidor de Eureka, donde dejan su IP y su puerto. Si cualquier microservicio necesita comunicarse, solo debe saber su nombre, para ello le preguntará Eureka su localización y le devolverá todas las IPs y puertos. Será el propio microservicio el encargado del balanceo de peticiones.
* **Spring Cloud Config:** Encargado de la centralización de la configuración, está pensado para emplearse en varios entornos. También proporciona seguridad a la configuración.
* **Ribbon:** Su funcionalidad es la del balanceo de carga entre llamadas de los microservicios. Se integra totalmente con Eureka.
* **Feign:** Nos permite hacer clientes REST fácilmente de una declarativa. Feign crea un balanceador de Ribbon para integrarse con todo el ecosistema.
* **Hystrix:** Implementa el patrón circuit breaker, con el que se controla el error de los microservicios mejorando la resilencia de los mismos. El patrón circuit breaker usa semáforos y métricas para indicar si el servicio funciona correctamente. Además, ofrece un dashboard (Turbine) donde se agregan las métricas que usa Hystrix.
* **Turbine:** Hystrix ofrece una interesante funcionalidad denominada Hystrix Stream que proporciona métricas en tiempo real del estado de los circuit breakers (Hystrix commands) de una aplicación. Para explotar esta información de forma gráfica, Netflix proporciona una interfaz llamada Hystrix Dashboard y un agregador de métricas conocido como Turbine.

## 4.3. Microservicios (APIs)

## 4.4. Flujos de trabajo

# 5. Resultados

# 6. Conclusiones

# 7. Bibliografía