2020

IES San Vicente

Francisco José Ferrer Rodríguez

[I.E.S. SAN VICENTE - GESTLINE]

GestLine es una aplicación web que simula un área de clientes de una compañía de telecomunicaciones.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

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 redesplegar 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 despliegue 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, para la parte backend se ha decidido implantar un sistema distribuido de microservicios basándonos en la arquitectura de 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 se ha 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, 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.

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

Para obtener todos los datos mencionados anteriormente dispondremos del microservicio **ms-contract**.

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-clientmanagement**.

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)

- **Angular:** es un framework opensource desarrollado por Google para facilitar la creación y programación de aplicaciones web de una sola página, las webs SPA (Single Page Application).

  Angular separa completamente el frontend y el backend en la aplicación, evita escribir código repetitivo y mantiene todo más ordenado gracias a su patrón MVC (Modelo-Vista-Controlador) asegurando los desarrollos con rapidez, a la vez que posibilita modificaciones y actualizaciones.

  En una web SPA, aunque la velocidad de carga puede resultar un poco lenta la primera vez que se abre, navegar después es totalmente instantáneo, ya que se ha cargado toda la página de golpe.

  Solamente es una ruta la que se tiene que enviar el servidor, y Angular lo que hace "*por debajo*" es cambiar la vista al navegar para que dé la apariencia de una web normal, pero de forma más dinámica.

  Entre otras ventajas, este framework es modular y escalable adaptándose a nuestras necesidades y al estar basado en el estándar de componentes web, y con un conjunto de interfaz de programación de aplicaciones (API) permite crear nuevas etiquetas HTML personalizadas que pueden reutilizarse.

- **TypeScript:** es el lenguaje principal de programación de Angular, y así toda la sintaxis y el modo de hacer las cosas en el código es el mismo, lo que añade coherencia y consistencia a la información, permitiendo, por ejemplo, la incorporación de nuevos programadores, en caso de ser necesarios, ya que pueden continuar su trabajo sin excesiva dificultad.

  Como ya se ha indicado, las plantillas de Angular almacenan por separado el código de la interfaz del usuario (front-end) y el de la lógica de negocio (back-end), que entre otros beneficios permite utilizar mejor otras herramientas anteriormente existentes.

  Y, por si fuera poco, los principales editores y entornos de desarrollo integrado (IDEs) ofrecen ya extensiones para poder trabajar con este framework con mayor comodidad.

  Por su programación reactiva, la vista se actualiza automáticamente tras realizar los cambios. Además, Angular dispone de asistente por línea de comandos para poder crear proyectos base y también se integra bien con herramientas de testing y con Ionic, lo que facilita la creación de web-responsive, es decir, adaptadas a móviles.

  Este aspecto cada día adquiere mayor importancia tanto por el creciente uso de estos dispositivos para acceder a internet como por la penalización que Google realiza de aquellas páginas que no facilitan su visita en cualquier dispositivo.

4.2. Microservicios (Backend)

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.

- **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.

A continuación, se muestra una ilustración de la arquitectura Spring Cloud Netflix que compone nuestra aplicación GestLine:

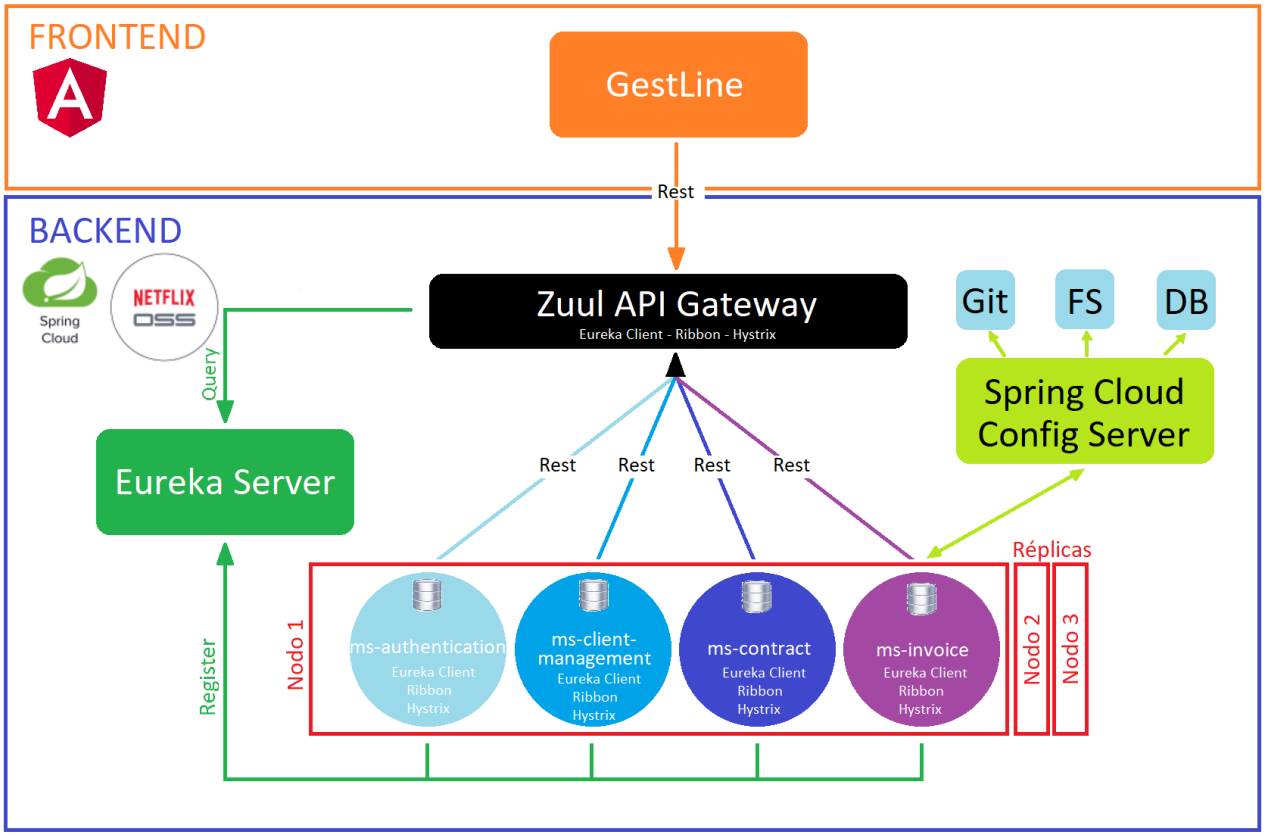

Ilustración 1 - Diseño - Microservicios - Arquitectura de la aplicación

4.3. APIs

A continuación, listaremos las APIs (endpoints) de los distintos microservicios que componen nuestro aplicativo.

API Microservicio: ms-authentication

Descripción	Devuelve todos los usuarios registrados
API	/v1/all
Método	GET
Variable de ruta	-
Cuerpo petición	-

Tabla 1 - Diseño - APIs - Devuelve todos los usuarios registrados

Descripción	Devuelve un usuario por documento
API	/v1/document/{document}
Método	GET
Variable de ruta	document
Cuerpo petición	-

Tabla 2 - Diseño - APIs - Devuelve un usuario por documento

Descripción	Valida el acceso del usuario al aplicativo
API	/v1/checkLogin
Método	POST
Variable de ruta	-

Cuerpo petición	{
	User: { user }
	}

Tabla 3 - Diseño - APIs - Valida el acceso del usuario al aplicativo

Descripción	Actualiza un usuario
API	/v1/updateUser
Método	POST
Variable de ruta	-

Cuerpo petición	{
	User: { user }
	}

Tabla 4 - Diseño - APIs - Actualiza un usuario

Descripción	Devuelve el token de validación de un usuario por documento
API	/v1/getToken/{document}
Método	GET
Variable de ruta	document
Cuerpo petición	-

Tabla 5 - Diseño - APIs - Devuelve el token de validación de un usuario

API Microservicio: ms-client-management

Descripción	Devuelve todos los clientes registrados
API	/v1/all
Método	GET
Variable de ruta	-
Cuerpo petición	-

Tabla 6 - Diseño - APIs - Devuelve todos los clientes registrados

Descripción	Devuelve un cliente por documento
API	/v1/document/{document}
Método	GET
Variable de ruta	document
Cuerpo petición	-

Tabla 7 - Diseño - APIs - Devuelve un cliente por documento

Descripción	Actualiza un cliente
API	/v1/updateClient
Método	POST
Variable de ruta	-

Cuerpo petición	{
	Client: { client }
	}

Tabla 8 - Diseño - APIs - Actualiza un cliente

API Microservicio: ms-contract

Descripción	Devuelve todos los contratos del cliente
API	/v1/all
Método	GET
Variable de ruta	-
Cuerpo petición	-

Tabla 9 - Diseño - APIs - Devuelve todos los contratos del cliente

Descripción	Devuelve un contrato del cliente por documento
API	/v1/document/{document}
Método	GET
Variable de ruta	document
Cuerpo petición	-

Tabla 10 - Diseño - APIs - Devuelve un contrato del cliente por documento

Descripción	Actualiza un contrato del cliente
API	/v1/updateContractsService
Método	POST
Variable de ruta	-

Cuerpo petición	{
	[String: { contractsService }]
	}

Tabla 11 - Diseño - APIs - Actualiza un contrato del cliente

API Microservicio: ms-invoice

Descripción	Devuelve todas las facturas del cliente
API	/v1/all
Método	GET
Variable de ruta	-
Cuerpo petición	-

Tabla 12 - Diseño - APIs - Devuelve todas las facturas del cliente

Descripción	Devuelve una factura por id de factura
API	/v1/id/{id}
Método	GET
Variable de ruta	id
Cuerpo petición	-

Tabla 13 - Diseño - APIs - Devuelve una factura por id de factura

Descripción	Devuelve las facturas filtrando por documento, fecha inicio y fecha fin
API	/v1/betweenDates/{document}/{startDate}/{endDate}
Método	GET
Variable de ruta	document, startDate, endDate
Cuerpo petición	-

Tabla 14 - Diseño - APIs - Devuelve las facturas filtrando por documento

Descripción	Devuelve últimas facturas filtrando por documento y número de facturas
API	/v1/lastInvoices/{document}/{numInvoices}
Método	GET
Variable de ruta	document, numInvoices
Cuerpo petición	-

Tabla 15 - Diseño - APIs - Devuelve las últimas facturas filtrando por documento

Descripción	Visualiza una factura en el navegador en formato PDF
API	/v1/showInvoice/{fileName}
Método	GET
Variable de ruta	fileName
Cuerpo petición	-

Tabla 16 - Diseño - APIs - Visualiza una factura en el navegador en formato PDF

Descripción	Descarga una factura en formato PDF
API	/v1/downloadInvoice/{fileName}
Método	GET
Variable de ruta	fileName
Cuerpo petición	-

Tabla 17 - Diseño - APIs - Descarga una factura en formato PDF

4.4. Bases de datos

Para la solución en fase de desarrollo se ha decidido utilizar H2. H2 es una base de datos ligera de código abierto desarrollada en Java. Puede integrarse en aplicaciones Java o ejecutarse en modo cliente-servidor. La base de datos H2 se puede configurar para ejecutarse como base de datos en memoria, lo que significa que los datos no persistirán en el disco.

Características más destacadas de esta herramienta:

- **Alta integración:** Debido a que como ya se ha dicho esta implementada en Java su integración con cualquier aplicación en este lenguaje es total (mediante API JDBC o ODBC).
- **Uso en diferentes plataformas:** Debido a que es Java se puede utilizar en cualquier plataforma.
- **Rápida:** Obtiene su gran velocidad gracias a su estrategia de optimización basada en costes, por lo que en muchos casos la hace destacar sobre otras bases de datos más conocidas.
- **Tamaño reducido:** Ocupa muchísimo menos que muchas de las bases de datos que se han nombrado anteriormente (el JAR ocupa aproximadamente 1MB).
- **Modo embebido:** Permite el funcionamiento en este modo realizando la gestión de los datos en archivos haciendo uso de una pequeña parte de memoria.
- **Modo «en memoria»:** Permite el funcionamiento en este modo realizando la gestión de los datos directamente sobre la memoria, lo que acelera enormemente las operaciones realizadas.
- **Además** de las anteriores características esta base de datos destaca en otros aspectos como se puede observar en la siguiente tabla comparativa.

A continuación, listaremos las bases de los distintos microservicios.

BBDD Microservicio: ms-authentication

USERS	
Columna	Tipo
document	VARCHAR
email	VARCHAR
password	VARCHAR

Tabla 18 - Diseño - BBDD - Tabla USERS

BBDD Microservicio: ms-client-management

ADDRESS	
Columna	Tipo
id	BIGINT
type	VARCHAR
direction	VARCHAR
number	VARCHAR
stairs	VARCHAR
floor	VARCHAR
location	VARCHAR
province	VARCHAR
postal_code	VARCHAR
country	VARCHAR

Tabla 19 - Diseño - BBDD - Tabla ADDRESS

BILLING	
Columna	Tipo
id	BIGINT
entity	VARCHAR
office	VARCHAR
dc	VARCHAR
account	VARCHAR
address (fk)	BIGINT

Tabla 20 - Diseño - BBDD - Tabla BILLING

CLIENT	
Columna	Tipo
document	VARCHAR
document_type	VARCHAR
client_type	VARCHAR
name	VARCHAR
first_surname	VARCHAR
second_surname	VARCHAR
birth_date	VARCHAR
email	DATE
online_invoice	BOOLEAN
due	INTEGER
limit_due	INTEGER
blacklist	BOOLEAN
address (fk)	BIGINT
billing (fk)	BIGINT

Tabla 21 - Diseño - BBDD - Tabla CLIENT

BBDD Microservicio: ms-contract

LINECONSUMPTION	
Columna	Tipo
id	BIGINT
total_minuts	DOUBLE
total_minuts_international	DOUBLE
usage_minuts	DOUBLE
usage_minuts_international	DOUBLE
total_sms	INTEGER
usage_sms	INTEGER

Tabla 22 - Diseño - BBDD - Tabla LINECONSUMPTION

DATACONSUMPTION	
Columna	Tipo
id	BIGINT
total_bytes	DOUBLE
total_bytes_international	DOUBLE
total_bytes_bonus	DOUBLE
usage_bytes	DOUBLE
usage_bytes_international	DOUBLE
usage_bytes_bonus	DOUBLE

Tabla 23 - Diseño - BBDD - Tabla DATACONSUMPTION

CONTRACTLINE	
Columna	Tipo
phone	VARCHAR
contract_id	VARCHAR
rate	VARCHAR
contract_type	VARCHAR
iccid	VARCHAR
pin	VARCHAR
puk	VARCHAR
tecnology	VARCHAR
partner_points	INTEGER

Tabla 24 - Diseño - BBDD - Tabla CONTRACTLINE

CONTRACTSERVICES	
Columna	Tipo
id	BIGINT
contract_line_id	VARCHAR
name	VARCHAR
description	VARCHAR
active	BOOLEAN

Tabla 25 - Diseño - BBDD - Tabla CONTRACTSERVICES

BBDD Microservicio: ms-invoice

INVOICEDOCUMENT	
Columna	Tipo
id	BIGINT
document	VARCHAR
invoice_date	DATE

Tabla 26 - Diseño - BBDD - Tabla INVOICEDOCUMENT

INVOICE	
Columna	Tipo
id	BIGINT
invoice_document (fk)	BIGINT
phone	VARCHAR
amount	DOUBLE
tax	DOUBLE
tax_amount	DOUBLE
total_amount	DOUBLE

Tabla 27 - Diseño - BBDD - Tabla INVOICE

4.5. Diagramas de flujo

Los siguientes esquemas muestran la navegabilidad entre pantallas, así como el flujo de datos entre ellas.

En primer lugar, tenemos el flujo que siguen los usuarios al acceder a la aplicación, donde lo primero que se comprobará es si el usuario está logueado o no. En caso de estar logueado se cargará la pantalla principal, en caso contrario, se le permitirá iniciar sesión.

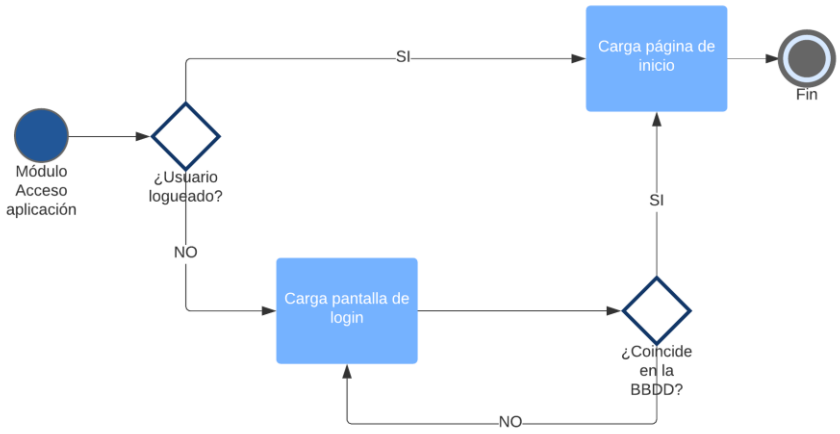

Ilustración 2 - Diseño - Diagramas de flujo - Módulo de acceso a la aplicación

Una vez el usuario está logueado y se encuentra dentro de la aplicación, se cargará la pantalla principal del módulo *Mis líneas*. Una vez en dicho módulo, en la parte izquierda se les mostrará información relativa a sus líneas con sus respectivos consumos y en la parte derecha se les mostrará un histograma con sus 6 últimas facturas anuales.

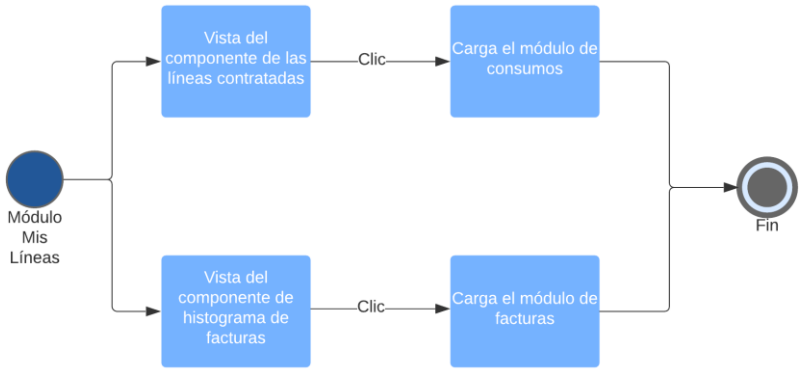

Ilustración 3 - Diseño - Diagramas de flujo - Módulo de Mis líneas

En el módulo de Consumos podremos visualizar información relativa a los consumos por línea del cliente. Como podrían ser el consumo de minutos de llamadas nacionales e internacionales, total de SMS disponibles y enviados, consumo de datos (nacional e internacional), consumo de bonos de datos, etc.

Ilustración 4 - Diseño - Diagramas de flujo - Módulo de Consumos

En el módulo de Servicios podremos visualizar información relativa a los servicios que tiene el cliente contratados como pueden ser las llamadas internacionales, la navegación en internet, el Roaming, los SMS, etc.

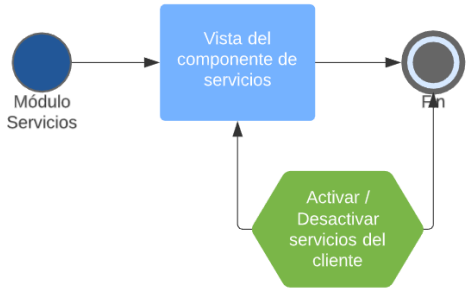

Ilustración 5 - Diseño - Diagramas de flujo - Módulo de Servicios

Si accedemos al módulo de Facturas, en la parte superior podremos visualizar un histograma con las últimas 6 facturas (6 meses) del cliente. En dicho histograma podremos hacer clic en el mes deseado que en la parte inferior se pueda cargar un resumen de dicha factura. Permitirá visualizar o descargar la factura deseada.

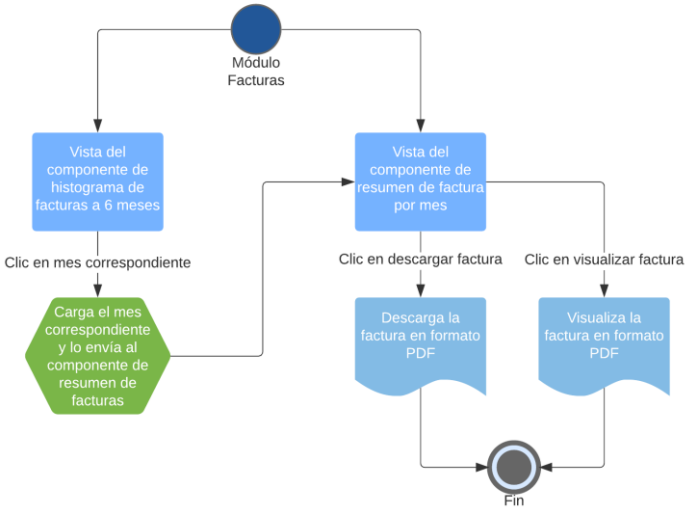

Ilustración 6 - Diseño - Diagramas de flujo - Módulo de Facturas

Si accedemos al módulo de Ajustes, el usuario podrá visualizar toda la información relativa a sus datos personales. Así como modificar todos estos datos a excepción del segmento y el documento de identidad.

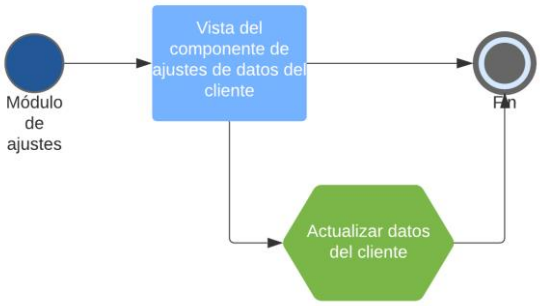

Ilustración 7 - Diseño - Diagramas de flujo - Módulo de Ajustes

5. Resultados

Una vez se ha realizado toda la implementación de la aplicación, se debe evaluar el proceso y los resultados obtenidos.

En cuanto a las funcionalidades definidas y deseadas inicialmente en la fase de diseño se ha conseguido implementar todo lo especificado, aunque se podría ampliar con funcionalidades extra o mejoras en cuanto a algunos componentes, por ejemplo, contratar nuevas líneas, cambios de tarifa, catálogo de tarifas, renoves, portabilidades, catálogo de terminales, etc.

Se ha intentado realizar una interfaz minimalista, intuitiva y de fácil uso para el cliente final, imitando en algunos casos y funcionalidad de aplicaciones conocidas como el área de clientes de Orange.

5.1. Pantallas de la aplicación

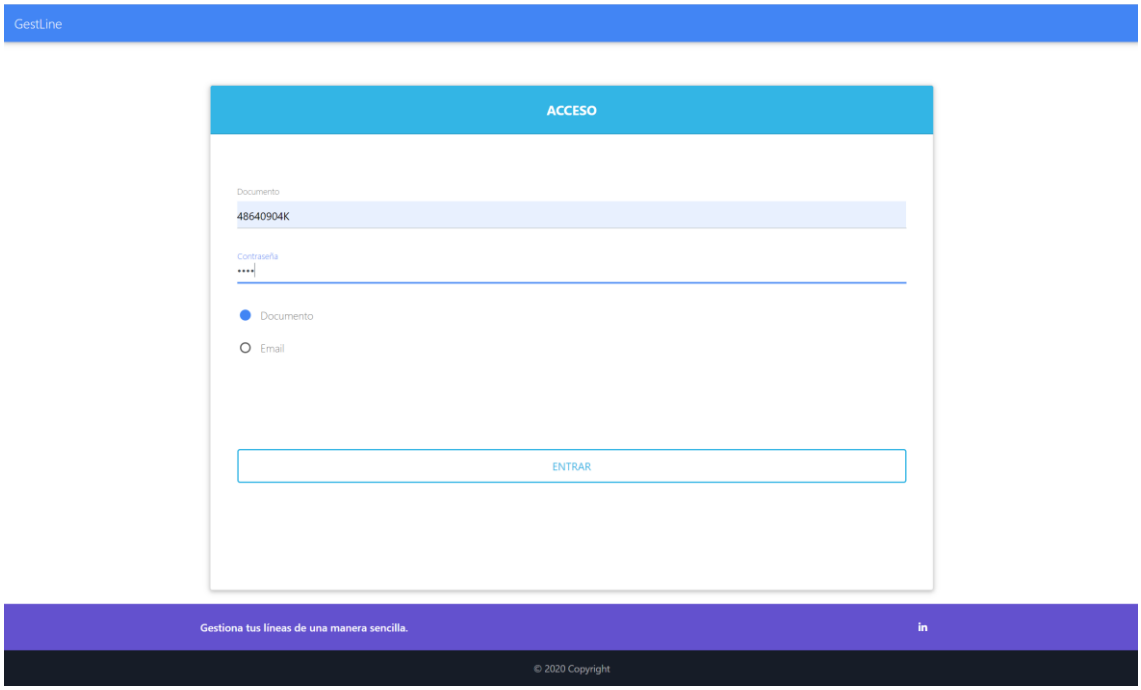

Ilustración 8 - Resultados - Pantallas de la aplicación - Módulo de Login

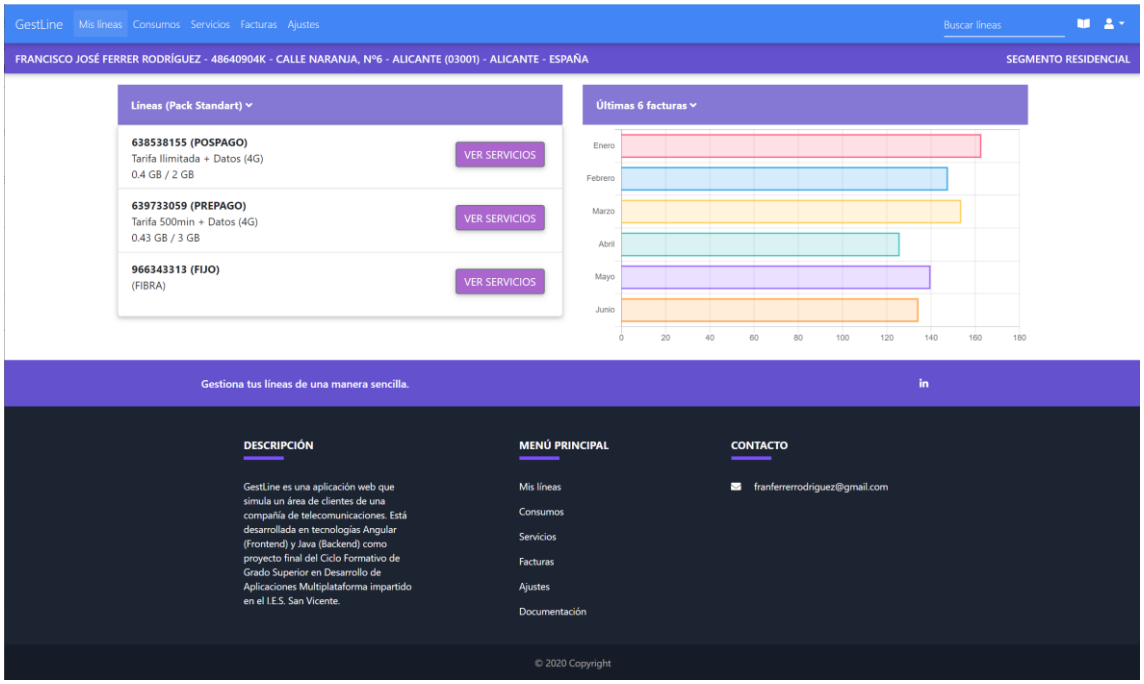

Ilustración 9 - Resultados - Pantallas de la aplicación - Módulo de Mis Líneas

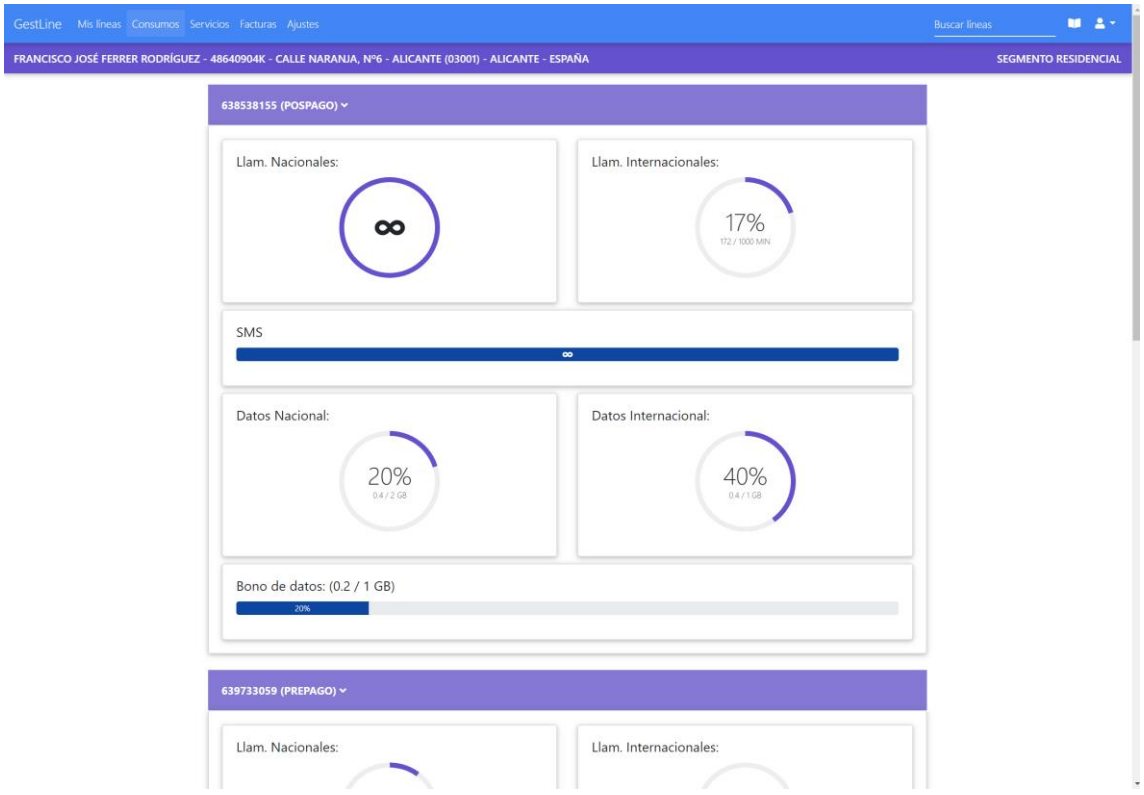

Ilustración 10 - Resultados - Pantallas de la aplicación - Módulo de Consumos

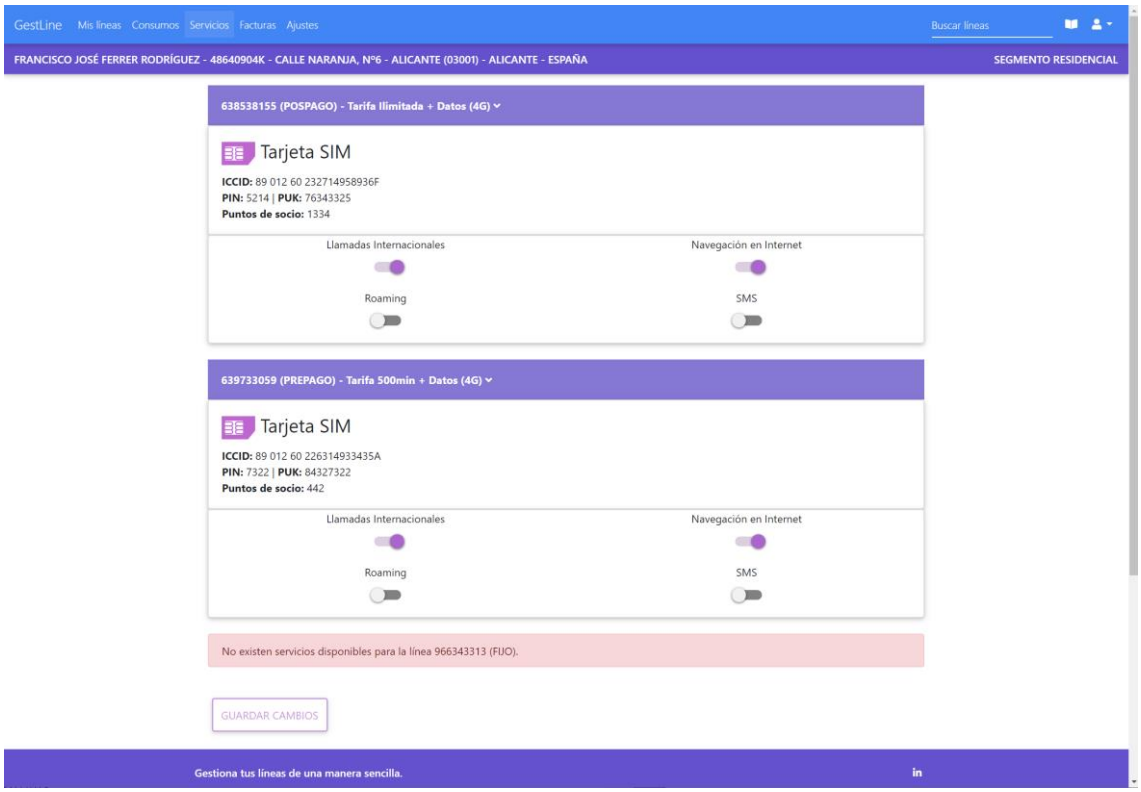

Ilustración 11 - Resultados - Pantallas de la aplicación - Módulo de Servicios

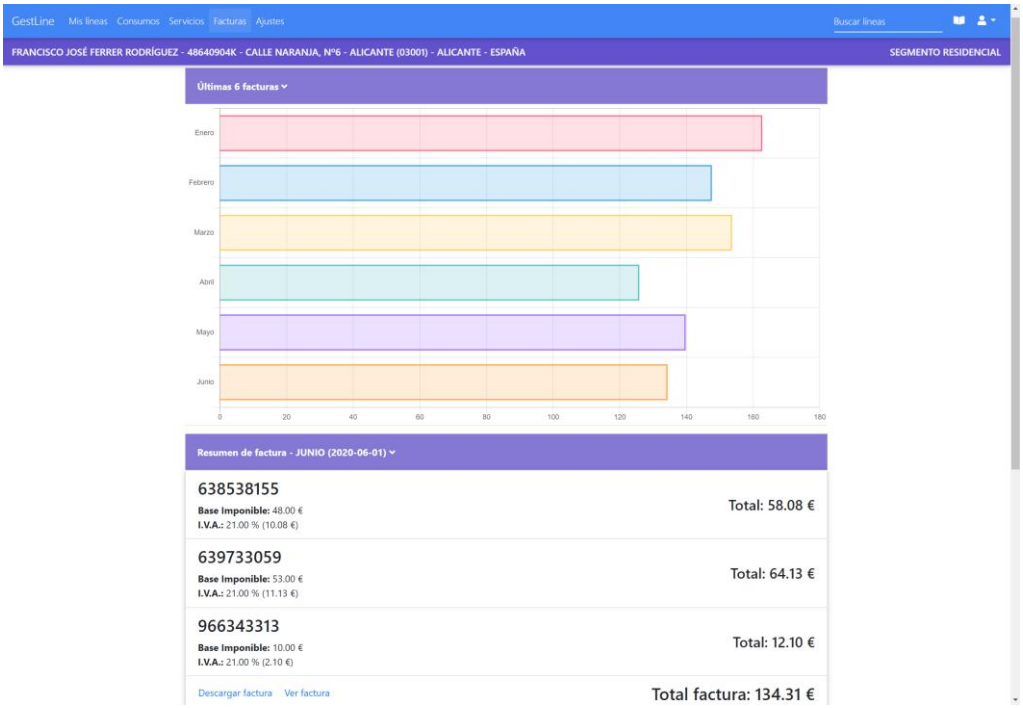

Ilustración 12 - Resultados - Pantallas de la aplicación - Módulo de Facturas

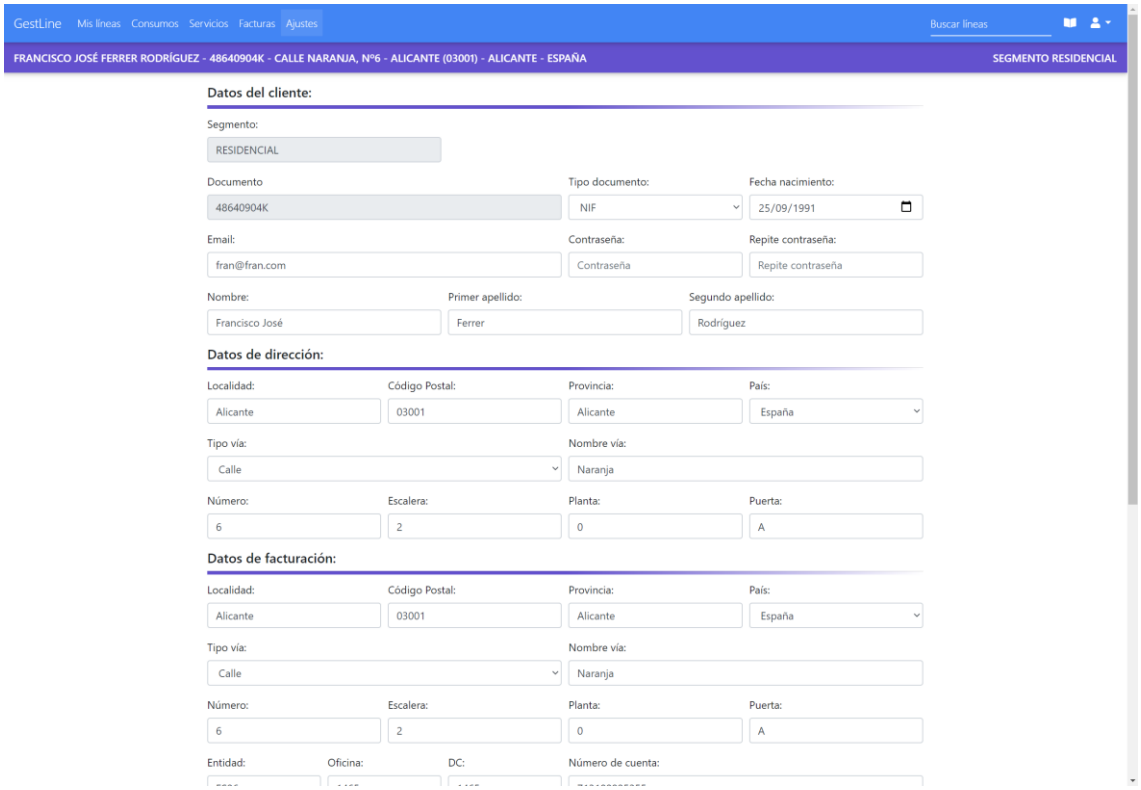

Ilustración 13 - Resultados - Pantallas de la aplicación - Módulo de Ajustes

5.2. Variables de sesión en navegador

Cuando un usuario se loguea en la aplicación se produce un proceso de validación donde se comprueba que el usuario logueado es realmente el usuario autenticado por el sistema.

En primer lugar, se guardan una serie de datos en las variables de sesión del navegador:

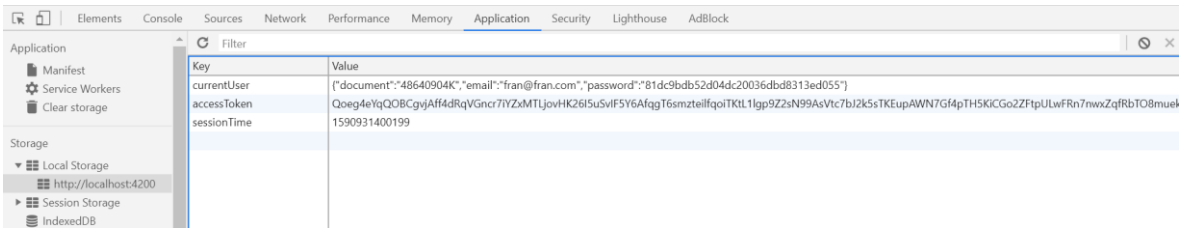

Ilustración 14 - Resultados - Autenticación de la sesión - Variables de sesión en navegador

- **currentUser:** guarda información relativa al cliente que tiene la sesión actual (documento, email y password encriptada).
- **accessToken:** cuando un usuario se loguea en el sistema, el microservicio ms-authentication genera un token único y temporal que es guardado tanto en las variables de sesión del navegador como en la base de datos del microservicio. Ambos token deben coincidir mientras dure la sesión del usuario, si alguno de los dos cambia (por manipulación de datos) se cerrará la sesión.
- **sessionTime:** guarda el tiempo (milisegundos) en el que se ha iniciado la sesión. Para posteriormente compararlo con el tiempo actual (también en milisegundos). De esta forma podemos controlar cuando cerrar sesión por inactividad.

5.3. Orquestación de microservicios (Eureka)

Eureka es una librería que forma parte del stack de Spring Cloud, desarrollada por Netflix y es el encargado del autodescubrimiento de los microservicios.

Eureka también nos ofrece un Dashboard que nos ayuda a tener una visión de qué instancias están activas y cuales han podido caer (poniendo en peligro el sistema), entre otra información. Aunque es modificable, Eureka utiliza por defecto el puerto 8761.

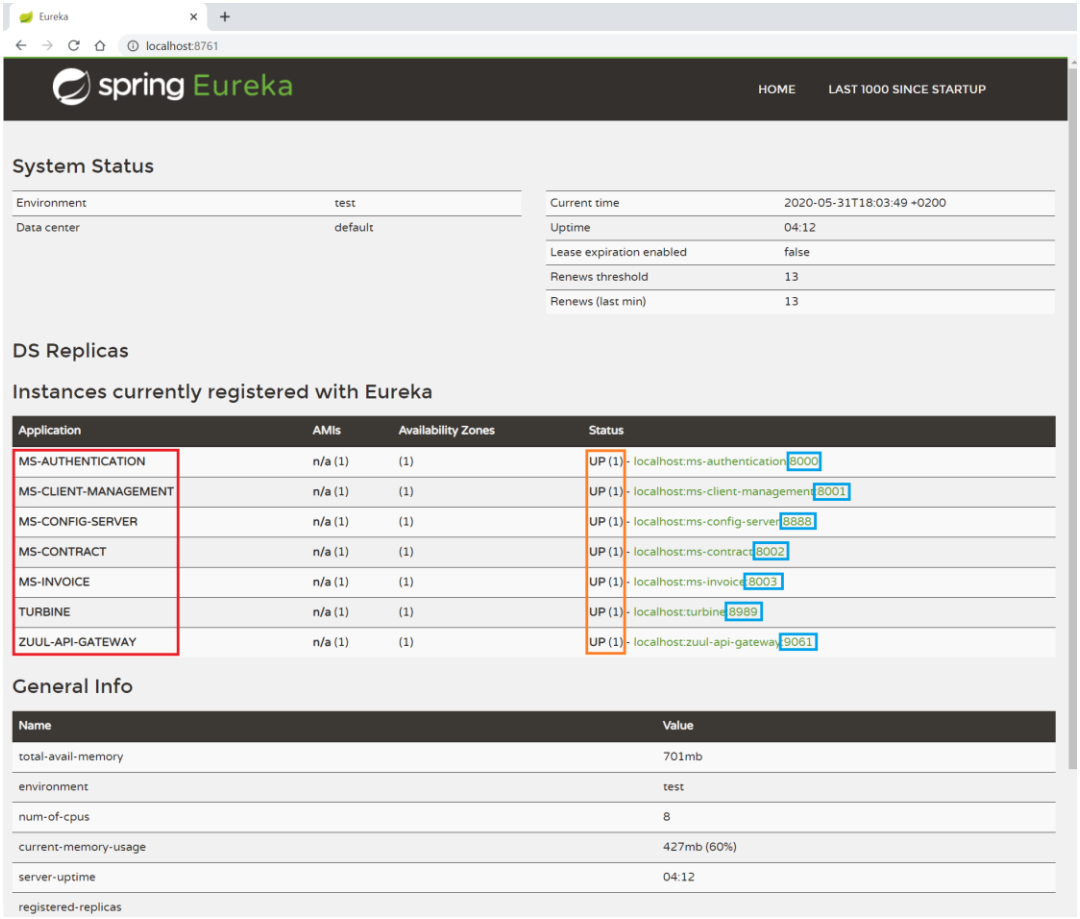

Ilustración 15 - Resultados - Orquestación de microservicios - Eureka

- Instancias de microservicios activas: si nos fijamos en el apartado *Application*, cuando un microservicio es levantado, Eureka lo registra automáticamente como por arte de magia.

- Número de réplicas: en el apartado *Status* podemos tener información de cuántas instancias de microservicios hay levantados en cada nodo. Por ejemplo, el microservicio ms-authentication podría estar levantado en 10 nodos diferentes (incluso en diferentes ubicaciones), de tal modo que, si uno de ellos colapsa, Eureka dejará de registrarlo y quedarían 9 instancias restantes que seguirían dando servicio, por lo que se genera una alta disponibilidad.

- Puerto: Cada microservicio tiene un puerto definido en su fichero de configuración, independientemente de cuántos nodos repliquen instancias de dicho microservicio, siempre tendrá el mismo puerto.

5.4. Orquestación de microservicios (Hystrix y Turbine)

Hystrix es una librería que forma parte del stack de Spring Cloud, desarrollada por Netflix, que facilita la implementación del patrón circuit breaker dentro de una arquitectura de servicios distribuidos.

La comunicación entre sistemas adolece de indisponibilidades debidas a las propias características de los mismos: microcortes en las comunicaciones, servicio no disponible temporalmente, lentitud en las respuestas por el excesivo uso de un servicio. El patrón circuit breaker permite gestionar estos problemas derivados de las comunicaciones estableciendo mecanismos de control, ayudando a mejorar la resiliencia de los sistemas.

Hystrix Dashboard es una consola que nos ofrece Netflix para explotar las métricas en tiempo real del estado de los circuit breakers de las aplicaciones, procesando los hystrix stream que éstas generan y representado los resultados de manera gráfica en un cuadro de mandos.

Por cada petición que se realiza al microservicio, se genera un hystrix stream en formato JSON que provee de información sobre la transacción en cuestión.

Ilustración 16 - Resultados - Orquestación de microservicios - hystrix stream

Para poder "clusterizar" todas las monitorizaciones Hystrix en un único punto de monitorización podemos usar Turbine. Turbine muestra en un mismo Dashboard toda la información proveniente de los hystrix stream de los distintos microservicios que tiene Eureka registrados. Proveyendo de información visual el estado del circuito y las peticiones que maneja en cada API.

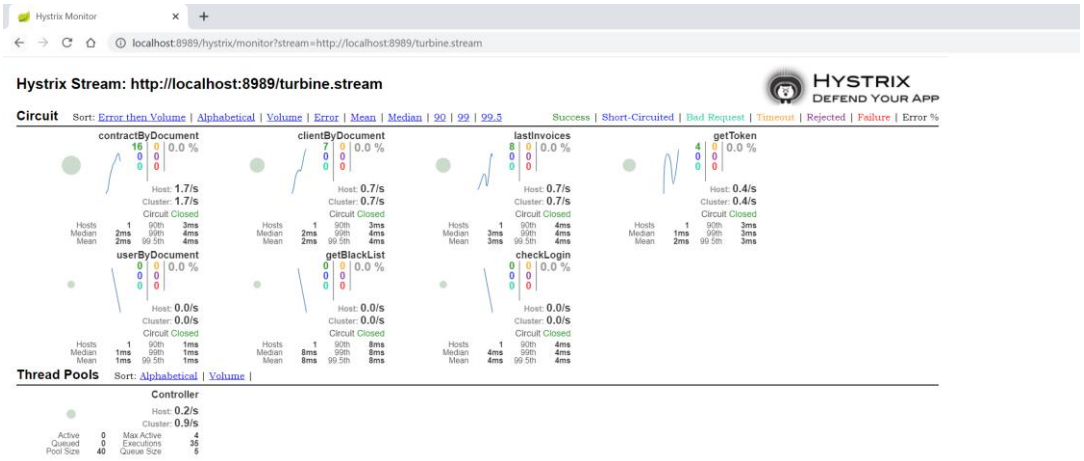

Ilustración 17 - Resultados - Orquestación de microservicios - Turbine

6. Conclusiones

A continuación, voy a detallar cómo ha ido la evolución del proyecto, aprendizaje de tecnologías, así como problemas encontrados durante su desarrollo.

6.1. Detalles de la solución

Se puede decir que he disfrutado mucho con el desarrollo de este proyecto de final de ciclo, desde su fase de análisis hasta su fase de desarrollo y resultado final. Además de reforzar los conocimientos adquiridos durante todo el curso, considero que me ha venido bien aprender y poner en práctica otras tecnologías como el framework Angular o el lenguaje de programación Java con la arquitectura de microservicios, muy utilizado en el ámbito laboral.

6.2. Problemas encontrados

Desde la fase de análisis se podría decir que tenía las ideas bastante claras con respecto a lo que quería desarrollar y cómo quería el resultado final. No obstante, he encontrado dificultades a la hora de desarrollar la parte del frontend en el framework Angular debido que no tenía unas nociones bien definidas y he tenido que estudiarlo para poder sacar adelante algunas partes del proyecto.

También he encontrado dificultades a la hora de implantar la solución de "Spring Cloud Netflix" para la parte del backend, debido a su complejo Stack y funcionalidad (Eureka, Zuul, Hystrix, microservicios, etc.). Pero finalmente, una vez que todas las piezas encajan, ha sido gratificante ver el resultado final en conjunto.

6.3. Evolución del proyecto

Podríamos seguir evolucionando los módulos existentes de la aplicación, además de diseñar nuevos módulos para realizar nuevos flujos de "*Cambios de tarifa*", "*Renoves*", "*Portabilidades*", "Tienda online", etc. También sería importante, aprovechando el framework Angular, podríamos desarrollar el aplicativo en dispositivos móviles utilizando Ionic.

7. Bibliografía

- Angular - https://angular.io
- Ventajas Angular - https://www.campusmvp.es/recursos/post/las-5-principales-ventajas-de-usar-angular-para-crear-aplicaciones-web.aspx
- Spring.io - https://start.spring.io
- Configuración Spring Cloud - https://www.baeldung.com/spring-cloud-configuration
- Arquitectura Spring Cloud - https://blog.bi-geek.com/microservicios-arquitectura-spring-cloud
- Tolerancia a fallos - https://blog.sarenet.es/tolerancia-fallos-netflix
- Microservicios 1 - http://www.springboottutorial.comcreating-microservices-with-spring-boot-part-1-getting-started
- Microservicios 2 - https://blog.bi-geek.com/arquitecturas-basadas-microservicios
- Zuul 1 - https://javabeginnerstutorial.com/spring-bootspring-boot-2-microservices-with-netflix-zuul-api-gateway
- Zuul 2 - https://blog.bi-geek.com/arquitecturas-basadas-en-microservicios-spring-cloud-netflix-zuul
- Eureka - https://blog.bi-geek.com/arquitecturas-spring-cloud-netflix-eureka
- Ribbon - https://blog.bi-geek.com/arquitecturas-basadas-en-microservicios-spring-cloud-ribbon
- Feign - https://blog.bi-geek.com/arquitecturas-basadas-en-microservicios-spring-cloud-feign