1 Introducción

Luego de la construcción de nuestra aplicación el siguiente paso es de cubrir varios aspectos que son claves hoy en día dentro de la producción de componentes software.
Las características de nuestro software son bastante completas gracias a haber respetado modelos de construcción asentados y con un marco teórico y experiencial por detrás, pero, ¿y la seguridad de la aplicación? Cómo se va a realizar el proceso de autenticación de usuarios o por el cual se van a cubrir las solicitudes a la API. O, ¿cómo podemos garantizar que el usuario es miembro de la Universidad? ¿O cómo garantizamos que a los técnicos se enteren de las solicitudes de préstamos que les llegan?
Dentro de este complemento de trabajo fin de grado nos dedicaremos a la resolución de estas problemáticas que surgen al haber finalizado la implementación de nuestra aplicación.

1.1 Motivaciones

Por respetar en pequeña parte la estructura del trabajo fin de grado he decidido poder realizar una pequeña sección donde hablar de los motivos de la realización de este complemento.
El motivmo principal es que lo tengo que realizar si o sí, de eso no cabe duda. El segundo radica en las posibilidades que nos brinda el software de las que tanto he hablado en el proyecto.
Una herramienta no acaba con el satisfacer los requisitos que se nos pedían en un principio. Creo que este aspecto nos diferencia bastante de las otras ingenierías. Puede ser quizás a que nuestra forma de implementación es más fácil, quizás. Pero la construcción de herramientas normalmente se expande forma continua y uniforme.
Lo podemos ver con la evolución de las redes sociales de hoy en día. Pongamos como ejemplo, Facebook. Quiero recalcar que en esta sección no estoy tratando de la originalidad de las ideas sino de la ejecución de las mismas.
Facebook al principio de los 2000 no era más que una red social cualquiera. En la que la gente podía interactuar, publicar lo que habían hecho durante el día o habían aprendido y más tarde alrededor del 2008 compartir logros dentro de un sistema de juegos integrados que venían con la plataforma. Ya la implantación de los juegos es "algo", pero no fue hasta que comercialiaron aplicaciones como Whatsapp o Instagram donde en verdad tuvo su éxito.
La tecnología da más tecnología. Es una estufa que no se apaga y las implementaciones y añadidos que se pueden realizar a un proyecto son gigantescas. Del modo y forma debidamente adecuados.

1.2 Objetivos

¿Qué se pretende con la realización de este complemento? Nuestros objetivo principal es la mejora de la aplicación relacionada en tres áreas:

- La seguridad.
- La escalabilidad.
- La comunicación.

1.2.1 La seguridad

La seguridad es algo fundamental que se pide hoy en día en internet. Y algo que suele verse relegado a un segundo plano por bastantes empresarios.
El objetivo de este apartado es centrarnos en dos aspectos: la seguridad en las transacciones con la Interfaz de Programación de Aplicaciones (API), esto lo realizaremos mediante una tokenización de las transacciones que gestionaremos al principio de nuestra aplicación. Lo segundo es la implementación de un sistema de copias de seguridad automatizado que nos ayudará a que en caso de pérdida de información podamos recuperar parte de esta gracias a ficheros almacenados de forma externa que contengan esa información. Estas las iremos generando siguiendo algunas normativas de generación de copias de seguridad en entornos empresariales.

1.2.2 La escalabilidad

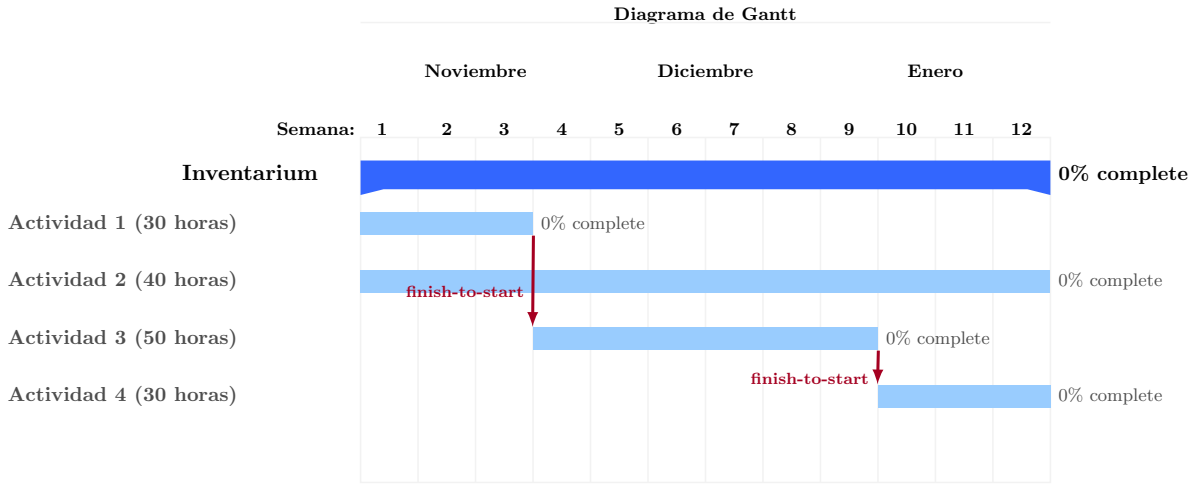
En el proyecto hablé sobre Docker y todas las utilidades que nos podía brindar. En este caso hablaremos de Docker Swarm, un sistema que implementa Docker para poder realizar un balanceo de carga en nuestra aplicación y que esta no se vea sobrecargada. Ideal por si ocurren caídas en algunas de nuestras máquinas o queremos rebajar un poco los recursos que consumen. Explicaré cómo implementaremos Docker Swarm en nuestro entorno y cómo estos tipos de herramientas pueden ser de gran utilidad para aspectos como la escalabilidad.

1.2.3 La comunicación

La interacción que una aplicación tiene con su usuario es un aspecto básico que tiene que ser cubierto.
Esta comunicación hoy en día se puede realizar de diferentes formas, por ejemplo, el pitido que hace un microondas cuando termina su ejecución. Ese ya es un método de comunicación que está teniendo con el usuario.
Al principio del auge de internet y de los sistemas informáticos la comunicación era unidireccional, es decir, iba del usuario a la máquina, y esta ya mostraba lo que el usuario le pedía. Nuestra aplicación funciona al igual que hacían las páginas webs hace 20 años. El usuario si necesita algo o quiere consultar si le han concedido una solicitud tiene que buscarlo.
Este aspecto lo cubriremos gracias a la implementación de un servidor de gestión de correo dentro de nuestra API. También decidiremos los modos y momentos en los que la aplicación se comunicará con el usuario.

1.3 Planificación

La planificación se dividirá en cuatro grandes grupos, muy parecidos a los del proyecto: Reuniones con el cliente, planificación y elaboración del desarrollo del proyecto, construcción de la aplicación y testeo y comprobación de la aplicación y comprobación de errores. Son los mismos puntos que en el proyecto nada más que no trataremos el punto de “Preparación del entorno de trabajo”.



- **Actividad 1** (30 horas) Reuniones con el cliente
  - Reunión para tratar sobre la comunicación a cubrir con el usuario
  - Reunión para tratar sobre la seguridad y la escalabilidad
- **Actividad 2** (40 horas) Planificación y elaboración del proyecto
- **Actividad 3** (50 horas) Construcción de la aplicación
  - Tokenización de las solicitudes a la base de datos
  - Configuración de generado automático de copias de seguridad
  - Implementación del sistema de envío de correos y gestión de la comunicación
  - Implementación de Docker Swarm
- **Actividad 4** (30 horas) Testeo, comprobación de la aplicación y comprobación de errores

La metodología del trabajo a realizar será en cascada. Es decir, iremos cubriendo cada uno de los apartados y hasta no terminar con el superior no podremos pasar al siguiente. Este modelo se realizará para las tres implementaciones que haremos.
Las primeras fases se centrarán en el análisis y planificación de la solución que se pretenda implementar. Las fases intermedias consistirán en esta implantación de la aplicación y las finales se centrarán en la realización de pruebas para la comprobación de que todas las funcionalidades que se pensaban implementar en un momento cumplen a la perfección su función y lo hacen de forma correcta.

2 Herramientas utilizadas

En este capítulo hablaré sobre las herramientas utilizadas durante el desarrollo del proyecto. Algunas de ellas ya las habré presentado en el anterior proyecto pero no serán demasiadas para no volver tan extenso el documento.
En la sección de hardware, eso sí, disponemos de los mismos dispositivos que antes así que no va a haber variación.

2.1 Hardware

Dentro del apartado de hardware disponemos de dos ordenadores. Su procesador no conlleva relevancia en el desarrollo de la aplicación debido a la utilización de servicios en la nube.

2.1.1 Torre de PC

La cual contiene como procesador un Xeon E5-2620 V3. 16GB de RAM DDR3. Una tarjeta gráfica RTX 570 de 4GB DDR5. Tiene 256GB de memoria SSD y 1TB de memoria HDD.

2.1.2 Un portátil

Es un MacBook Air M1 de 2020 con 8GB de RAM y 256GB de almacenamiento.

2.2 Software

2.2.1 Entorno de desarrollo

Nuestro entorno de desarrollo y desde donde haremos casi absolutamente todo será desde Visual Studio Code. Este es un editor de código desarrollado por Microsoft que soporta varias distribuciones de sistemas operativos, entre ellas: Windows, Mac Os y Ubuntu.
Una de las características de esta herramienta que la hacen la predilecta de varios desarrolladores es el gran soporte que tiene por parte de la comunidad. Tiene un mercado de plugins bastante grande que apoya la creación continua de código para todos los desarrolladores.
Dispone de integraciones con Git, resaltado en errores de sintaxis, finalización de código y hasta conexión remota a otros entornos de trabajo mediante SSH.
Otra enorme ventaja que presenta es el consumo de memoria que tiene, bastante pequeño. Es un programa para ordenadores de todos los tamaños y precios, un software gratuito y una herramienta increíblemente potente al alcance de todos.

2.2.2 Redacción del documento

Estas líneas están siendo escritas ahora mismo desde LaTeX. LaTeX es un sistema de composición de textos que está formado mayoritariamente por órdenes construidas a partir de comandos TeX. En un principio no estaba seguro de qué herramienta utilizar, ya que la posición de varios profesores respecto a esta herramienta era bastante férrea pero Word siempre había ido agarrado a mi mano desde comienzos del instituto.
Luego de pasar de Word a LaTeX y de LaTeX a Word bastantes veces no fue hasta que mi profesora Rosa, en una de mis visitas matinales a su despacho me dijo: Yo hice mi TFG en

LaTeX.
No me lo podía creer y al comprobar la fecha de publicación de este programa de procesado de textos me sorprendí al ver que su lanzamiento oficial fue en 1980. "Si el programa ha durado tanto es que algo de importante tendrá" pensé. Y aquí me hallo redactando este documento con un programa que facilita el control de versiones de Git de una manera asombrosa. Facilita también los procesos de documentación y disposición de las diferentes subsecciones. Y, lo que más me gusta sin lugar a dudas, que puedo realizar una separación de cada capítulo por documentos separados y es que a mí, el tener las cosas descompuestas, me puede.

2.2.3 Google Cloud

Volveremos a usar Google Cloud pero esta vez no para el deploy de nuestra página web. Ya que ya lo hemos hecho. Sino que será para la generación de un entorno red donde desplegaremos una o dos máquinas virtuales más. Esto lo haremos con el objetivo de generar una unión entre ellas y poder utilizar una de las herramientas que nos incorpora Docker.

2.2.4 Docker Swarm

Docker Swarm nos permite tener varios contenedores Docker interconectados entre sí en diferentes máquinas virtuales o físicas.
Es decir, Docker Swarm nos permite la gestión de un clúster de servidores Docker. Además nos aporta herramientas para poder gestionar estos como si se tratase de la gestión de un simple contenedor.

3 Implementación de las funcionalidades

3.1 Tokenización

La utilización de tokens durante el desarrollo de APIs ha incrementado a lo largo de los últimos años. Estos tokens permiten aislar de esa capa de seguridad de implementación extra a la que te antes te veías sometido por hacer aplicaciones que gestionasen sus propias consultas con la base de datos.
A medida que iba realizando la construcción de la aplicación pensaba en esta implementación de seguridad.
En un principio lo que hice fue almacenar los datos del usuario y la contraseña de este en caché. Cada vez que se mandaba una consulta se utilizaban los datos en caché del usuario para analizarla y proceder a un sistema de autenticación.
Este proceso de autenticación también iba ligado a una "API" podríamos llegar a considerarlo aunque nunca llegó a ser tan potente ni bien planificada como la actual.
El proceso de generación de tokens desde un entorno del cliente resulta un proceso algo tedioso y que nunca hay que realizar. Por suerte, para el entorno del servidor de nuestra API había algunas herramientas que nos podrían ayudar.

3.1.1 Generación del token

La generación de tokens siempre es un proceso que me ha plasmado dudas, ya que en un principio pensaba que el almacenamiento de este realizaba sobre la base de datos en vez de en el propio servidor. Por suerte me encontré con el plugin ideal el cual se encuentra en el Node Package Manager, jwt-simple.
Gracias a este plugin podría realizar el codificado y decodificado de un plugin generado cada vez que un usuario iniciara sesión.
Dentro de las fases del codificado de este token necesitamos pasarle como parámetros unas fechas que será la duración que tendrán de validez estos "simbólicos" (traducción de token al español).
El proceso de codificado de nuestro token se realizará dentro de nuestro inicio de sesión. Quedando de la siguiente forma:

```
const createToken = (usuario) => {
    let payload = {
        userId: usuario.idUsuario,
        createdAt: moment().unix(),
        expiresAt: moment().add(1, 'day').unix()
    }
    return jwt.encode(payload, process.env.TOKEN_KEY);
};
```

Esta función se ejecuta cuando hemos podido corroborar que el usuario ha iniciado sesión correctamente y sin problemas.
Lo que codificamos dentro de nuestra variable TOKEN_KEY ubicada en el archivo .env es un payload que cogerá como parámetros: el id del usuario, el momento en el que se ha logueado y el momento donde expirará su sesión.

Teniendo ya nuestro token generado tenemos que de alguna forma hacer que cada vez que quiera hacer una consulta el usuario pueda hacerlo con su token.

¿Por qué almacenamos la id del usuario en nuestro payload?

La id del usuario se almacena porque cada vez que queramos acceder a una consulta siendo clientes podremos meter en nuestra soicitud ese nuevo campo. Esta acción la realizaremos siempre por lo que nos aseguramos disponer de los datos del usuario en cada uno de nuestros casos de uso.
Por ejemplo, si un usuario intenta acceder a una consulta la cual solo se le permite a los técnicos, gracias a poder identificar el id del usuario en base a su token podremos denegar dicha consulta y que no se permita.

3.1.2 El middleware

El middleware, como concepto, representa a todo software que se sitúa entre el software y las aplicaciones que corren sobre él. Este funciona como una capa de traducción que posibilita la comunicación y la administración de datos en aplicaciones distribuidas. En nuestro caso nuestra función no trabajará en si como un "middleware" pero si que será una capa intermedia que se irá ejecutando a medida que el usuario interactúe con nuestra API aportándonos la comprobación de la validez de su token y su id de usuario.
Dentro del middleware tendremos una única función que correrá en cada momento que el usuario acceda a uno de nuestros endpoints.
La implementación sería la siguiente:

```
const jwt = require("jwt-simple");
const moment = require("moment");

const checkToken = (req, res, next) => {
if (!req.headers['user_token'])
    return res.json({
        error: "You must include the header"
    });

const token = req.headers['user_token'];
let payload = null
try {
    payload = jwt.decode(token, process.env.TOKEN_KEY);
} catch (err) {
    return res.json({
        error: "Invalid token"
    });
}

if (moment().unix() > payload.expiresAt) {
    return res.json({ error: "Expired token" });
};

req.userId = payload.userId;

next();
};
```

Dentro de esta función realizamos 3 comprobaciones.

Comprobación de los headers

Nuestro token irá dentro del encabezado de las solicitudes que hagamos a la API. En concreto con la key "user_token". Si no se incluye este encabezado no podríamos continuar con la comprobación por lo que se devuelve como respuesta al cliente *You must include the header.*

Decodificación del payload

Luego de la comprobación de que haya "algo" al menos dentro de nuestro header procedemos a llamar a nuestro plugin de jwt y preguntarle si tiene un payload almacenado con la clave.
En el caso de que no pueda identificarlo, es decir, que no sea válido, obtendremos como respuesta en nuestro sitio web el siguiente mensaje: *Invalid token.*

Verificación de la validez

Después del segundo paso podemos haber obtenido un payload pero tenemos que comprobar que este no haya expirado. Es decir, accedemos a su variables "expiresAt". En el caso de que lo haya hecho devolerá un error como respuesta con el siguiente mensaje: *Expired token.*

Incorporamos el user id a nuestra solicitud y con esto ya tendríamos implementado nuestro middleware.

3.1.3 ¿Cómo añadimos el proceso de verificación a cada consulta?

Gracias a haber hecho nuestro código de forma estructurada y haber distinguido en servicios y rutas con solo añadir una línea a nuestro campo de rutas hará que podamos añadir este paso de verificación intermedio.
Primero importaremos nuestro fichero "middleware" y luego lo llamaremos en el código con la siguiente función:

```
router.use(middleware.checkToken);
```

¿Cómo iniciamos sesión si no tenemos un token?

La respuesta es muy sencilla y va en relación a la lógica de nuestro archivo de rutas. Al entrar una petición a uno de nuestros endpoints lo que ocurre es que lo hace en forma de barrido en nuestros ficheros. Es decir que hasta que no llegue a la línea de código de antes no se habrá hecho la comprobación.
Por lo tanto la solución es que dentro del fichero donde se realice el login y las demás acciones del usuario. Se coloque este primero en la parte de arriba, en medio nuestra comprobación y abajo el resto de consultas que también podemos hacer.

References

Biot, M. A. (1962). Mechanics of deformation and acoustic propagation in porous media. *Journal of applied physics*, *33*(4), 1482–1498.

Heinz, M., Carsten, & Hoffmann, J. (2014, March). *The listings package, march 2014.* http://texdoc.net/texmf-dist/doc/latex/listings/listings.pdf. Retrieved 12/12/2014, from `http://texdoc.net/texmf-dist/doc/latex/listings/listings.pdf`