

FACULTAD DE CIENCIAS

DEPARTAMENTO D INFORMÁTICA Y AUTOMÁTICA



**VNiVERSIDAD
D SALAMANCA**

CAMPUS DE EXCELENCIA INTERNACIONAL

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

GroupApp: Aplicación de gestión de grupos

Realizado por Alonso Núñez, Mario

Dirigido por

Villarrubia González, Gabriel

Sales Mendes, André

De Paz Santana, Juan Francisco

Salamanca, Julio de 2022

CERTIFICADO DE LOS TUTORES

D. Juan Francisco de Paz Santana, D. André Filipe Sales Mendes y D. Gabriel Villarrubia González, profesores del Departamento de Informática y Automática de la Universidad de Salamanca.

HACEN CONSTAR:

Que el trabajo titulado “Aplicación de gestión de eventos y grupos distribuida y centralizada” y la memoria del trabajo, han sido realizados por Alonso Núñez, Mario, con el número de documento 45137109T, con el objetivo de la superación de la asignatura Trabajo de Fin de Grado de la Titulación Grado de Ingeniería Informática de la Universidad de Salamanca

Y para que así conste a todos los efectos oportunos.

En Salamanca, a 7 de Julio de 2022.

D. Juan Francisco de
Paz Santana

D. André Filipe Sales
Mendes

D. Gabriel Villarrubia
González

Resumen del proyecto

El presente proyecto ha sido desarrollado por el estudiante Alonso Núñez, Mario como proyecto Fin de Grado de la titulación de Ingeniería Informática impartida por la Universidad de Salamanca.

El proyecto trata de una aplicación colaborativa de gestión de grupos heterogéneos y eventos de los mismo de una forma intuitiva para los usuarios y versátil para las empresas haciendo especial incapié en la representación eficiente de la información. Esto trata de abarcar la necesidad de organización colectiva de una forma rápida y simple.

El sistema consta de una arquitectura cliente-servidor que proporciona a los usuarios la comodidad de tener acceso a sus datos de forma universal y la capacidad de ponerse en contacto con los demás miembros de la plataforma. El uso de tecnologías híbridas permiten que sea utilizable en una gran diversidad de soportes físicos.

Durante el desarrollo se ha llevado a cabo la elaboración tanto del proyecto cliente, encargado de interacción con el usuario, como del proyecto servidor, cuya funcionalidad abarca el tratamiento de datos como servir de intermediario entre la base de datos y el cliente.

También se ha elaborado la documentación necesaria tanto para entender:

- Las acciones llevadas a cabo por el sistema y los objetivos que intentan conseguir.*
- Manuales que explican como llevar a cabo las distintas acciones de los usuarios de la aplicación.*
- Manuales que explican la estructura del proyecto, el uso de bibliotecas y el montaje del mismo a los futuros desarrolladores.*

summary project

The project deals with a collaborative application for managing heterogeneous groups and their events in an intuitive way for users and versatile for companies, with special emphasis on the efficient representation of information. This tries to cover the need for collective organization in a quick simple way.

The project deals with a collaborative application for managing heterogeneous groups and their events in an intuitive way for users and versatile for companies, with special emphasis on the efficient representation of information. This tries to cover the need for collective organization in a quick and simple way.

The system consists of a client-server architecture that provides users with the convenience of universal access to their data and the ability to contact other members of the platform. The use of hybrid technologies allows it to be usable in a wide variety of physical media.

During development, both the client project, in charge of interaction with the user, and the server project, whose functionality covers data processing and serving as an intermediary between the database and the client, have been developed.

The necessary documentation has also been prepared to understand:

- The actions carried out by the system and the objectives they try to achieve.*
- Manuals that explain how to carry out the different actions of the application users.*
- Manuals that explain the structure of the project, the use of libraries and the assembly of the same to future developers.*

INDICE GENERAL

Lista de Imágenes.....	11
1º Introducción.....	13
2º Objetivos del proyecto.....	15
3º Proyectos y aplicaciones similares.....	17
3.1 Google Calendar.....	17
3.2 Doodle.....	17
3.3 Microsoft Teams.....	18
3.4 Odoos.....	19
4º Conceptos teóricos.....	20
4.1 Arquitectura Cliente-Servidor.....	20
4.2 Aplicaciones híbridas.....	20
5º Herramientas utilizadas.....	21
5.1.1 TypeScript.....	21
5.1.2 Git y GitHub.....	22
5.1.3 Socket.IO.....	23
5.1.4 Jason Web Token.....	24
5.2 Herramientas exclusivas del Frontend.....	24
5.2.1 Angular.....	25
5.2.2 Ionic.....	26
5.2.3 HTML.....	27
5.2.4 SASS.....	27
5.3 Herramientas exclusivas del Backend.....	28
5.3.1 Node y NPM.....	28
5.3.2 Express.....	29
5.3.3 MongoDB y Moongose.....	29
5.3 Herramientas de documentación.....	30
5.3.1 Visual Paradigm for UML.....	30
5.3.2 Microsoft Project.....	31
5.3.3 JSDoc.....	32
6º Aspectos relevantes del desarrollo.....	33
6.1 Metodología de trabajo.....	33
6.2 Planificación temporal del proyecto.....	34
6.3 Especificación de requisitos.....	35
6.3.1 Participantes del proyecto.....	35
6.3.2 Objetivos del sistema.....	35
6.3.3 Requisitos de información.....	36
6.3.4 Requisitos no funcionales.....	37
6.3.5 Requisitos funcionales.....	37
6.4 Análisis de requisitos.....	40
6.4.1 Modelo de dominio.....	40
6.4.2 Paquetes y clases de análisis.....	40
6.4.3 Vista de interacción.....	41
6.4.4 Vista arquitectónica.....	42

6.4 Diseño del sistema.....	42
6.4.1 Patrones empleados.....	43
6.4.2 Sistemas y subsistemas de diseño.....	43
6.4.3 Clases de diseño.....	43
6.4.4 Modelo de diseño.....	44
6.4.5 Modelo de despliegue.....	45
6.5 Momentos destacables del desarrollo.....	46
6.5.1 Estructura y tipo de proyecto.....	46
6.5.2 Implementación de <i>Ionic</i>	48
6.5.3 Creación del paquete <i>DataBaseAccess</i>	49
6.5.4 Implementación de la consistencia de datos.....	50
6.5.5 Creación de tres modelos de datos distintos.....	52
6.5.6 Implementación de gestures.....	56
7° Trabajos relacionados.....	58
7.1 Versión antigua de GroupApp-client.....	58
8° Líneas de trabajo futuras.....	59
8.1 Interacción de usuarios sin registro.....	59
8.2 Implementación del funcionamiento offline.....	60
8.3 Creación de los paquetes ConnectionService.....	61
8.4 Creación de eventos avanzados.....	62
8.5 Implementación de roles dentro de un grupo.....	63
9° Conclusiones.....	65
Bibliografía.....	69
Glosario.....	73

Lista de Imágenes

Figura 1: Ejemplo Google calendar.....	17
Figura 2: Doodle.....	18
Figura 3: Ejemplo Microsoft Project.....	19
Figura 4: Ejemplo código Typescrip vs JavaScript.....	21
Figura 5: Repositorio GitHub de SocketIO.....	22
Figura 6: Ejemplo básico de comunicación mediante SocketIO.....	23
Figura 7: Ejemplo JWT.....	24
Figura 8: Arquitectura Angular.....	25
Figura 9: Ionic distintos dispositivos.....	26
Figura 10: Tipos de código de etiquetado.....	28
Figura 11: MongoDB.....	30
Figura 12: Visual Paradign.....	31
Figura 13: Microsoft Project.....	31
Figura 14: Ejemplo JSDoc.....	32
Figura 15: Fases proceso unificado.....	33
Figura 16: Definición de las tareas a desarrollar.....	34
Figura 17: Gestión de cuentas de usuario.....	38
Figura 18: Ejemplo modelo de dominio.....	40
Figura 19: Ejemplo Diagrama paquete análisis.....	41
Figura 20: Ejemplo clase de análisis.....	41
Figura 21: Ejemplo diagrama de interacción.....	42
Figura 22: Sistema de diseño.....	43
Figura 23: Subsistema Frontend_Model_DeprecatedData.....	44
Figura 24: Ejemplo diagrama de secuencia.....	45
Figura 25: Ejemplo diagrama de estados.....	45
Figura 26: Ejemplo modelo de despliegue.....	46
Figura 27: Estructura del proyecto 2.....	48
Figura 28: Contenido Package.json.....	49
Figura 29: Repositorio DataBaseService.....	50
Figura 30: Funcionamiento JWT 1.....	51
Figura 31: Funcionamiento JWT 2.....	51
Figura 32: Ejemplo código Validar JWT.....	52
Figura 33: Modelo DB.....	53
Figura 34: Modelo SD.....	53
Figura 35: Modelo Data.....	54
Figura 36: Ejemplo código ensamble Data 1.....	54
Figura 37: ficheros ensambleData.....	54
Figura 38: Ejemplo código ensamble Data 2.....	55

Figura 39: Ficheros ensambleDB.....	55
Figura 40: Ejeemplo código ensambleDB 1.....	55
Figura 41: Ejemplo código ensambleDB 2.....	55
Figura 42: Ejemplo implementación gestures 1.....	56
Figura 43: Ejemplo implementación gestures 2.....	56
Figura 44: Ejemplo código gestures.....	57
Figura 45: Ejemplo acceso antiguo.....	58
Figura 46: Ejemplo acceso nuevo.....	58
Figura 47: Servicio local-data.....	60
Figura 48: Clase deprecated Data.....	61
Figura 49: Creación de los paquetes ConnectionService.....	61
Figura 50: Ejemplo creación de eventos avanzados.....	63
Figura 51: Ejemplo implementación de roles dentro de un grupo.....	64

1º Introducción

Las actuales tecnologías han condicionado la forma de interaccionar los unos con los otros hasta el punto de que se han convertido en un elemento clave para hacer que nuestra voz sea escuchada por el resto del mundo.

Grandes plataformas como *Twitter*, *Facebook* o *Instagram* ya han logrado hacer que particulares puedan hablar a grandes masas de personas con la única dificultad de pulsar las teclas de su teléfono móvil. Pese a esto, no contamos con una plataforma eficiente para hacer que diversos grupos de personas puedan coordinarse fácilmente y sin la necesidad de tener que interaccionar directamente los unos con los otros.

La idea de este proyecto surge de una premisa que me plantearon apenas un año atrás, construir una simple aplicación donde los clientes de un servicio de catering puedan ver tanto los horarios como menús de las comidas, a la vez que les permite indicar si asistirán o no a las mismas.

A primera vista se trata de una idea básica con una aplicación muy limitada, por lo que no suscitó gran atención. Sin embargo, poco a poco fue madurando debido a cortas conversaciones con personas cercanas.

“Todo el mundo habla sin decir nada. Sin embargo, cuando nosotras decidimos que hacer, nadie nos escucha por que nuestro mensaje se pierde entre todos los escritos”. Esta frase, o una parecida, fue la que me dijeron dos compañeras que organizaban el movimiento de exámenes en su carrera debido a la subida de casos de COVID-19.

Por otra parte, recuerdo la expresión de mi hermana al hablarme sobre lo difícil que le resultaba, tanto a ella como a su pareja, lograr coordinar su trabajo a realizar con el llevado a cabo por sus compañeros. *“No sabes cuantas veces hemos tenido que cambiar los horarios para poder repartirlo bien ¡Y todo por correo!”*.

Fue a raíz de esto cuando me di cuenta de algo importante. Dentro del saturado mundo actual, en el que la comunicación se encuentra en su máximo apogeo y está cimentada por empresas millonarias que buscan darnos una solución a todo lo que necesitemos. En un momento como este, la población sigue demandando una necesidad sin solución, poder coordinarse con diversos grupos de personas de una forma óptima y sencilla.

Pese a que la idea inicial está bien definida, aún faltan componentes que consigan convertirla en una idea más sólida y llamativa para el uso de los usuarios. Finalmente logré dar con la idea: Integrar bajo una misma aplicación tanto la coordinación sencilla y eficiente del usuario con los grupos a los que pertenece, como poder organizar sus propios horarios y eventos personales.

Esto hace que la propuesta del presente proyecto haya evolucionado hasta lo que realmente es hoy en día, nuestra propia agenda interactiva en la que podemos escribir tanto nosotros como aquellas personas de los grupos a los que pertenecemos y los cuales queramos tener en cuenta.

A lo largo de la presente memoria trataremos las siguientes secciones principales:

- **Objetivos del proyecto:** Expondremos los objetivos que queremos cumplir con el desarrollo del proyecto, tanto requisitos de software como no funcionales.
- **Conceptos teóricos:** Se realiza una breve explicación de conceptos necesarios para que el usuario pueda entender lo expuesto en el documento.
- **Herramientas utilizadas:** Describiremos las diferentes herramientas utilizadas a lo largo del proyecto y el por que de su uso.
- **Aspectos relevantes del desarrollo:** Explicaremos cuales han sido los aspectos y elementos mas destacables que se han producido a lo largo del desarrollo del proyecto, siempre dando un punto de vista y experiencia personales.
- **Trabajos relacionados con el proyecto:** En esta sección trataremos proyectos anteriores que tienen relación con el proyecto presentado.
- **Líneas de trabajo futuras:** Dedicamos este espacio a hablar sobre los planes actuales para la continuación del proyecto y por que llevarlos a cabo.
- **Conclusiones:** Tomamos un último momento para dar nuestras opiniones personales sobre el desarrollo del proyecto y una contraposición del mismo con todo lo elaborado durante la elaboración del grado.

El documento también se encuentra complementado por los siguientes anexos:

- **Anexo 1 - Planificación del proyecto:** Expone la planificación temporal llevada a cabo durante el proceso de desarrollo del proyecto.
- **Anexo 2 – Especificación de requisitos software:** Analiza los requisitos abarcados por el sistema, además de los objetivos y casos de uso.
- **Anexo 3 – Análisis de requisitos:** Recoge el análisis de los requisitos expuestos en el anexo anterior y el desarrollo teórico de los mismos.
- **Anexo 4 – Especificación de diseño:** Documento que explica el diseño del sistema la interacción entre sus componentes.
- **Anexo 5 – Documentación técnica:** Explica la funcionalidad del código elaborado de cara a futuros desarrolladores.
- **Anexo 6 – Manual de usuario:** Documenta los pasos a seguir para desarrollar las diferentes actividades de pueden llevar a cabo los usuarios.

2º Objetivos del proyecto

El objetivo principal del proyecto es el desarrollo de una aplicación colaborativa para gestión de grupos heterogéneos y eventos de los mismos, de modo que la interacción se lleve a cabo de una forma intuitiva y sencilla para los usuarios, a la vez que versátil y potente para las empresas.

Consolidar un sistema mediante el cual, los distintos grupos (ya sean empresas o no) puedan exponer sus eventos de una forma clara, versátil e interactiva con los usuarios de la aplicación. De esta manera, los usuarios podrán ser informados de aquellos eventos que los distintos grupos decidan transmitir de manera pública y externa al mismo.

Los usuarios deben poder integrar los eventos ofrecidos por los diversos grupos y empresas dentro de su organización personal, de manera que todo se cohesione de forma homogénea y permita al usuario organizar de forma sencilla su propia sección dentro de la aplicación.

El proyecto se apoyará en tres pilares fundamentales desarrollar sus objetivos de forma exitosa:

1. Representación eficiente de la información, proporcionando a un usuario la capacidad de acceder de forma rápida, clara y satisfactoria a la misma, consiguiendo informarse en cuestión de segundos sobre aquello que desea consultar.
2. Interacción gratificante tanto con la aplicación como con la información, lo que hará que un usuario pueda permanecer horas consultando grupos y eventos en la misma, de forma ociosa y sin fatiga.
3. Implementación polivalente de la aplicación que facilitará el acceso a la misma, pudiendo ser usada desde cualquier soporte informático, ya sea un portátil, tablet o un dispositivo móvil. Logrando que el acceso a la misma sea tan fácil y rápido como sacar el teléfono del bolsillo.

La siguiente frase determina exactamente aquello que busca desempeñar el presente trabajo: *“Un solo vistazo a tu teléfono móvil u ordenador debe bastar para entender todo aquello que esta ocurriendo tanto en tu entorno personal como en los grupos de tu preferencia.”*

Pasando a un apartado más técnico, podemos diferenciar los siguientes objetivos de software:

- **Gestión de cuentas de usuario:** El sistema deberá almacenar correctamente la información referente a las cuentas de usuario en la base de datos. El tratamiento de la información debe ser versátil y seguro, de modo que el servidor pueda realizar las acciones solicitadas por el usuario tanto de peticiones de datos como de modificación de los mismos
- **Gestión de tablas personales:** El sistema deberá gestionar todas las tablas personales realizadas por los usuarios registrados. Se deberá almacenar toda la información posible referente a dichas tablas como el nombre de las mismas, eventos creados, fechas e información de dichos eventos etc.

- **Gestión de grupos y tablas grupales:** El sistema deberá gestionar la información relativa a todos los grupos y las tablas pertenecientes a los mismos, realizadas por los usuarios registrados. Los usuarios identificados deben poder tanto crear nuevas grupos y tablas en los mismos como eliminar los ya existentes.
- **Gestión de búsquedas y descubrimientos:** El sistema deberá proporcionar al los usuarios identificados un método de búsquedas mediante el cual pueda descubrir los diferentes grupos que están registrados en el sistema.
- **Gestión de notificaciones:** El sistema deberá proveer de un gestor básico de notificaciones que avise al usuario en el momento que se produzca una acción relevante en cualquiera de los grupos en los que se encuentra registrado.

Además de estos, podemos ver los siguientes objetivos no funcionales:

- **Portabilidad:** El sistema deberá funcionar tanto en los principales navegadores web como en la mayor cantidad de dispositivos posibles, ya sean de escritorio, portátiles, móviles o elementos pertenecientes al llamado *internet de las cosas*.
- **Seguridad y protección de datos:** El sistema deberá garantizar la seguridad y privacidad de la información referente a cada uno de los usuarios registrados en el sistema. Esto quiere decir que una persona o entidad ajena no puede obtener la información sensible de terceros usuarios ni del propio sistema.
- **Usabilidad:** El sistema deberá proporcionar una interfaz de usuario cómoda e intuitiva, la cual permita a cualquier tipo de cliente utilizar la aplicación.
- **Funcionamiento en tiempo real:** El sistema deberá actualizarse continuamente para que los cambios establecidos en el mismo sean puestos en funcionamiento lo antes posible. Un usuario debe poder recibir actualizaciones llevadas a cabo en sus grupos sin por ello necesitar recargar o activar la aplicación.

3º Proyectos y aplicaciones similares

3.1 Google Calendar

Google Calendar es una herramienta web desarrollada por *Google* cuyo objetivo es dar al usuario la capacidad de crear eventos en un calendario propio de forma fácil e intuitiva. El uso de esta aplicación se realiza mediante el navegador web y requiere que el usuario esté autenticado con una cuenta de *gmail*.

Calendar también brinda la capacidad de poder invitar a otros usuarios para que puedan participar en la gestión de tu calendario, ya sea de forma activa o únicamente contemplativa.

El gran punto a favor de la herramienta de *Google* es la facilidad e intuitividad a la hora de interactuar con la misma unido a la gran cantidad de usuarios que tienen acceso a la aplicación. Sin embargo, carece de la capacidad de gestión y descubrimiento de grupos que en el presente proyecto intentamos abarcar. En la figura 1 podemos ver el formato de interfaz que sigue *Google Calendar*.

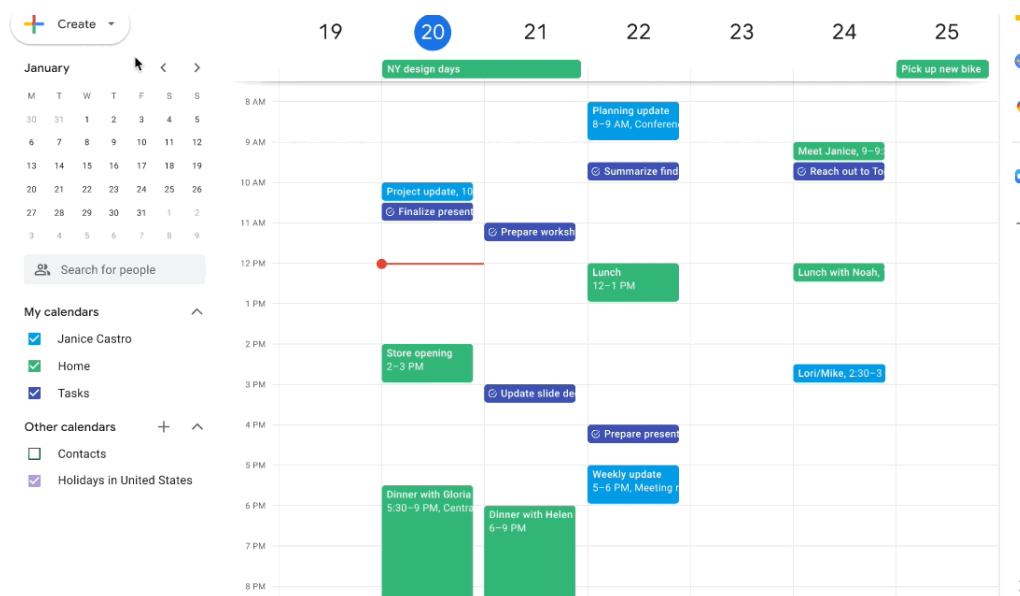


Figura 1: Ejemplo Google calendar

3.2 Doodle

Doodle es una aplicación desarrollada con el fin de proporcionar a un soporte de organización temporal a nivel empresarial, ya sea para grandes organizaciones o pequeños empresarios.

La aplicación está pensada para que un grupo de personas puedan compartir un mismo organizador de forma rápida y potente, de modo que sean capaces de llevar a cabo

modificaciones y especificaciones que ayuden a organizar su entorno de trabajo. En la figura 2 podemos ver como es la interfaz de *Doodle*.

A diferencia de la herramienta de *Google* vista anteriormente, uno de los principales objetivos de *Doodle* es organizar y gestionar reuniones empresariales, ya sean entre los miembros de la empresa o con clientes.

A parte de esto, la herramienta proporciona una gran diversidad de funcionalidades, como gestión de encuestas, gestión de roles y permisos o integración con las distintas herramientas de videoconferencia.

El principal inconveniente de *Doodle* es que al estar enfocado a un ambiente empresarial, si modelo de rentabilidad es mediante pago mensual, lo que lo convierte en una aplicación alejada del gran público.

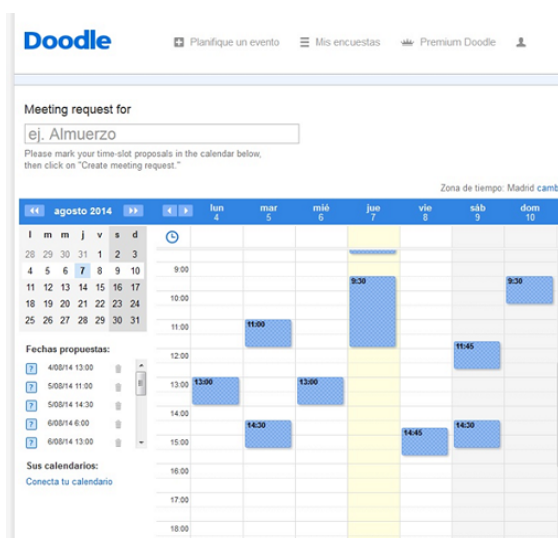


Figura 2: Doodle

3.3 Microsoft Teams

Llegamos a otra de las grandes aplicaciones para la gestión de grupos. *Microsoft Teams* es una de las más utilizadas a nivel empresarial, ya que proporciona a sus usuarios la capacidad de organizar las intervenciones de todo el equipo en una estructura de temas fácil de entender y utilizar.

La gran versatilidad a la hora de utilizar, no únicamente chat escritos en forma de intervenciones, sino una gran variedad de contenidos multimedia, hace que se trate de una herramienta excelente para el intercambio de recursos dentro de del marco de trabajo.

Al igual que la aplicación anterior, *Teams* no esta diseñado para que sea una aplicación utilizable por todo el mundo y con una vertiente más social, lo que lo reduce únicamente a ser usado en un ámbito de trabajo o educativo.

Otro de los grandes inconvenientes es su interfaz de usuario, la cual demuestra estar diseñada únicamente para su utilización en entornos de escritorio. Tratarse de una aplicación no utilizable en dispositivos móviles le resta la capacidad de ser una herramienta disponible en todo momento. En la figura 3 se expone un ejemplo de como se ve un proyecto de grupos en *Microsoft Teams*.

GroupApp: Aplicación de gestión de eventos y grupos distribuida y centralizada

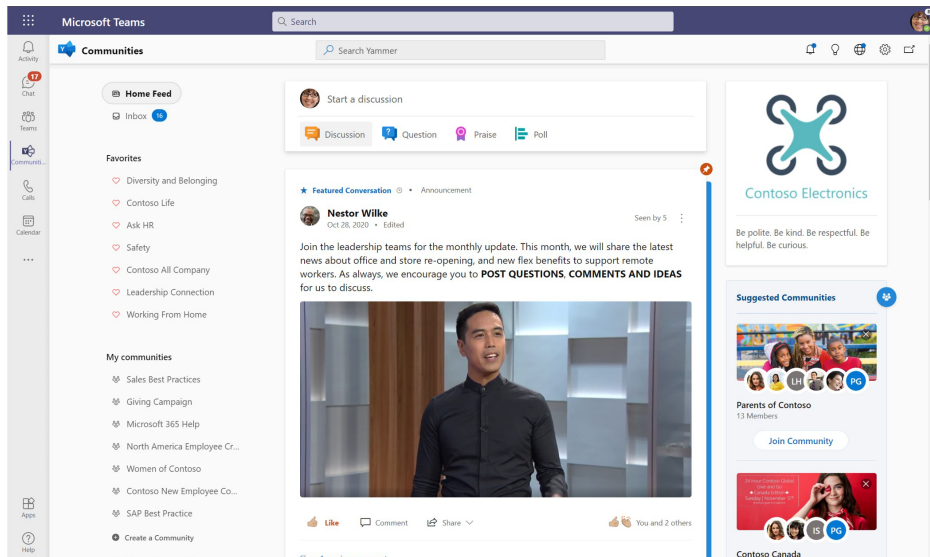


Figura 3: Ejemplo Microsoft Project

3.4 Odo

Odo es una aplicación multiplicación que es capaz de proporcionar una gran cantidad de herramientas para la gestión de una empresa. Se trata del concepto de navaja suiza llevada al ámbito de gestión empresarial.

De entre todas las prestaciones que nos aporta Odo se encuentran la gestión del trabajo, la planificación de las distintas actividades, organización de los datos empresariales, desarrollar el ciclo de vida de los productor o realizar controles de calidad.

Entre todas las funcionalidades de la aplicación podemos encontrar la gestión de eventos en forma de calendarios. Esto nos permite poder gestionar las actividades de los miembros integrantes de nuestra empresa de una forma visual e interactiva.

Sin embargo, al igual que ocurre con *Microsoft Teams*, el uso de esta aplicación esta completamente orientada al desarrollo dentro de un ámbito empresarial, por lo que no cuenta con la capacidad de interacción social en un ambiente abierto que te permita descubrir nuevos grupos.

4º Conceptos teóricos

4.1 Arquitectura Cliente-Servidor

La arquitectura de un proyecto software es un conjunto de patrones y abstracciones que permiten proporcionar un marco definido de la aplicación y la especificación de la interacción entre los distintos componentes de la misma. Tener una arquitectura software bien definida es importante ya que sirve de guía a los desarrolladores a la hora de llevar a cabo el desarrollo.

El presente proyecto ha sido desarrollado siguiendo una arquitectura cliente-servidor. Este es un tipo de arquitectura distribuida donde dos o más nodos independientes entre sí se comunican para llevar a cabo el procesamiento de información conjunta.

Existen varios tipos de arquitecturas distribuidas, sin embargo, la arquitectura cliente-servidor se caracteriza por ser centralizada, de modo que existe un nodo que tiene una mayor carga de procesamiento de información general (el servidor) y una gran cantidad de pequeños nodos que únicamente procesan la información local (los clientes).

4.2 Aplicaciones híbridas

El concepto de aplicación híbrida puede ser nuevo para muchas de las personas que leen el presente escrito. Una aplicación híbrida es aquella que ha sido diseñada para poder funcionar en una gran diversidad de sistemas, cada uno con características físicas diferentes.

Dejar claro este concepto es importante, puesto que una gran parte del proyecto se trata de una aplicación híbrida. Como ya hemos visto anteriormente, el proyecto consta de tres fases principales, de las cuales, el subproyecto dedicado al desarrollo del *Frontend* se trata de una aplicación híbrida.

Esto es debido a que hemos decidido utilizar la herramienta *Ionic*, la cual, aunque trabaja utilizando los diferentes *frameworks* de desarrollo web que existen, esta diseñada para adaptar un proyecto a una gran diversidad de plataformas y sistemas operativos.

El uso de una plataforma híbrida trae consigo la gran ventaja de poder desarrollar un único proyecto con el que dar soporte a toda la base de usuarios que utilicen la aplicación.

Por otra parte, esto trae consigo problemas como la necesidad de tener que implementar servicios que identifiquen y operen de manera distinta dependiendo el soporte físico, o bien disponer de una menor rendimiento a la hora de ejecutar la aplicación.

5º Herramientas utilizadas

El presente trabajo puede desarrollarse en una gran cantidad de herramientas de programación, cuyo único requisito es que consten de entorno gráfico para el usuario, pues de lo contrario no tendría sentido llevarlo a cabo.

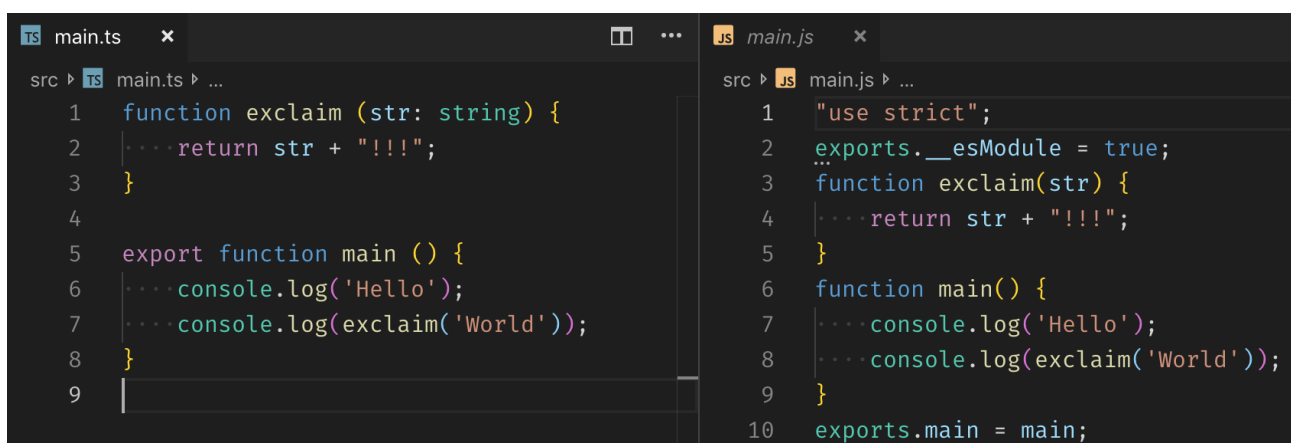
Tras meditarlo sin saber muy bien como llevarlo a cabo, la decisión final fue elaborar el proyecto utilizando herramientas de desarrollo web. Las necesidades a cumplir no se limitaban únicamente de utilizar un *Framework* web, sino que requerían llevar a cabo el desarrollo de una aplicación completa, la cual debía incluir tanto un *Frontend* como un *Backend* y la gestión de una base de datos propia.

5.1 Herramientas comunes a todo el proyecto

5.1.1 TypeScript

TypeScript es un lenguaje de programación desarrollado por *Microsoft* en el año 2012 y conforma un nivel superior del lenguaje de programación *JavaScript*. Esto quiere decir que *TypeScript* funciona en base a *JavaScript*, de modo que cuando queremos ejecutar un código, este se traduce primero a código nativo *JavaScript* y posteriormente se ejecutara con el mismo compilador que usa este último.

En este [enlace](#) enlace podremos encontrar la pagina oficial de *Typescript*, la cual esta llena de recursos y documentación para aprender a utilizar el lenguaje. En el caso de querer realizar un vistazo rápido al funcionamiento del mismo, recomendamos leer el tutorial oficial de introducción [aquí](#). En la figura 4 podemos ver una contraposición del programa Hola mundo escrito en *Typescript* y el *JavaScript*



```
main.ts x
src > TS main.ts > ...
1 function exclaim (str: string) {
2   ...return str + "!!!";
3 }
4
5 export function main () {
6   ...console.log('Hello');
7   ...console.log(exclaim('World'));
8 }
9

main.js x
src > JS main.js > ...
1 "use strict";
2 exports.__esModule = true;
3 function exclaim(str) {
4   ...return str + "!!!";
5 }
6 function main() {
7   ...console.log('Hello');
8   ...console.log(exclaim('World'));
9 }
10 exports.main = main;
```

Figura 4: Ejemplo código Typescrip vs JavaScript

La mayoría de los programadores web utilizan *JavaScript* para llevar a cabo sus proyectos, sin embargo, hemos decidido utilizar *Typescript* como lenguaje de programación principal del proyecto debido a varias ventajas derivadas de su uso:

- TypeScript es un lenguaje orientado al uso de tipos que nos permite crear nuestras propias clases y definir el tipo de dato de cada uno de los parámetros utilizados en nuestro proyecto. Además de esto, las propias bibliotecas nativas de TypeScript ya incluyen clases complejas definidas que no existen en JavaScript.
- El uso de tipos a la hora de llevar a cabo el desarrollo de nuestro proyecto hace que el IDE sea capaz de detectar una gran cantidad de errores en tiempo de compilación, evitando así sufrir los mismos en el momento de la ejecución.
- La posibilidad de crear interfaces que nos indican la estructura de un conjunto de datos sin la necesidad de tener que definir una clase. Esto será muy útil en el paso de mensajes entre el cliente y el servidor.
- Las funciones definidas en *Typescript* permiten recibir parámetros opcionales, lo cual facilita la tarea de definir las funcionalidades asociadas a las clases.

En el siguiente [enlace](#) podemos ver un análisis y opinión de las diferencias entre *TypeScript* y *JavaScript*, tanto a nivel de desarrollo como de uso en la comunidad, además de algunas conclusiones reseñables.

5.1.2 Git y GitHub

Git es una herramienta de control de versiones que nos permite llevar a cabo el desarrollo de un proyecto salvaguardando diversos puntos del desarrollo del mismo y estructurándolos en un árbol de versiones (todo registrado en fichero *.git* que se encuentra en la base de cada una de las partes que componen el proyecto).

El uso de Git es muy versátil y nos provee de una gran cantidad de acciones a llevar a cabo con las versiones realizadas. La capacidad de poder retomar el estado del proyecto correspondiente a un punto anterior del tiempo y posteriormente volver a la situación actual nos permite corregir errores críticos o solventar problemas externos al desarrollo del mismo, como la corrupción del código.

Por si fuera poco, Git es capaz de sincronizarse con sitios web de repositorios Online como *GitHub* que nos permite subir al mismo las distintas versiones de nuestro proyecto y descargarlas con únicamente ejecutar un comando. En la figura 5 podemos visualizar el contenido del repositorio de *GitHub* de la biblioteca *Socket.IO*.

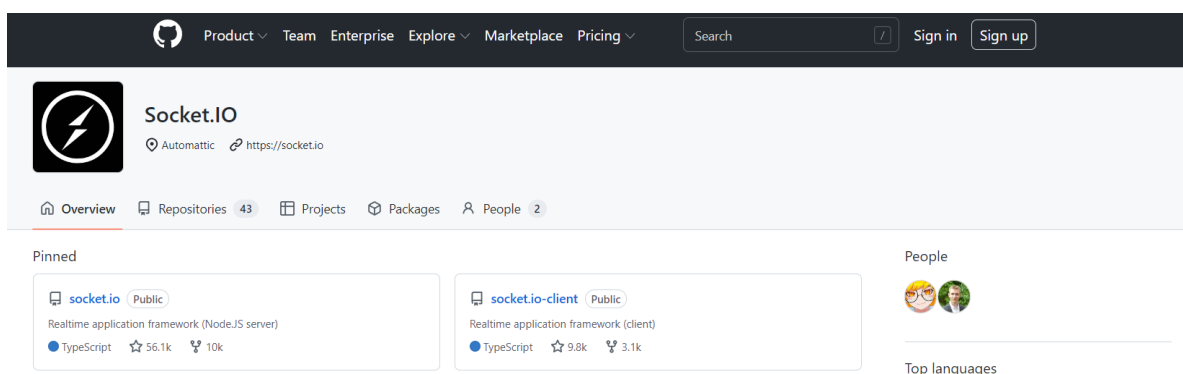


Figura 5: Repositorio GitHub de SocketIO

En el siguiente [enlace](#) podemos ver la pagina web de *Git* con su documentación oficial. Por otra parte, [aquí](#) podremos encontrar el portar de acceso a *GitHub*.

Todo proyecto serio debería utilizar un sistema de control de versiones como *Git* para poder solventar los errores que puedan producirse, a la vez, si el proyecto es código abierto, también debería sincronizarse con un sistema de control Online de versiones.

En nuestro caso, hemos decidido llevar un control de versiones diferentes para cada parte del proyecto, de modo que el *Frontend* y el *Backend* están organizados por árboles de versiones diferentes.

En el caso de querer aprender las bases para el uso de *Git* y *GitHub* recomendamos leer el siguiente [tutorial](#).

5.1.3 Socket.IO

Socket.IO es una biblioteca realizada en JavaScript cuyo objetivo es facilitar al desarrollador el trabajo a realizar necesario a la hora de llevar a cabo una comunicación bidireccional entre el Frontend y el backend de la aplicación.

La funcionalidad de esta herramienta se basa en el uso de *Sockets* de bajo nivel y la definición de identificadores a los cuales el desarrollador asocia manejadoras que se ejecutan cuando se recibe un mensaje correspondiente con dicho identificador.

En el siguiente [enlace](#) podemos ver el repositorio oficial de *NPM* donde se encuentra registrada la biblioteca de *Socket.IO*. En la figura 6 visualizamos un código simple que implementa el envío y recepción de mensajes en el cliente y en el servidor usando *Socket.IO*.



```
import { Server } from "socket.io";

const io = new Server(3000);

io.on("connection", (socket) => {
  // send a message to the client
  socket.emit("hello", "world");

  // receive a message from the client
  socket.on("howdy", (arg) => {
    console.log(arg); // prints "stranger"
  });
});
```

```
import { io } from "socket.io-client";

const socket = io("ws://localhost:3000");

// receive a message from the server
socket.on("hello", (arg) => {
  console.log(arg); // prints "world"
});

// send a message to the server
socket.emit("howdy", "stranger");
```

Figura 6: Ejemplo básico de comunicación mediante SocketIO

Socket.IO es uno de los elementos fundamentales para la realización del presente proyecto debido a la necesidad de crear un puente de comunicación bidireccional asíncrono entre el cliente y el servidor.

Existen más bibliotecas para el uso de *WebSocket*, sin embargo, características como la escalabilidad y el alto rendimiento aportados por *socket.IO* junto con características ya implementadas de manera nativa como el registro de conexiones y desconexiones con reintentos hace que sea una gran opción a utilizar.

Una de las características que tiene esta herramienta es la existencia de dos bibliotecas diferentes, una para el cliente y otra para el servidor, reduciendo así la carga de elementos no utilizados en cada uno de los lados de la aplicación.

En la [página](#) oficial de *Socket.IO* podemos encontrar documentación sobre su instalación y su implementación por parte del desarrollador.

5.1.4 Jason Web Token

Podemos definir *Jason Web Token* como un elemento utilizado para difundir las credenciales de los usuarios de una plataforma de manera bidireccional y siguiendo los estándares de seguridad. En la siguiente [página](#) podemos encontrar la web oficial de *JWT*, la cual contiene información sobre su uso e implementación.

Cabe destacar de que *JWT* no es una librería concreta, sino un estándar que define como realizar el proceso de forma segura. En el siguiente [enlace](#) podemos ver una lista de las diferentes librerías existentes para el tratamiento de *JWT* y las funcionalidades que implementan correctamente.

En la práctica se trata de una simple cadena de texto encriptada por parte del servidor donde se puede enviar información sensible, la cual necesitaría de una gran prueba de esfuerzo para poder obtener los datos. En la figura 7 podemos ver la tres partes que conforman la cadena de texto de un *JWT*.

Esta herramienta es muy utilizada actualmente en aplicaciones web que requieren de verificación de credenciales. Nosotros la utilizaremos en nuestro proyecto con el fin de asegurar la consistencia de los datos devueltos por el servidor sin la necesidad de tener que utilizar almacenamiento local para ello.

En apartados posteriores hablaremos de como hemos implementado *JWT* a nuestro proyecto y del funcionamiento básico del mismo.

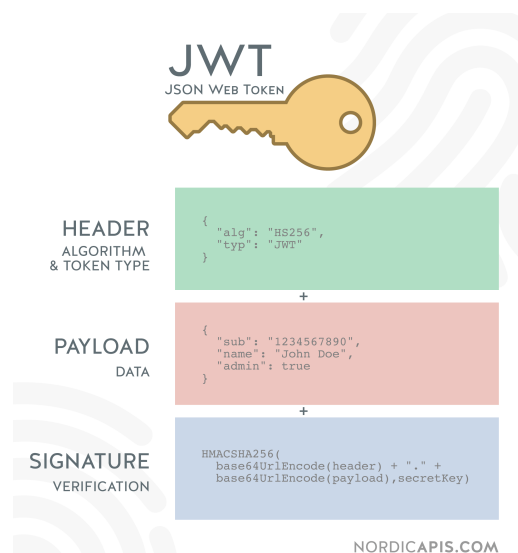


Figura 7: Ejemplo JWT

5.2 Herramientas exclusivas del Frontend

5.2.1 Angular

Angular es un *Framework* de código abierto orientado al desarrollo web de aplicaciones *Frontend*, se encuentra desarrollado en base al lenguaje *TypeScript* y es mantenida y actualizada por la empresa *Google*.

Dentro de sus aptitudes cabe resaltar que se trata de una de las herramientas de desarrollo web más utilizadas con un alcance mundial y es capaz de brindar al desarrollador facilidades como disponer de una gran biblioteca propia de trabajo e integración con múltiples herramientas externas como *Jasmine*.

Importante destacar que *Angular* trabaja intrínsecamente con las herramientas *HTML* y *CSS* para poder así construir una interfaz gráfica atendiendo a una arquitectura de *componentes*. Un punto positivo a tener en cuenta es que *Angular* ya esta preparado para que el usuario trabaje utilizando dicha arquitectura sin tener que preocuparse la organización de la implementación de la misma.

Para entender mejor el funcionamiento de Angular, en el siguiente [enlace](#) podréis encontrar la documentación oficial de introducción al mismo. En la figura 8 vemos un esquema simplificado de la arquitectura implementada por Angular.

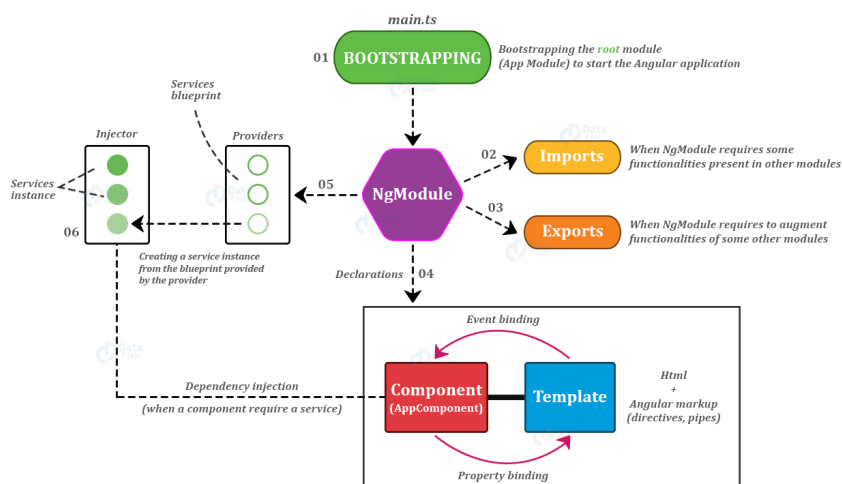


Figura 8: Arquitectura Angular

Una vez se tenía claro el objetivo principal a abarcar y el uso de herramientas web, se dedicó tiempo a analizar cada una de las opciones que podían desempeñar este papel el importante papel del *Framework Frontend*, destacando tres opciones lógicas: *Angular*, *React* y *Vue.js*.

Descartar la opción de *Vue.js* fue bastante rápida, pues hace unos 9 meses (desconozco la situación actual) aún se implementaban importantes cambios en el software que obligaban a sus desarrolladores tener que modificar buena parte de sus proyectos para adaptarlo a los mismos. Teniendo en cuenta que en el momento de comenzar el proyecto mis conocimientos sobre herramientas web eran nulos, decidimos evitar problemas e introducirnos en el ámbito usando herramientas más estables.

Por otra parte, aunque la curva de aprendizaje de *React* es más amigable que la de *Angular*, la capacidad de administrar todo el proyecto en componentes independientes y el uso nativo de *TypeScript* hicieron que fuera la opción final para llevar a cabo la aplicación.

En este [artículo](#) podréis leer una comparación entre los distintos aspectos de los tres *frameworks* comentados.

5.2.2 Ionic

Ionic es un *framework* de código abierto empleado para la creación de aplicaciones híbridas que utiliza como base los *frameworks* de *Angular*, *React* o desde hace unos meses *Vue.js*.

Aunque esto puede sonar un poco extraño al principio, es fácil de entender, *Ionic* no proporciona de por sí un marco de trabajo completo, sino que implementan los *frameworks* anteriormente citados para dar la estructura al proyecto.

El objetivo de *Ionic* es poder adaptar y traducir las aplicaciones llevadas a cabo con el marco de trabajo al cual envuelve, con la finalidad de convertirlas en una aplicación híbrida que pueda adaptarse a las distintas plataformas utilizadas actualmente. En la figura 9 visualizamos una misma aplicación de *Ionic* siendo ejecutada en sistemas móvil que siguen estilos de diseño diferentes.

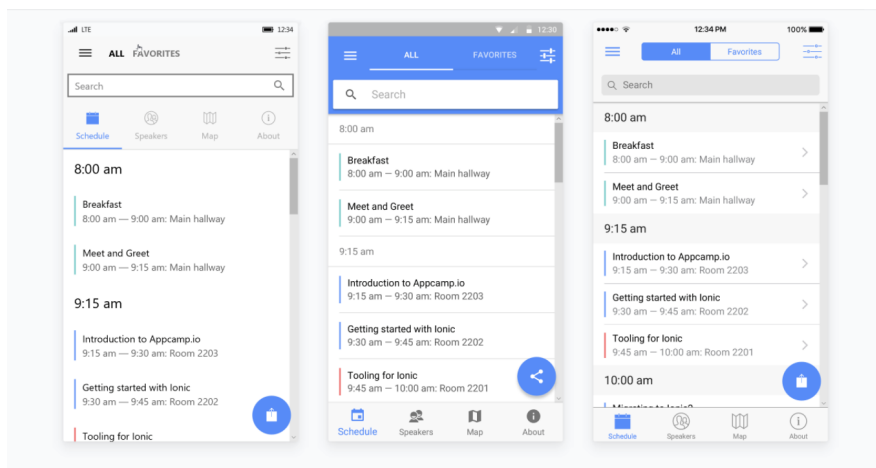


Figura 9: Ionic distintos dispositivos

En el siguiente [enlace](#) podemos encontrar la página oficial de *Ionic*, la cual está llena de documentación para la configuración del mismo y de tutoriales de todos los componentes que implementa en cada uno de los *frameworks* sobre los que trabaja..

Las aplicaciones híbridas tienen la desventaja de contar con un rendimiento menor que las aplicaciones dedicadas a una plataforma concreta. Esto es debido a que al implementarse de manera óptima para varias plataformas no pueden estar tan optimizadas para cada una de ellas en concreto.

Pese a esto, *Ionic* nos aporta la gran ventaja de poder desarrollar un mismo *Frontend* común a todas las plataformas en la que se va a implementar el proyecto, de modo que podremos crear, por ejemplo, una *.pwa* para dispositivos móviles sin la necesidad de hacerla desde cero. También podremos mantener un estilo más cohesivo y normalizado entre las diversas versiones.

5.2.3 HTML

HTML es un lenguaje de Hipertexto que define tanto el significado como la estructura del diseño web que implementaremos en nuestra aplicación. *HTML* es uno de los componentes básicos del diseño web actual y en nuestro proyecto se encuentra implementado de forma nativa dentro de *Angular*.

Gracias a que utilizamos la capa superior que nos proporciona *Ionic*, muchos de los elementos clásicos implementados mediante el lenguaje *HTML* serán sustituidos por los elementos de este último.

Debido a que *Angular* implementa un paradigma orientado a componentes, el lenguaje *HTML* utilizado en el interior de cada uno de estos sirve como nexo de unión e intercambio de información entre los diversos componentes que se implementan unos dentro de otros. En el caso de querer aprender *HTML*, recomendamos seguir los tutoriales indicados en al siguiente [página](#) web.

5.2.4 SASS

SASS es un lenguaje de etiquetado que define los estilos implementados en los componentes del código *HTML* en el proyecto. Este se trata de un preprocesador de *CSS*, el cual nos permite definir características extras que no se podrían hacer en este último.

Esta herramienta nos permite implementar nuestro código de estilos en formato *SASS* como en formato *SCSS* (este último es el utilizado en la totalidad del proyecto *frontend*). En el siguiente [enlace](#) a la página web oficial de *SASS* podemos encontrar documentación y guías para llevar a cabo la instalación y saber utilizar la herramienta.

Normalmente se utiliza *CSS*, y sería lo lógico debido a que es el lenguaje de estilos implementado nativamente por *Angular*, sin embargo, en proyecto se utiliza *SASS* debido a que este es implementado directamente en todos los proyectos de *Ionic*.

El código *SASS* se caracteriza no solo por proporcionar características extras, sino por poder implementarse en el mismo cualquier código *CSS* clásico, por lo que no supone ningún problema adicional trabajar con el mismo.

En el caso de querer aprender a utilizar *CSS* recomendamos la lectura de los siguientes [tutoriales](#). En la figura 10 podemos ver una comparativa entre los tres tipos de código de etiquetado que podemos utilizar:

SASS	SCSS	CSS
<pre> \$color: red \$color2: lime a color: \$color &:hover color: \$color2 </pre>	<pre> \$color: #f00; \$color2: #0f0; a { color: \$color; &:hover { color: \$color2; } } </pre>	<pre> a { color: red; } a:hover { color: lime; } </pre>

Figura 10: Tipos de código de etiquetado

5.3 Herramientas exclusivas del Backend

5.3.1 Node y NPM

Node es un entorno de trabajo de código abierto y multiplataforma que permite a los usuarios crear aplicaciones y herramientas *backend* usando el lenguaje *JavaScript* (aunque también es posible utilizar *Typescript*, tal y como nosotros hemos hecho).

En el siguiente [enlace](#) encontraremos la página web de *Node*, donde no solo podemos descargar el programa, sino que tenemos a nuestra disposición información y documentación sobre el mismo.

Una de las principales ventajas de *Node* es el uso del gestor de paquetes *NPM*, el cual hace que sea muy sencillo mantener actualizada nuestra aplicación y administrar aquellos paquetes que queremos que formen parte de la versión final y los que únicamente usaremos durante el desarrollo del proyecto.

Otro de los principales motivos por el que hemos decidido utilizar *Node* es su gran popularidad en el uso de aplicaciones para servidores, su alto rendimiento y optimización.

Como podremos ver más adelante, también hemos decidido crear nuestro propio paquetes para la parte del proyecto que se encargará del acceso a la base de datos y publicarlo en los repositorios oficiales de *NPM*.

En este [enlace](#) tenemos la página de acceso al portal oficial de *NPM* y en este [otro](#) encontramos el repositorio del paquete que hemos creado para la gestión del acceso a la base de datos.

5.3.2 Express

Express es uno de los *frameworks* más populares de *node* para la creación de aplicaciones web de servidor. En el siguiente [enlace](#) podremos encontrar el repositorio de *NPM* donde se encuentra el paquete *Express*.

Esta herramienta trae consigo el uso de varias características necesarias para la creación y desarrollo de aplicaciones *backend* como la escucha de los puertos de forma automática, y el uso de las URLs junto con la diferenciación de peticiones como *POST* y *GET*.

Decidir que *framework* usar para la creación del servidor es una decisión importante, aunque en este caso fue bastante sencilla. *Express* no solo facilita enormemente el trabajo a realizar, sino que se trata del marco de trabajo más utilizado para la creación de servidores.

Esta popularidad trae consigo una mayor facilidad para obtener documentación y un mayor rendimiento de la aplicación. En la [página](#) oficial de *express* podemos encontrar tanto información general sobre el mismo como explicaciones acerca de su uso y documentación variada.

5.3.3 MongoDB y Moongose

MongoDB es una base de datos distribuida basada en el registro de documentos sobre los cuales se pueden realizar gran diversidad de consultas y operaciones. Los documentos registrados siguen la estructura *JSON*, de modo que son fácilmente tratables a la hora de interactuar con otras herramientas.

Una de las principales ventajas que aporta *MongoDB* es su gran optimización y rapidez a la hora de realizar consultas y operaciones sobre los datos registrados. También se trata de una herramienta multiplataforma y multilenguaje, de modo que tiene gran flexibilidad a la hora de ser utilizada.

En el siguiente [enlace](#) podemos encontrar la página web oficial de *MongoDB*, donde no solo entramos información sobre la misma y las herramientas anexas que podemos utilizar, sino que también podemos registrarnos y crear nuestro propio servidor de datos.

Además de esto, *MongoDB* consta de un paquete construido en *JavaScript* para su interacción de *Node*. Este paquete se llama *Moongose*, es utilizado directamente en el proyecto y podemos ver su repositorio de *NPM* en el siguiente [enlace](#). En la figura 11 podemos ver la infraestructura sobre la cual trabaja *MongoDB*.

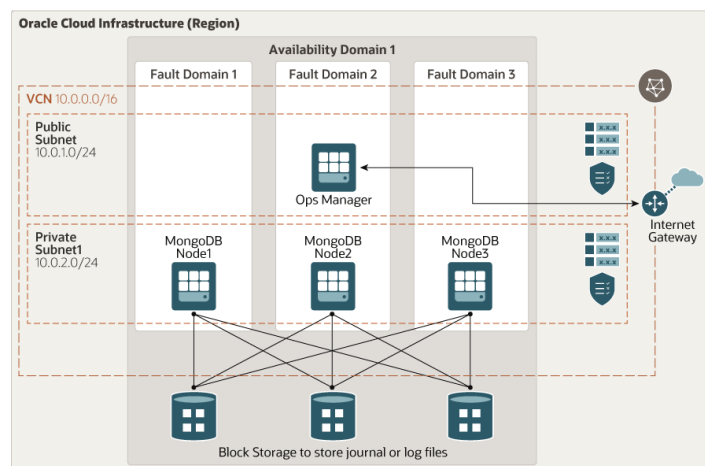


Figura 11: MongoDB

Existen varios motivos para utilizar *MongoDB*, uno de ellos es la gran cantidad de recursos gratuitos de los que puedes disponer si utilizas esta herramienta. *Mongo Atlas* pone a disposición de sus usuarios servidores físicos reales gratuitos en los que pueden crear sus propias bases de datos, lo cual hace que sea una alternativa muy utilizada actualmente.

Otras herramientas como *Mongo Compass* nos permiten consultar y modificar fácilmente el contenido de nuestra base de datos de forma gráfica. Por otra parte, utilizar un sistema de bases de datos que se encuentra en pleno auge es una opción a tener en cuenta debido a las futuras posibilidades que esto puede brindar.

En nuestro caso, debido a la envergadura del servicio con el que realizamos el acceso y las peticiones al servidor de datos de *MongoDB*, hemos decidido encapsular dicha funcionalidad en un paquete de *Node*.

5.3 Herramientas de documentación

5.3.1 Visual Paradigm for UML

Visual Paradigm es una herramienta que nos permite realizar el modelado de diferentes elementos empleados en la documentación dentro del ámbito del desarrollo software. Se trata de uno de los programas más potentes del sector y abarca una gran cantidad de elementos que pueden ser modelados.

El uso de *Visual Paradigm* en el desarrollo de la documentación del proyecto es algo crucial debido a que nos permite modelar los diagramas utilizados durante las fases de especificación y análisis de requisitos (Equivalentes a los anexos 3 y 4).

La utilización de la herramienta está restringida a la adquisición de una licencia comercial, sin embargo, para el desarrollo del proyecto se ha utilizado una licencia gratuita proveída

GroupApp: Aplicación de gestión de eventos y grupos distribuida y centralizada

por la Universidad de Salamanca. A continuación en la figura 12, visualizamos un ejemplo de diseño utilizando el *Visual Paradigm*.

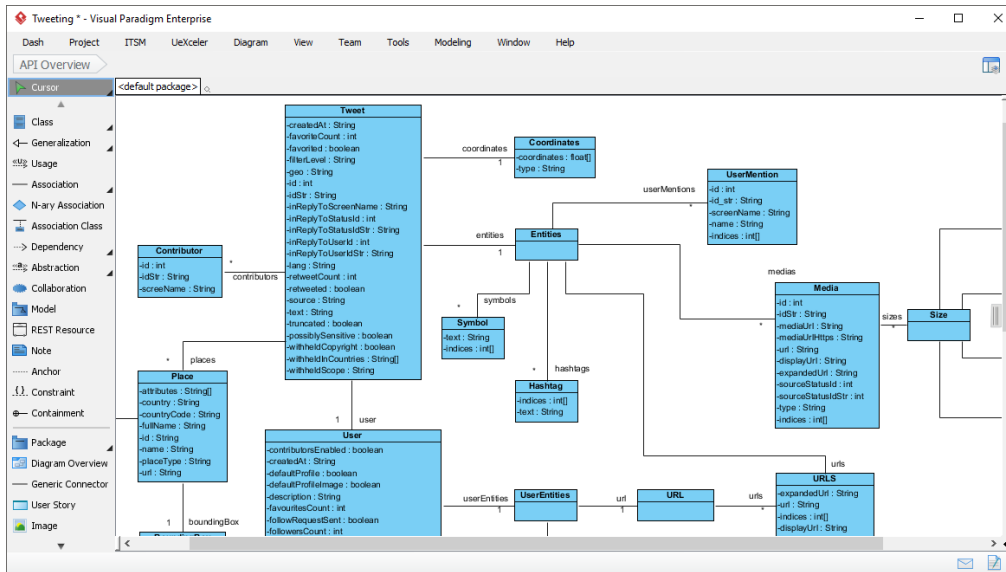


Figura 12: Visual Paradigm

5.3.2 Microsoft Project

Microsoft Project es un programa de gestión de proyectos cuya finalidad es proporcionar al administrador la capacidad de gestionar los distintos elementos que componen e influyen durante el proceso de desarrollo de un producto software. En la figura 13 podemos ver el modelado de un diagrama de *Gantt* utilizando *Microsoft Project*.

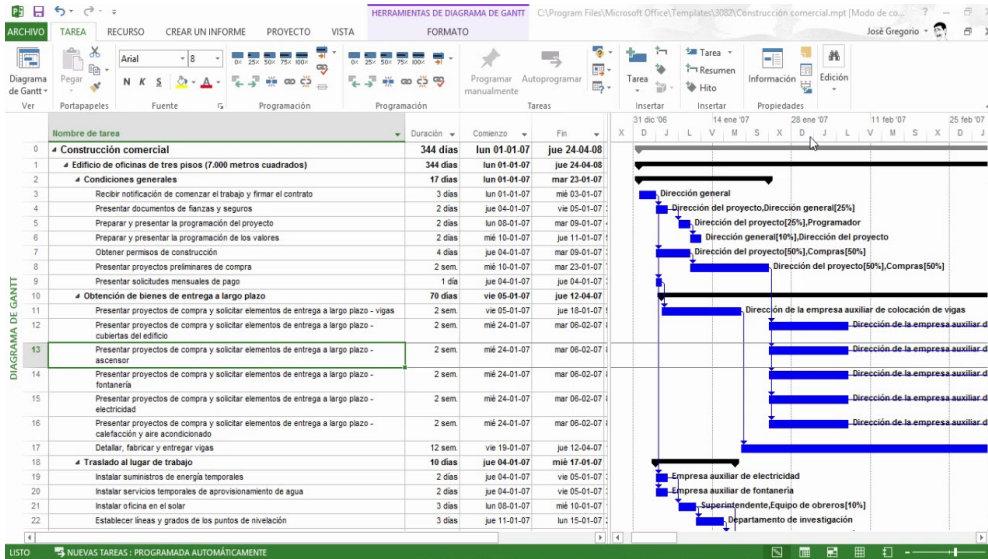


Figura 13: Microsoft Project

La empresa creadora *Microsoft* ha conseguido crear uno de los programas de gestión de proyectos más completos del mercado, esto es debido a que la persona encargada de gestionar el proyecto puede definir aspectos relevantes del mismo como: Las actividades

a realizar durante el desarrollo, los recursos disponibles a utilizar, definición de los calendarios que marcarán el desarrollo, relaciones entre las actividades etc.

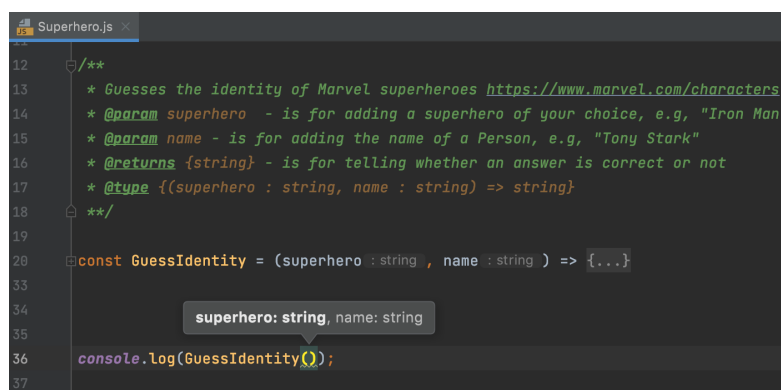
La implementación de dicha herramienta en el desarrollo de la documentación del proyecto es importante debido a que permite organizar las diversas actividades en diagramas de Gantt. Esto nos proporciona la capacidad de mostrar al lector una vista general y gráfica del desarrollo del proyecto (En el Anexo 1 podemos ver el uso de esta herramienta).

5.3.3 JSDoc

JSDoc es una herramienta de generación automática de documentación para el lenguaje *JavaScript*. La documentación producida se almacena en un conjunto de ficheros *HTML*, de modo que podremos visualizarla mediante el uso de un navegador web de forma interactiva.

Existen un conjunto de extensiones que hacen más versátil a *JSDoc*, en nuestro caso hemos utilizado aquella que nos permite utilizar la herramienta adaptándola al lenguaje *Typescript*.

El uso de herramientas como *JSDoc* es muy importante debido a que proporcionan un estándar a la hora de crear documentación orientada al uso por parte de terceros desarrolladores. Este es el motivo de utilizar esta herramienta y no generar una documentación manual que no sigue ningún estándar. En la figura 14 podemos ver un ejemplo de comentario empleado en *JSDoc*.



```

12  /**
13   * Guesses the identity of Marvel superheroes https://www.marvel.com/characters
14   * @param superhero - is for adding a superhero of your choice, e.g, "Iron Man"
15   * @param name - is for adding the name of a Person, e.g, "Tony Stark"
16   * @returns {string} - is for telling whether an answer is correct or not
17   * @type {(superhero : string, name : string) => string}
18  */
19
20  const GuessIdentity = (superhero : string , name : string ) => {...}
21
22
23
24
25
26  console.log(GuessIdentity());
27

```

Figura 14: Ejemplo JSDoc

En nuestro caso hemos utilizado la herramienta únicamente para documentar la biblioteca *DataBaseService*. Esto es debido a que al tratarse de un paquete, los usuarios que lo implementan en su código tienen esta documentación como principal referencia.

En el resto de la aplicación hemos decidido seguir las recomendaciones que *Robert C. Martin* explica en su libro *Código Limpio*, optando por un código libre de comentarios pero que sea capaz de explicarse el solo gracias a la información proporcionada por los nombres (ya sea de variables o funciones) empleados en el mismo.

6º Aspectos relevantes del desarrollo

6.1 Metodología de trabajo

Durante la elaboración del proyecto se ha empleado la metodología designada por el proceso unificado. Se trata de un modelo de desarrollo iterativo e incremental, de modo que el desarrollo está dividido en diversas iteraciones que tienen como objetivo producir un incremento de las funcionalidades del producto.

El modelo unificado está centrado en la utilización de casos de uso, los cuales definen todas las acciones que se pueden llevar a cabo en la aplicación entre el usuario y el sistema. En cada una de las iteraciones se busca desarrollar nuevos casos de uso e integrarlos con los ya existentes. En la figura 15 visualizamos un esquema de las distintas fases que componen el Proceso Unificado.

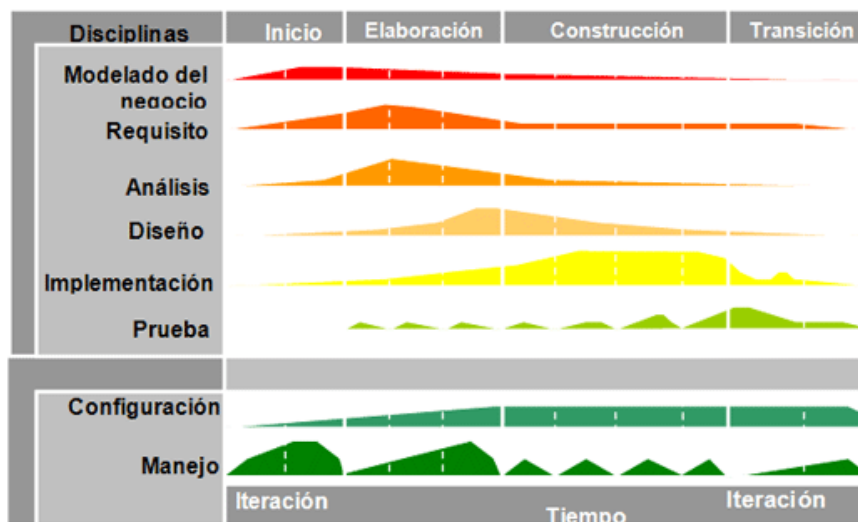


Figura 15: Fases proceso unificado

En el proceso unificado podemos diferenciar cuatro etapas principales que se solapan:

- **Etapas de Análisis:** Donde definimos los requisitos y objetivos que deberá cumplir el proyecto.
- **Etapas de diseño:** Determinaremos como se utilizarán las distintas herramientas que componen el proyecto y el funcionamiento de las mismas con el objetivo de lograr cumplir con los objetivos propuestos en la etapa de análisis.
- **Fase de implementación:** Llevaremos a cabo la implementación del proyecto con el fin de obtener un producto que cumpla con los objetivos y requisitos marcados, siguiendo para ello los pasos establecidos en la fase de diseño.
- **Fase de pruebas:** Realizaremos las pruebas pertinentes para comprobar el correcto funcionamiento de la aplicación.

Para obtener más información sobre este apartado consultar el *Anexo 1: Planificación del proyecto*.

6.2 Planificación temporal del proyecto

La elaboración del proyecto ha sido guiada mediante la implementación de una planificación temporal, la cual define todas las actividades a llevar a cabo durante el desarrollo y la relación entre las mismas.

Esta planificación nos ha llevado a definir una serie de tareas y otorgarle a las mismas unos periodos de trabajo y fechas establecidas para su desarrollo. En la figura 16 vemos el conjunto de tareas definidas en la planificación del proyecto:

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1		Iteración 1	130 días?	mié 01/09/21	sáb 29/01/22		
2		Fase de preparación	24 días?	mié 01/09/21	mar 28/09/21		
3		Necesidades de los usuarios	24 días?	mié 01/09/21	mar 28/09/21		Mario
4		Herramientas de desarrollo	24 días	mié 01/09/21	mar 28/09/21		Mario
5		Fase de análisis	70 días	mié 29/09/21	sáb 18/12/21		
6		Análisis de requisitos	70 días	mié 29/09/21	sáb 18/12/21		Mario
7		Análisis de objetivos funcionales	70 días	mié 29/09/21	sáb 18/12/21		Mario
8		Análisis de objetivos no funcionales	70 días	mié 29/09/21	sáb 18/12/21		Mario
9		Fase de diseño	102 días?	lun 04/10/21	sáb 29/01/22		
10		Diseño del Frontend	102 días?	lun 04/10/21	sáb 29/01/22		Mario
11		Diseño del Backend	102 días?	lun 04/10/21	sáb 29/01/22		Mario
12		Diseño de la base de datos	102 días?	lun 04/10/21	sáb 29/01/22		Mario
13		Fase de desarrollo	96 días	lun 11/10/21	sáb 29/01/22		
14		Desarrollo base de datos	96 días	lun 11/10/21	sáb 29/01/22		
15		Aprendizaje: Typescript, MongoDB, Moongose	96 días	lun 11/10/21	sáb 29/01/22		Mario
16		Gestión de las colecciones	90 días	lun 18/10/21	sáb 29/01/22		Mario
17		Gestión de las consultas	84 días	lun 25/10/21	sáb 29/01/22		Mario
18		Desarrollo Backend	78 días	lun 01/11/21	sáb 29/01/22		
19		Aprendizaje: Node, SocketIO y express	78 días	lun 01/11/21	sáb 29/01/22		Mario
20		Implementación gestión de acceso y cuentas de usuarios	72 días	lun 08/11/21	sáb 29/01/22		Mario
21		Desarrollo Frontend	66 días	lun 15/11/21	sáb 29/01/22		
22		Aprendizaje: Angular e Ionic	66 días	lun 15/11/21	sáb 29/01/22		Mario
23		Implementación gestión de acceso y cuentas de usuario	54 días	lun 29/11/21	sáb 29/01/22		Mario
24		Fase de pruebas	30 días?	lun 27/12/21	sáb 29/01/22		Mario
25		Pruebas base de datos	30 días	lun 27/12/21	sáb 29/01/22		Mario
26		Pruebas gestión de acceso y cuentas de usuario	24 días?	lun 03/01/22	sáb 29/01/22		
27		Hito 1: Desarrollo del acceso y cuentas de usuario	0 días	sáb 29/01/22	sáb 29/01/22	23;20;17;16;1	
28							

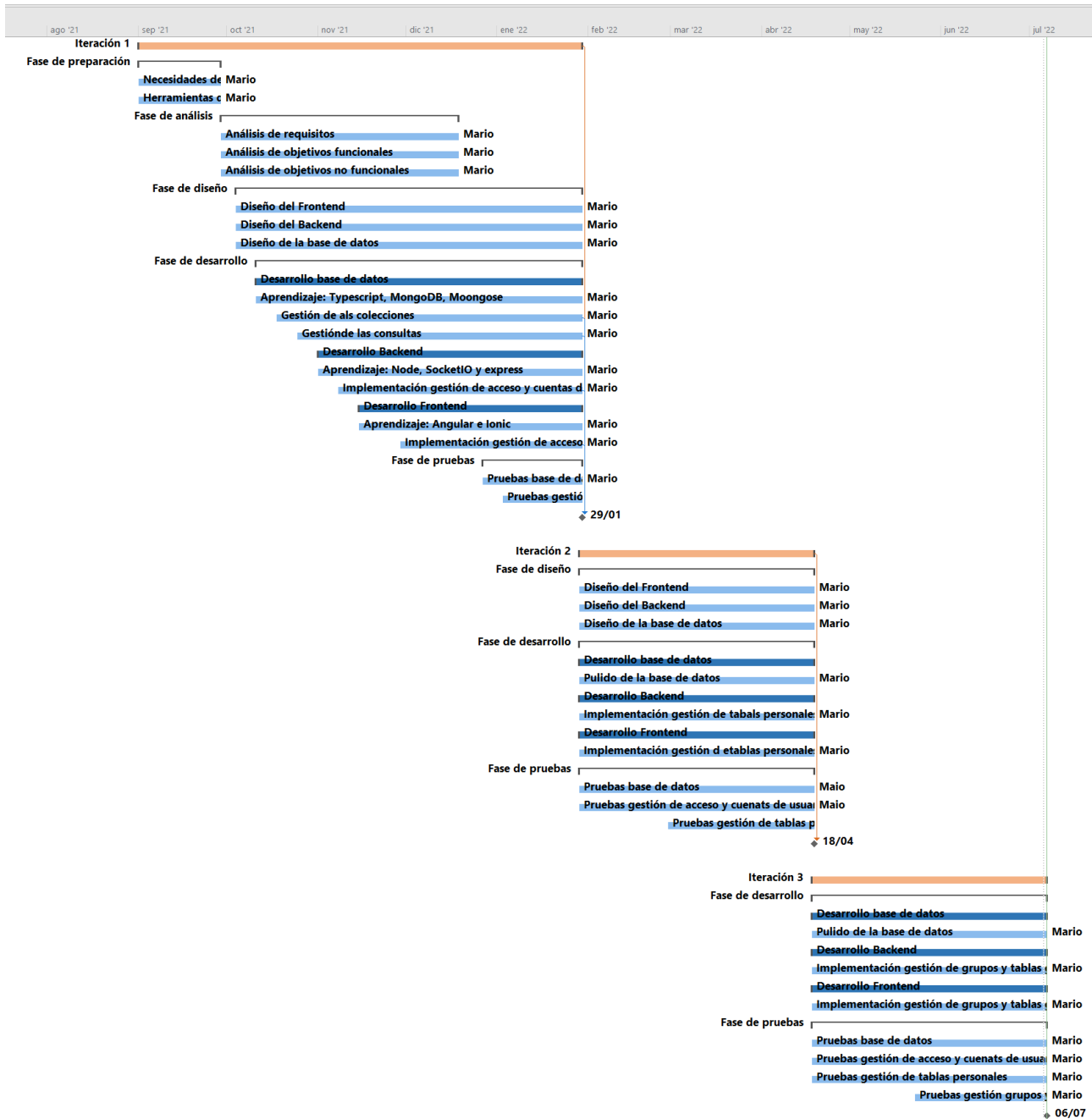
Figura 16: Definición tareas, primera iteración

GroupApp: Aplicación de gestión de eventos y grupos distribuida y centralizada

28							
29	📅	↳ Iteración 2	68 días?	sáb 29/01/22	lun 18/04/22		
30	📅	↳ Fase de diseño	68 días?	sáb 29/01/22	lun 18/04/22		
31	📅	Diseño del Frontend	68 días?	sáb 29/01/22	lun 18/04/22		Mario
32	📅	Diseño del Backend	68 días?	sáb 29/01/22	lun 18/04/22		Mario
33	📅	Diseño de la base de datos	68 días?	sáb 29/01/22	lun 18/04/22		Mario
34	📅	↳ Fase de desarrollo	68 días?	sáb 29/01/22	lun 18/04/22		
35	📅	↳ Desarrollo base de datos	68 días?	sáb 29/01/22	lun 18/04/22		
36	📅	Pulido de la base de datos	68 días?	sáb 29/01/22	lun 18/04/22		Mario
37	📅	↳ Desarrollo Backend	68 días?	sáb 29/01/22	lun 18/04/22		
38	📅	Implementación gestión de tablas personales	68 días?	sáb 29/01/22	lun 18/04/22		Mario
39	📅	↳ Desarrollo Frontend	68 días?	sáb 29/01/22	lun 18/04/22		
40	📅	Implementación gestión de tablas personales	68 días?	sáb 29/01/22	lun 18/04/22		Mario
41	📅	↳ Fase de pruebas	68 días?	sáb 29/01/22	lun 18/04/22		
42	📅	Pruebas base de datos	68 días?	sáb 29/01/22	lun 18/04/22		Maio
43	📅	Pruebas gestión de acceso y cuentas de usuario	68 días?	sáb 29/01/22	lun 18/04/22		Maio
44	📅	Pruebas gestión de tablas personales	43 días?	lun 28/02/22	lun 18/04/22		
45	📅	Hito 2 Desarrollo de las tablas personales	0 días	lun 18/04/22	lun 18/04/22	29	
46							

46							
47	📅	↳ Iteración 3	69 días?	lun 18/04/22	mié 06/07/22		
48	📅	↳ Fase de desarrollo	69 días?	lun 18/04/22	mié 06/07/22		
49	📅	↳ Desarrollo base de datos	69 días?	lun 18/04/22	mié 06/07/22		
50	📅	Pulido de la base de datos	69 días?	lun 18/04/22	mié 06/07/22		Mario
51	📅	↳ Desarrollo Backend	69 días?	lun 18/04/22	mié 06/07/22		
52	📅	Implementación gestión de grupos y tablas grupales	69 días?	lun 18/04/22	mié 06/07/22		Mario
53	📅	↳ Desarrollo Frontend	69 días?	lun 18/04/22	mié 06/07/22		
54	📅	Implementación gestión de grupos y tablas grupales	69 días?	lun 18/04/22	mié 06/07/22		Mario
55	📅	↳ Fase de pruebas	69 días?	lun 18/04/22	mié 06/07/22		
56	📅	Pruebas base de datos	69 días?	lun 18/04/22	mié 06/07/22		Mario
57	📅	Pruebas gestión de acceso y cuentas de usuario	69 días?	lun 18/04/22	mié 06/07/22		Mario
58	📅	Pruebas gestión de tablas personales	69 días?	lun 18/04/22	mié 06/07/22		Mario
59	📅	Pruebas gestión grupos y tablas grupales	39 días?	lun 23/05/22	mié 06/07/22		Mario
60	📅	Hito 3: Desarrollo de grupos y tablas grupales	0 días	mié 06/07/22	mié 06/07/22		

A continuación podemos ver el diagrama de Gantt generado para la planificación del proyecto:



Además de definir las diferentes tareas y la relación entre ellas, la planificación temporal del proyecto también especifica las fechas de comienzo y finalización del proyecto y las horas de trabajo del mismo. Para obtener más información sobre este apartado consultar el *Anexo 1: Planificación del proyecto*.

6.3 Especificación de requisitos

Uno de los apartado llevados a cabo durante la elaboración del proyecto ha sido la especificación de los diferentes requisitos a cumplir por la aplicación. Para realizado esto

se han elaborado una serie de tablas y esquemas que nos permiten conocer cada uno de los distintos objetivos del sistema. Para obtener más información sobre este apartado consultar el *Anexo 2: Especificación de requisitos*.

6.3.1 Participantes del proyecto

Estas tablas informan de aquellos tipos de usuarios que han participado en el desarrollo del proyecto. Participantes definidos en el sistema:

- Equipo de desarrollo.
- Directores del proyecto.

A continuación podemos ver un ejemplo de una de las tablas de participantes:

6.3.2 Objetivos del sistema

Estas tablas tienen el objetivo de especificar aquellos objetivo funcionales que el sistema debe desarrollar para cumplir con su funcionalidad. Objetivos del sistema definidos:

- Gestión de cuentas de usuario.
- Gestión de tablas personales.
- Gestión de grupos y tablas grupales.
- Gestión de búsquedas y descubrimientos.
- Gestión de notificaciones.

A continuación podemos ver un ejemplo de una de las tablas de objetivos del sistema:

Tabla 1: Tabla ejemplo objetivos del sistema

OBJ-001	Gestión de cuentas de usuario
Versión	Cliente: 0.3.1 – Servidor: 0.3.1
Autores	Equipo de desarrollo
Fuentes	Personas anónimas no conocidas por el sistema Usuarios identificados en el sistema
Descripción	El sistema deberá almacenar correctamente la información referente a las cuentas de usuario en el sistema. El tratamiento de la información debe ser versátil y seguro, de modo que el servidor pueda realizar las acciones solicitadas por el usuario tanto de peticiones de datos como de modificación de los mismos. El usuarios anónimos deben poder tener acceso a los datos con el fin de identificarse o registrarse en el sistema. Por otra parte, los usuarios identificados deben poder realizar modificaciones en los datos referentes a su cuenta.
Subobjetivos	Se debe implementar una capa de seguridad que impida el uso inadecuado de los datos sensibles del sistema y de los usuarios, impidiendo que se puedan obtener de forma fraudulenta.
Importancia	Vital

Urgencia	Inmediatamente
Estado	En construcción
Estabilidad	Baja
Comentarios	Ninguno

6.3.3 Requisitos de información

Los requisitos de información son aquellos que especifican el conjunto de datos con los que tiene que interaccionar la aplicación sobre cada uno de los elementos a controlar. Se han definidos los siguientes objetivos de información:

- Credenciales de usuario.
- Perfil de usuario.
- Tablas personales y grupales.
- Grupos.
- Eventos.

A continuación podemos ver un ejemplo de una de las tablas de requisitos de información:

Tabla 2: Ejemplo tabla requisitos de información

IRQ-001	Credenciales de usuario
Versión	Cliente: 0.3.1 – Servidor: 0.3.1
Autores	Equipo de desarrollo.
Fuentes	Usuarios identificados en el sistema
Dependencias	UC-001: Crear una nueva cuenta de usuarios UC-002: Acceder al sistema UC-003: Eliminar cuenta de usuarios UC-004: Cerrar sesión UC-005: Modificar información de la cuenta
Descripción	El sistema deberá almacenar toda la información referente a las credenciales de los usuarios necesarias para que estos puedan autenticarse en la aplicación
Datos específicos	ID de usuario Email
Importancia	vital
Urgencia	inmediatamente
Estado	En desarrollo
Estabilidad	baja
Comentarios	Ninguno

6.3.4 Requisitos no funcionales

Los requisitos no funcionales son aquellos que especifican metas que están relacionadas con los estándares que debe cumplir el proyecto pero sin registrar directamente la funcionalidad que este debe seguir. Se han definido los siguientes requisitos no funcionales:

- Portabilidad.
- Seguridad y protección de datos.
- Usabilidad.
- Funcionamiento en tiempo real.

6.3.5 Requisitos funcionales

Estos requisitos especifican cómo se debe comportar el sistema ante una de las posibles acciones llevadas a cabo por el usuario. Están directamente relacionados con la funcionalidad de la aplicación y para tener una concepción general se han realizado diagramas de casos de uso, los cuales son:

- Diagrama de gestión de cuentas de usuario.
- Diagrama de gestión de tablas.
- Diagrama de gestión de grupos.
- Diagrama de gestión de búsquedas y descubrimientos.

A continuación, en la figura 17 podemos ver un ejemplo de uno de estos diagramas:

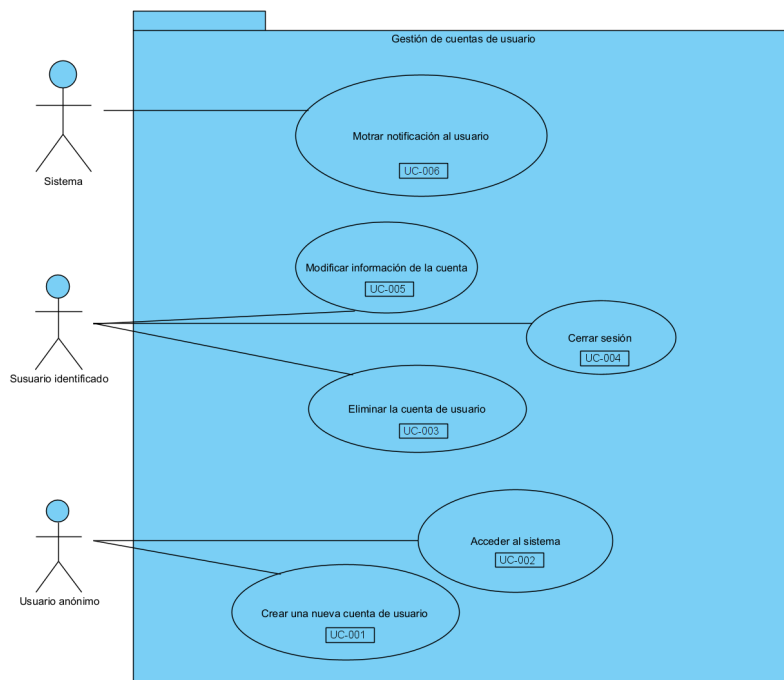


Figura 17: Gestión de cuentas de usuario

como podemos ver en la imagen, estos diagramas están compuestos por dos elementos: Actores, los cuales realizan la acción y los casos de uso, los cuales definen el proceso para realizar dicha acción.

Ambos de estos elementos están definidos mediante tablas. A continuación podemos ver un ejemplo de tabla de actor:

Por otra parte, la siguiente tabla es un ejemplo de las tablas de descripción de casos de uso:

UC-008: Modificar una tabla

UC-008	Modificar una tabla
Versión	Cliente: 0.3.1 – Servidor: 0.3.1
Autores	Equipo de desarrollo
Fuentes	Usuario identificado
Dependencias	Ninguno
Descripción	El sistema deberá comportarse tal y como se describe en el presente caso de uso cuando un usuario identificado quiera modificar los datos asociados a una tabla, ya sea personal o grupal.
Precondición	El usuario debe estar identificado
Secuencia normal	<ol style="list-style-type: none"> 1. El actor solicita la modificación de los datos de una tabla concreta, ya sea personal o grupal. El caso de uso comienza. 2. El sistema solicita al actor que realice los cambios que indique los cambios que quiere realizar. 3. El actor modifica los datos deseados. 4. El sistema verifica los los cambios indicados por el actor y los actualiza. El caso de uso finaliza.
Postcondición	El sistema ha modificado la tabla solicitada por el cliente, ya sea personal o asociada a un grupo concreto.
Excepciones	<ul style="list-style-type: none"> • Paso 3: Si el actor indica al sistema que no quiere actualizar los datos, el caso de uso finaliza. • Paso 4: Si alguno de los campos introducidos por el actor es erróneo, el sistema pedirá de nuevo el ingreso de los datos.
Rendimiento	<ol style="list-style-type: none"> 1. Inmediato 2. Inmediato 3. Tiempo requerido por el actor 4. Inmediato
Frecuencia esperada	Baja
Importancia	vital
Urgencia	hay presión

Estado	Pendiente de validación
Estabilidad	Alta
Comentarios	Ninguno

6.4 Análisis de requisitos

Otra de los elementos llevados a cabo durante el desarrollo ha sido el análisis de los requisitos detallados anteriormente, con el fin de determinar las acciones llevadas a cabo por el sistema para poder cumplirlos. Para obtener más información sobre este apartado consultar el *Anexo 3: Análisis de requisitos*.

6.4.1 Modelo de dominio

Para poder realizar el análisis de los requisitos lo primero que hemos desarrollado ha sido un modelo de dominio, el cual nos indica la relación entre las distintas clases del sistema y especifica los requisitos de información de las mismas.

A continuación, en la figura 18 podemos ver una imagen del modelo de dominio desarrollado:

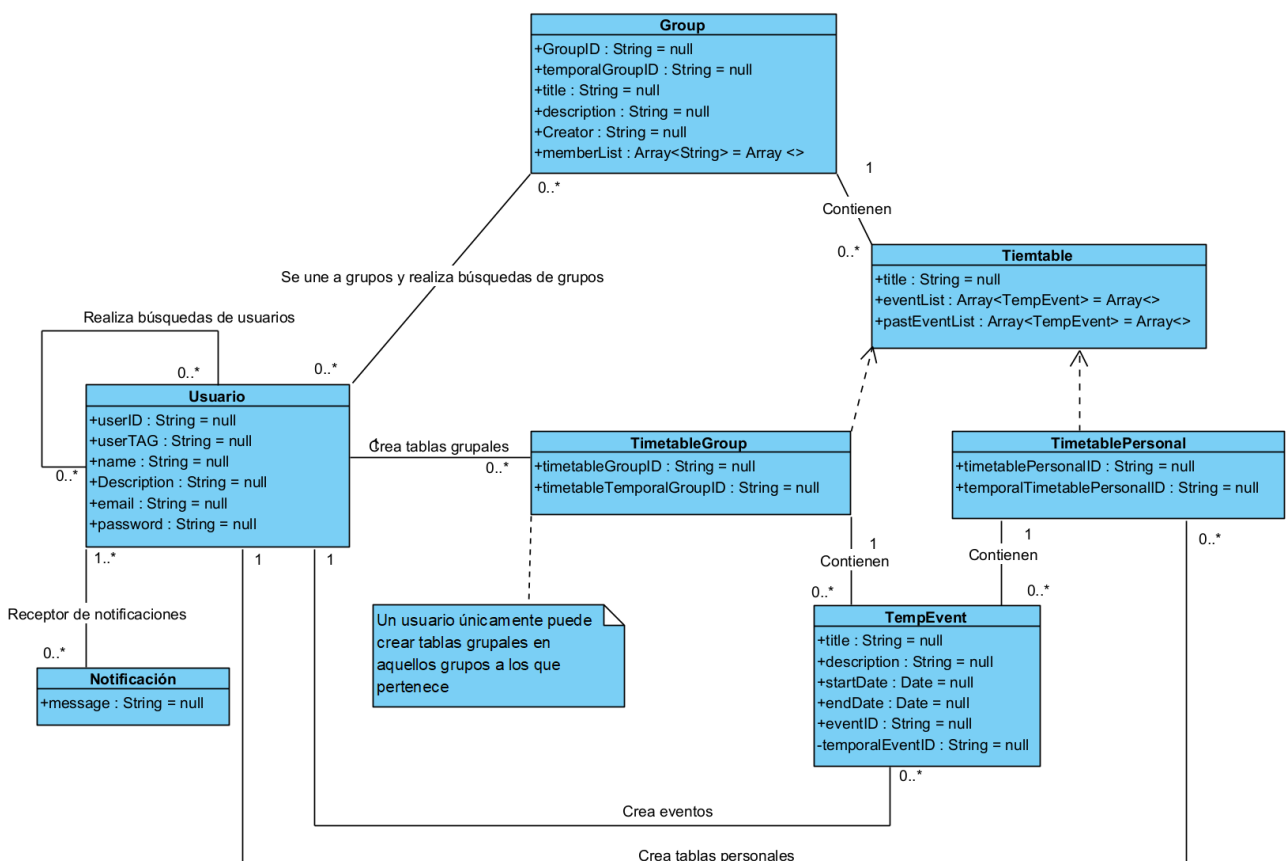


Figura 18: Ejemplo modelo de dominio

6.4.2 Paquetes y clases de análisis

A parte del modelo de dominio se han desarrollado un diagrama de interacción que relaciona los distintos componentes que conforman el sistema, ya sean de interfaz, control o información, en base a como interaccionan entre ellos. Estos son los mismos componentes que se espesarán más adelante en la vista de interacción,

A continuación, en las figuras 19 y 20 podemos ver el diagrama del paquete de análisis un ejemplo del contenido de una de sus clases, respectivamente:

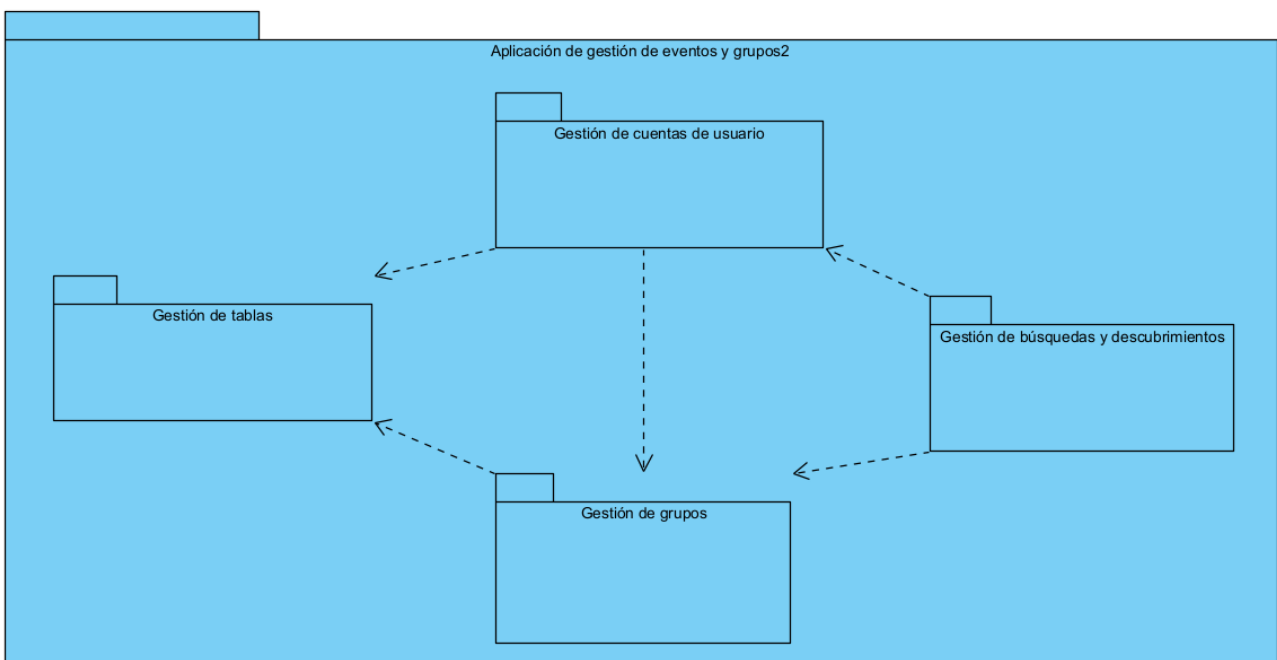


Figura 19: Ejemplo Diagrama paquete análisis

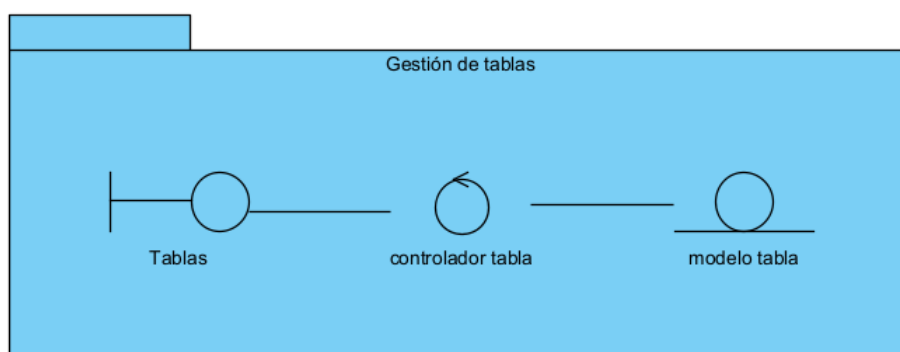


Figura 20: Ejemplo clase de análisis

6.4.3 Vista de interacción

La vista de interacción expone, de una forma más específica que en las tablas, las acciones a realizar entre el usuario y cada uno de los componentes del sistema que intervienen, para llevar a cabo los casos de uso especificados en el Anexo 2.

Estos diagramas también tienen el objetivo de especificar el número de componentes que conformarán el sistema. Dichos componentes son los mismos que se expresan en cada una de las clases de análisis.

A continuación, en la figura 21 podemos ver el ejemplo de uno de estos diagramas:

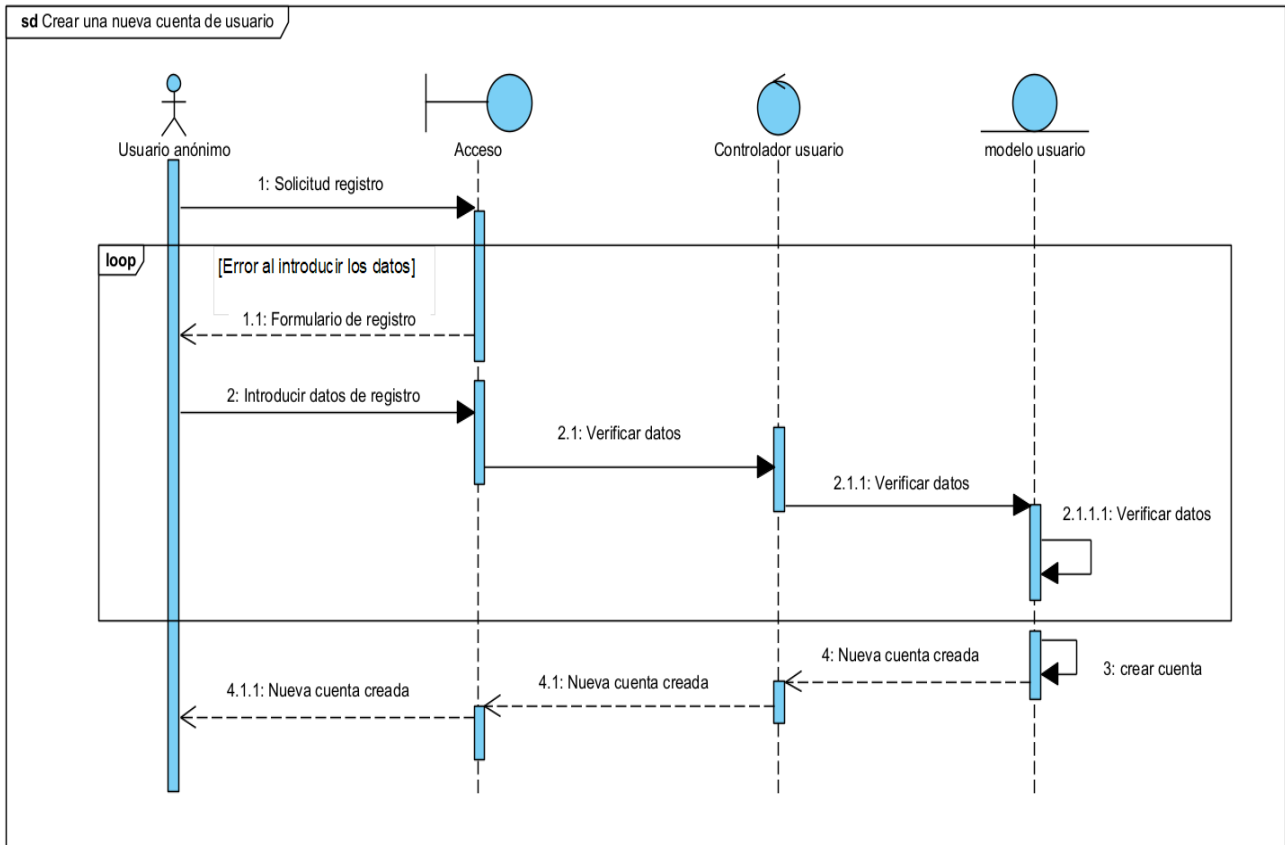


Figura 21: Ejemplo diagrama de interacción

6.4.4 Vista arquitectónica

La vista arquitectónica recoge todas los componentes especificados tanto en la vista de interacción como en las clases de análisis y especifica sus interacciones siguiendo el diseño marcado por el diagrama de análisis. Esto sirve para hacer comprender al usuario las relación que se crearán entre los distintos componentes que conforman el sistema.

6.4 Diseño del sistema

Durante el desarrollo hemos realizado una fase de diseño, la cual se basa en especificar el conjunto de clases que conformarán cada uno de los componentes especificados en la sección de análisis. Esta sección se basa en proporcionar un modelo muy parecido a lo que sería una implementación real, especificando las diversas clases con funciones y atributos. Para más información pueden recurrir al *Anexo 4- Especificación de diseño*.

6.4.1 Patrones empleados

El desarrollo de la aplicación se ha hecho en base a el patrón de componentes especificado por *Angular*, ya que es el *Framework* con el que trabajamos. Anteriormente *Angular* implementaba un patrón Modelo – Vista – Controlador clásico, sin embargo, desde la salida de *Angular v2* el patrón arquitectónico llevado a cabo fue el patrón por componentes.

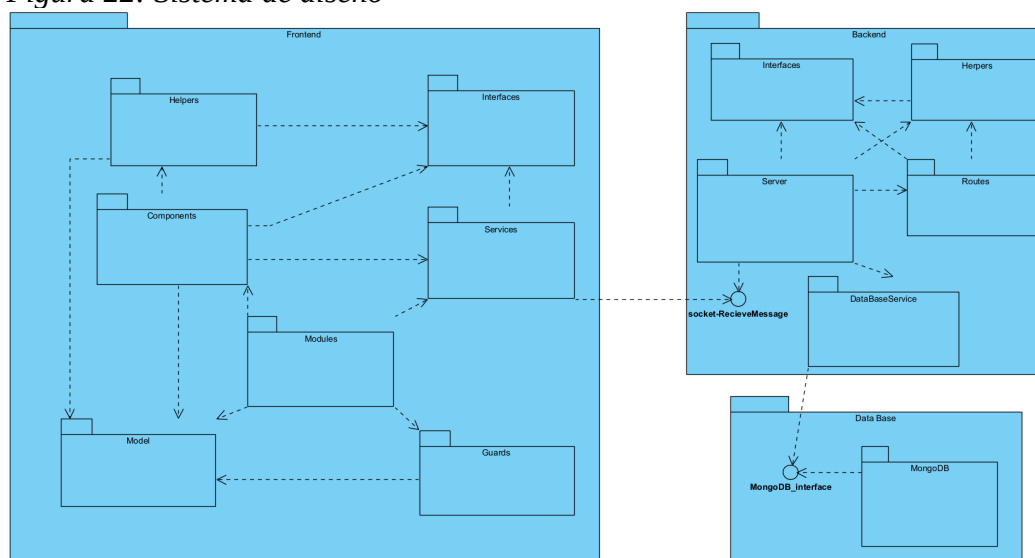
En este patrón se basa en que cada componente posee su propia vista y su propio controlador, y dos componentes diferentes pueden relacionarse entre si a través de sus vistas (mediante el uso del código HTML) o importándose en el constructor de sus controladores (mediante el uso de *Typescript*).

Además de esto, la gestión de datos del modelo también emplea el uso del patrón *Singleton*, con el objetivo de tener un modelo de datos común en toda la aplicación.

6.4.2 Sistemas y subsistemas de diseño

El sistema de diseño realiza una división del sistema en distintos subsistemas que interaccionan entre sí, los cuales contienen a su vez las clases de diseño. En la figura 22 poderemos ver el sistema de diseño desarrollado en la aplicación.

Figura 22: Sistema de diseño



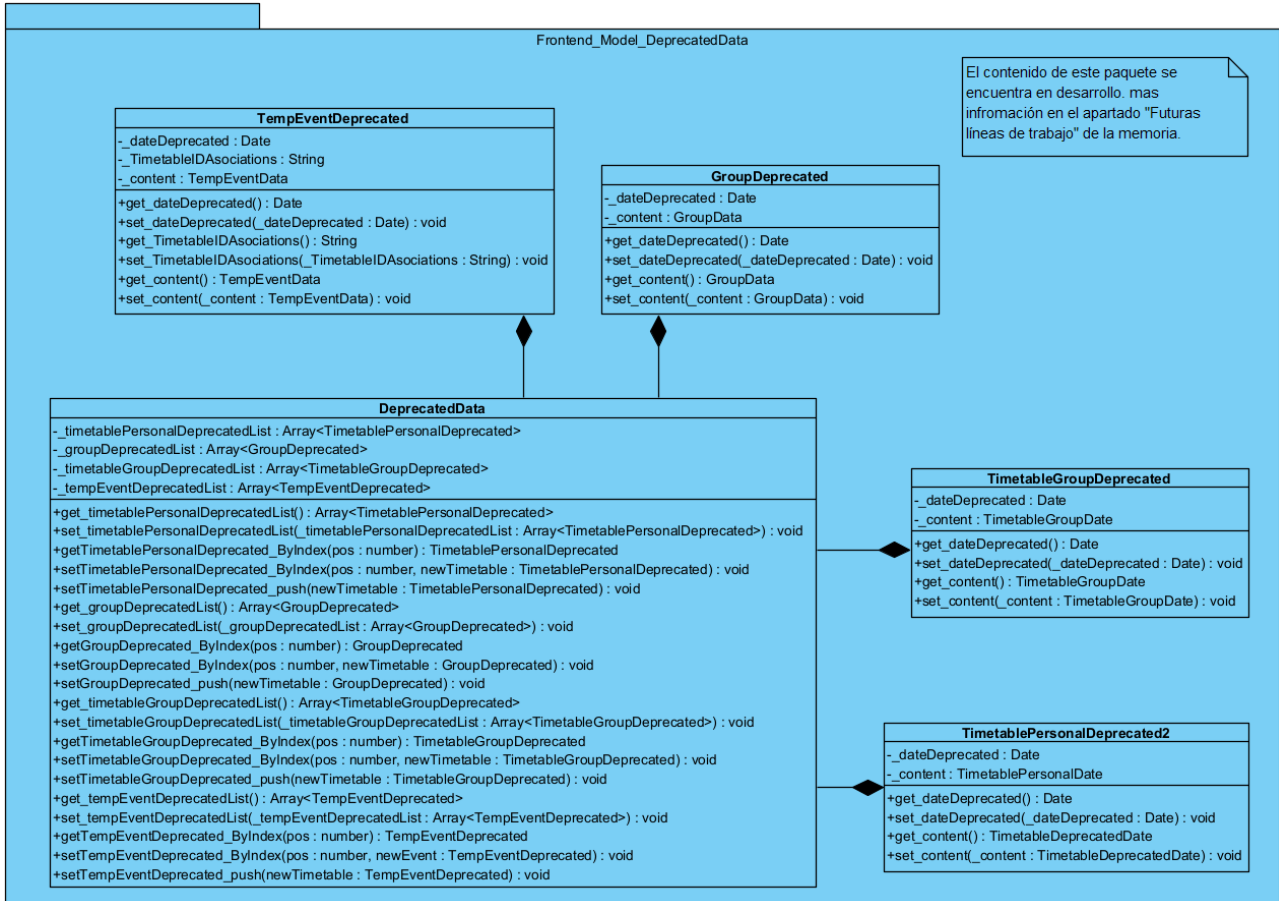
6.4.3 Clases de diseño

GroupApp: Aplicación de gestión de eventos y grupos distribuida y centralizada

En esta sección se ha especificado cada una de las clases que conforman los paquetes definidos en el apartado anterior. Dentro de cada una de estas clases se han especificados tanto sus atributos principales como las funciones clave para desarrollar sus funcionalidad.

A continuación, en la figura 23, podemos ver una de las imágenes que define las respectivas clases de una paquete perteneciente al *Frontend*:

Figura 23: Subsistema *Frontend_Model_DeprecatedData*



6.4.4 Modelo de diseño

En el modelo de diseño se ha llevado a cabo al especificación de cada uno de los casos de uso mediante la utilización de uno de los tres siguientes tipos de diagramas:

- Diagramas de secuencia.
- Diagramas de máquinas de estado.
- Diagramas de actividad.

Al contrario que pasaba en la sección de Análisis de requisitos, estos diagramas ya están desarrollados en base a las clases definidas en el el diseño del sistema, lo cual ahce que sean diagramas muy próximos a la implementación real.

En la figura 24 y 25 podemos ver un ejemplo de los diagramas de secuencia y de actividad mencionados anteriormente:

Figura 24: Ejemplo diagrama de secuencia

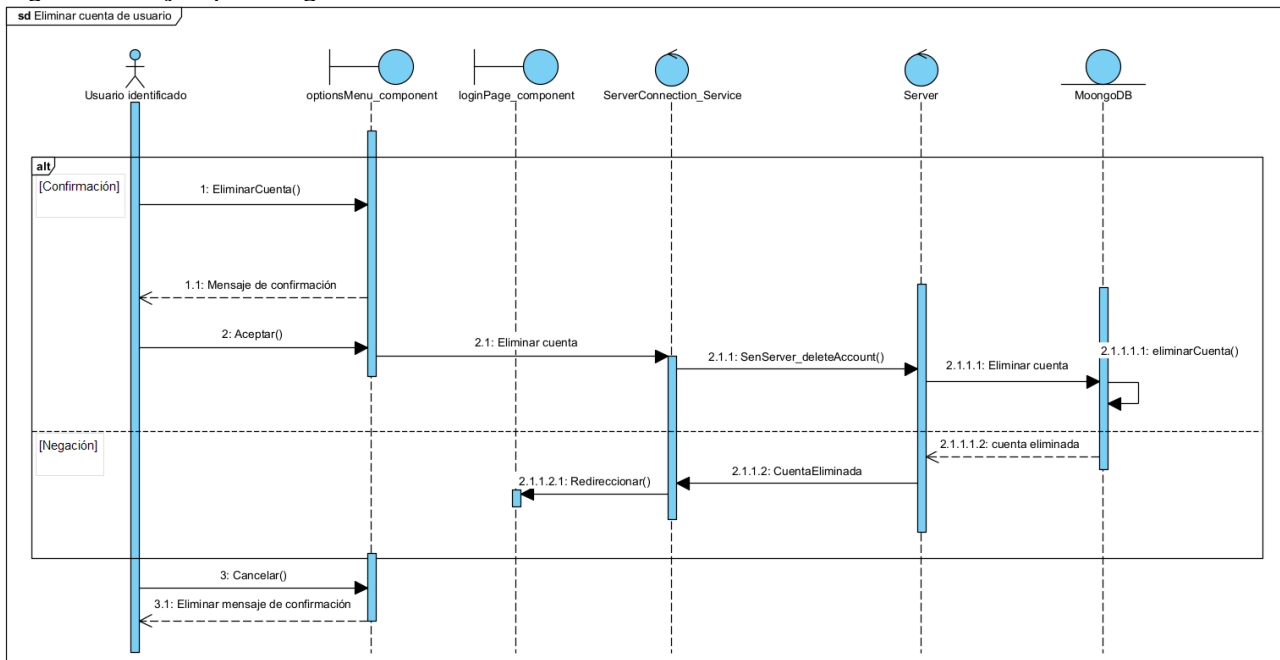
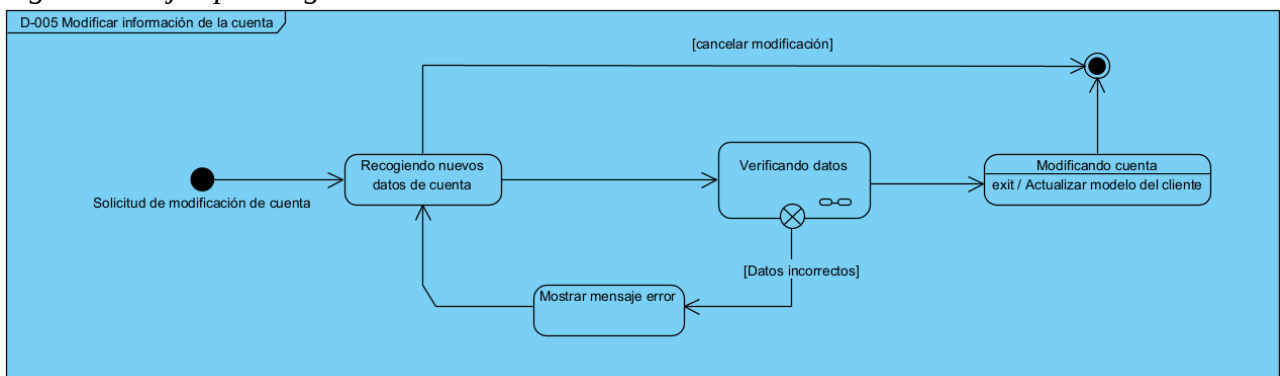


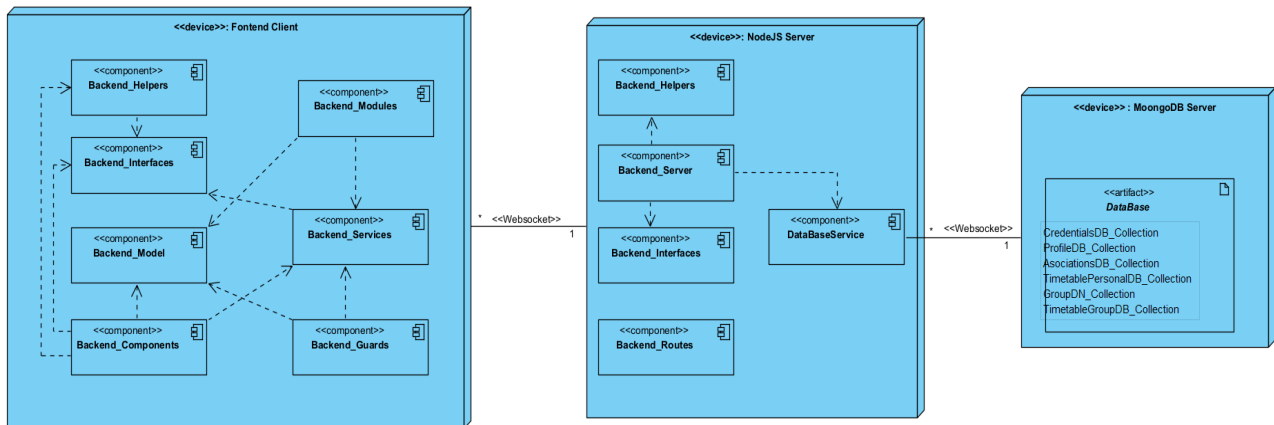
Figura 25: Ejemplo diagrama de estados



6.4.5 Modelo de despliegue

El modelo de despliegue explica todos los subsistemas que componen el sistema y la relación entre los mismos. Este modelo se divide en dispositivos, los cuales están relacionados entre sí y a su vez de componen de los componentes especificados en el sistema de diseño. En la figura 26 imagen podemos ver el modelo de despliegue de la aplicación:

Figura 26: Ejemplo modelo de despliegue



Como podemos ver, el modelo se compone de tres dispositivos:

- **Dispositivo Frontend Client:** Este dispositivo es el encargado de gestionar la totalidad de la interacción con el usuario y la presentación de la información pro pantalla. Se trata de un dispositivo genérico debido a que la parte *Frontend* es una aplicación híbrida, de modo que puede ser ejecutada en una gran diversidad de dispositivos.
- **Dispositivo NodeJS Server:** Dispositivo encargado de gestionar los mensajes recibidos del *Frontend*, tratar con al información sensible de la aplicación y los usuarios y ponerse en contacto con la base de datos.
- **Dispositivo MongoDB Server:** Se trata de un dispositivo que encapsula la base de datos que gestiona toda la información sensible de la aplicación. Esta se pone en contacto únicamente con el servidor para recibir solicitudes de iteraciones y devolver los resultados.

6.5 Momentos destacables del desarrollo

6.5.1 Estructura y tipo de proyecto

Corrían las primeras etapas del presente curso cuando nos topamos con esta decisión y pese a estar los objetivos principales ya establecidos, decantarnos por un tipo concreto de proyecto, con el cual definir también la estructura del mismo, no era sencillo.

Únicamente había dos requisitos clave que tenía que cumplir el tipo de proyecto: Las herramientas utilizadas debían contar con una interfaz gráfica para su interacción con el usuario y la información del usuario debía estar desligada del dispositivo utilizado por el mismo.

La segunda de las opciones nos llevo a decidir implementar un sistema distribuido centralizado, de modo que los datos de todos los usuarios se guarden a buen recaudo en una base de datos a la cual se acceda mediante el uso de un servidor.

El primero de los requisitos no era tan sencillo debido por un lado a la gran diversidad de lenguajes que integran una interfaz gráfica de forma nativa y por el otro a la existencia de muchas herramientas que proporcionan dicha interfaz de forma compatible con lenguajes que no la tienen integrada.

Después de estar buscando entre varios lenguajes como *Python* o herramientas como *UML* llegamos a la conclusión de que la mejor opción era utilizar un *framework* de desarrollo web. Esto trae consigo muchos puntos a favor aplicados a nuestro proyecto:

- Se cumplen con los dos requisitos iniciales a la hora de decidir el tipo y estructura del proyecto.
- Las aplicaciones web son versátiles gracias a que los dispositivos multimedia cada vez son más comunes y cuentan con acceso a internet.
- Están pensadas para aplicaciones y herramientas con poca demanda de recursos, como la nuestra.
- Acceder a la misma es muy cómodo, puesto que no se necesita ningún tipo de instalación. Esto hace que los usuarios estén más dispuestos a empezar a utilizarla.
- Las herramientas de diseño web están preparadas para ser utilizadas con una arquitectura cliente-servidor gracias a que es su uso más utilizado. Esto facilita el proceso de adaptación al uso de la herramienta y aumenta la disponibilidad de bibliotecas de terceros.

Sin embargo, pese a todos los puntos positivos que trae consigo el uso de herramientas de desarrollo web, mi desconocimiento total sobre el área y las herramientas que se utilizan sobre la misma constituían un aspecto negativo muy importante a tener en cuenta.

Pese a esto, finalmente decidimos desarrollar el proyecto bajo este enfoque y utilizando una gran variedad de herramientas web. Una de las motivaciones principales para decantarnos por esta opción fue la gran importancia actualmente que tiene el sector del desarrollo web y su popular uso dentro de las empresas actuales.

Posteriormente, debido al crecimiento de la sección encargada de la interacción directa con la base de datos, se decidió encapsular su funcionalidad dentro de un paquete propio de NPM. Finalmente se trataría de una aplicación cliente-servidor construida por tres secciones o subproyectos:

- GroupApp-client: Encargado de realizar toda la interacción con el usuario y presentar la información necesaria para que este pueda llevar a cabo las acciones que quiere correctamente. Esta sección no cuenta con dependencias sobre el tratamiento de los datos.
- GroupApp-server: Se comunica con el anterior mediante paso directo de mensajes y es el encargado tanto de realizar el tratamiento de los datos antes de operar con la base de datos como solicitar extracciones a la misma que posteriormente serían tratadas y enviadas al cliente o frontend.

- *DataBaseAccess*: Encargada de gestionar todo el acceso con la base de datos de *MangoDB*. Esto incluye elementos como definir los esquemas de los documentos a guardar o encapsular todas las solicitudes a la base de datos.

En la figura 27 podemos ver una sistematización de las interacciones entre los subproyectos definidos anteriormente.

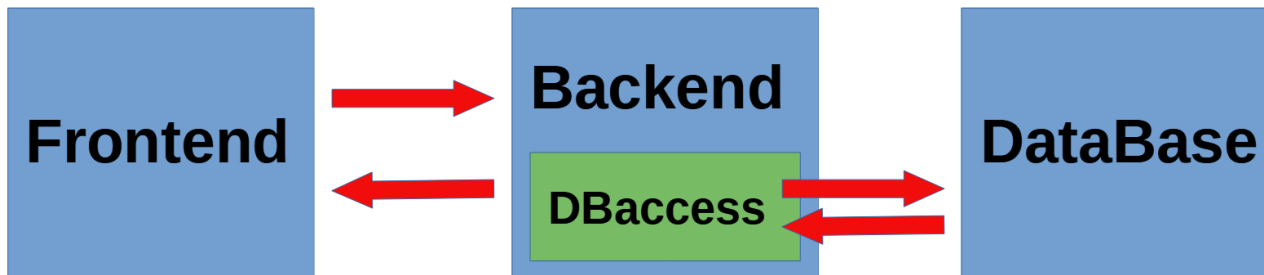


Figura 27: Estructura del proyecto 2

6.5.2 Implementación de Ionic

Nos encontrábamos en la mitad del proceso de desarrollo cuando nos dimos cuenta de un gran percance: La aplicación que estábamos desarrollando tenía una dirección mucho más enfocada a su uso en dispositivos móviles que en equipos de sobremesa o portátiles.

En teoría no deberíamos tener ningún problema debido a que *Angular* es un *framework* de desarrollo web, por lo que su uso puede realizarse desde cualquier dispositivo con conexión a internet. Sin embargo, la demanda del público no es que la aplicación pueda usarse desde un navegador, sino que esta pueda ser descargada en el dispositivo desde las *Apps Stores* y usarse sin tener que pasar por ningún motor de búsqueda.

Aunque existen varias opciones de compiladores que nos pueden generar de forma automática una *pwa* de nuestro proyecto. Sin embargo, la solución al problema fue el uso de *Ionic*, el cual podía ser incluido en el proyecto respetando el *framework* de *Angular* y nos proporcionaba la capacidad de mantener un estilo cohesivo y normalizado.

Sin embargo, trabajar con *Ionic* implementa una capa de dificultad muy extensa debido a la gran cantidad de componentes que nos proporciona y los cuales son necesarios para poder mantener un estilo normalizado tanto en dispositivos móviles *IOS* como *Android*.

Aunque el proyecto se ha desarrollado en base al marco de trabajo *Angular*, la inclusión de *Ionic* provoca que más del 90% de los elementos declarados en los ficheros *HTML* tengan que ser sustituidos por elementos proporcionados por este último.

Se trataba de una decisión difícil, la aplicación ya contaba con la sección de acceso y todas las verificaciones implementadas al completo, y añadir *Ionic* pasaba por volver a empezar todo el *frontend* desde cero. Pese a esto, decidimos dar el paso y volver a comentar la construcción del cliente.

Gracias al instaurar el uso de *Ionic* dentro del proyecto contamos con un *fontend* mucho más versátil y adaptable a las distintas plataformas, un estilo mucho más reconocible por los usuarios y una mayor ventana de acceso gracias a la facilidad y comodidad a la hora de acceder al mismo por parte de los clientes.

6.5.3 Creación del paquete DataBaseAccess

El acceso y la gestión de la base de datos es un elemento muy importante dentro del proyecto, no solo se trata del núcleo de interacción con la información sensible que maneja el servidor, sino que también define la eficiencia a la hora de dar resolución a las peticiones de los usuarios.

El uso de *MongoDB* como base de datos y su interacción mediante *mongoose* hace que dentro del servidor se tenga que construir un servicio que contenga una gran estructura de ficheros destinados tanto a la definición de la estructura de la base de datos como a desarrollar todas las posibles interacciones con la misma.

En este momento nos encontramos con una clase principal que utilizaba todos estos ficheros y donde se definían mas de una treintena de funciones publicas. Contar con una sección tan grande que pedía a gritos poder ser encapsulada y dividida para su mejor tratamiento nos llevo a convertirla en un proyecto a parte.

El *backend* se encuentra construido sobre *node*, por lo que la opción más lógica era utilizar el propio gestor de paquetes *NPM (Node Package Manager)* para implementar esta nueva parte del proyecto como un paquete independiente.

Cuando realizamos la instalación del paquete mediante *NPM*, este se guardará dentro de la carpeta *node_modules* del proyecto. En la figura 28 podemos ver la referencia al mismo que se crea automáticamente dentro del fichero de configuración.

```
"databaseservice-groupaplibrary": {
  "version": "0.4.6",
  "resolved": "https://registry.npmjs.org/databaseservice-groupaplibrary/-/databaseservice-groupaplibrary-0.4.6.tgz",
  "integrity": "sha512-EHsiUE55ITmSGUC7nk4e8MF/AufqJ6kUE1xHFew1Fm+/WfN00jcXubfXUMgFZ5vTLqyEYcI9+qQwD1+Gdk2+Kw==",
  "requires": {
    "mongoose": "^6.2.11",
    "nodejs": "^0.0.0",
    "typescript": "^4.6.3"
  }
},
```

Figura 28: Contenido Package.json

Si exploramos el contenido del proyecto podemos ver que este se divide en dos carpetas principales con las correspondientes subcarpetas:

- **DataBase:** Contiene los ficheros principales para dar estructura a la base de datos y definir las interacciones realizadas con la misma. Se divide en tres subcarpetas:
 - **ControllersDB:** Se trata de la definición de funciones que realizan operaciones sobre la propia base de datos pero no sobre los datos en sí.
 - **OperationsDB:** Define las funciones que llevan a cabo las distintas operaciones que interaccionan con la información almacenada en la base de

datos. Estas funciones se encuentran organizadas según la colección con la que operan.

- **SchemasDB:** Contienen la implementación del formato de los documentos que conformarán la base de datos. La estructura de estos datos debe ser definida previamente en una interfaz.
- **Interfaces:** Carpeta que recoge la definición de todas las interfaces utilizadas por el paquete *DataBaseAccess*, tanto las privadas como las que son públicas y pueden utilizarse desde otros proyectos. Se divide en dos subcarpetas:
 - **OperationResult-Interfaces:** Contiene aquellas interfaces que se devuelven al servidor cuando este realiza una petición o modificación en la base de datos a mediante el uso del paquete.
 - **Schemas-interfaces:** Estas interfaces definen la estructura de los documentos que se van a implementar en la base de datos.

Debido a que el gestor de paquete requiere que el paquete que hemos creado sea publicado en los repositorios oficiales, decidimos crear un perfil en la web de *NPM* y subir el paquete de forma pública. En este [enlace](#) podemos acceder al repositorio oficial. En la figura 29 podemos ver el contenido del registro de nuestro paquete en *NPM*:

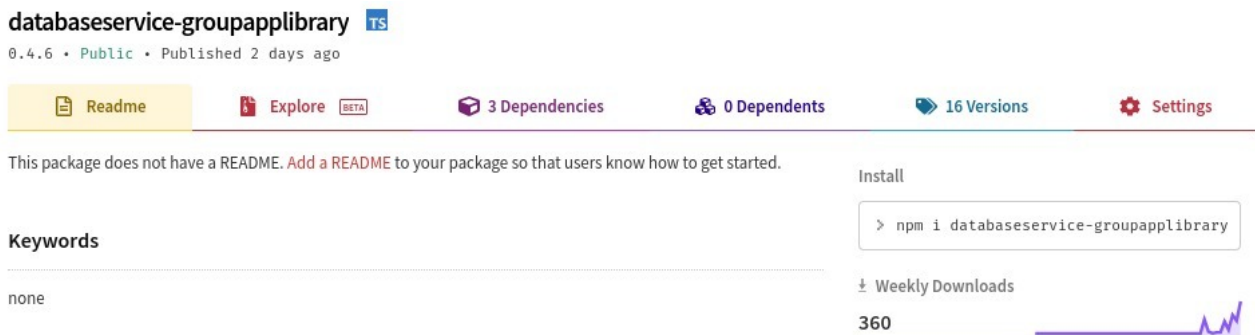


Figura 29: Repositorio DataBaseService

6.5.4 Implementación de la consistencia de datos

Uno de los problemas que trae consigo diseñar una aplicación web es mantener una consistencia de los datos que el cliente obtiene desde el servidor, cuando se ha llevado a cabo un paso previo que requiere la verificación de credenciales.

Esto se debe a que cuando se realiza cualquier acción que provoque el cambio de la dirección *URL* dentro de la aplicación o el refresco de la misma, los datos utilizados por esta se borrarán de forma automática, puesto que no se encuentran almacenados de forma consistente.

Debido a que nuestro proyecto presenta la necesidad de realizar una acreditación de credenciales, no podemos únicamente guardar los datos recogidos en el *Local Storage* de nuestro navegador, pues esto trae consigo una brecha de seguridad. Esto nos obliga a realizar una petición de datos al servidor cada vez que se den estas situaciones.

Para poder hacer las peticiones sin la necesidad de que el usuario vuelva a ingresar sus credenciales, necesitamos implementar el uso de *Jason Web Tokens*. [Aquí](#) podréis acceder al sitio web oficial donde encontrareis toda la documentación necesaria para poder utilizar la biblioteca.

JWT se basa en un concepto muy sencillo, poder evaluar que las credenciales que el usuario me proporciona son las correctas sin las necesidad de que el usuario tenga que ingresarlas directamente, gracias al uso del almacenamiento local del navegador.

Para evitar que se produzcan problemas de seguridad que provocaría tener guardadas las credenciales del usuario en bruto dentro del navegador, lo que se almacena es el propio *JWT*, el cual ha sido creado por el servidor. Se trata de una cadena de información encriptada que contiene, entre otras, cosas las credenciales y que se renueva con cada nuevo o refresco de la aplicación por parte del cliente. En las figuras 30 y 31 podemos ver las partes que constituyen el *JWT* y el esquema de transmisión de mensajes para renovarlo, respectivamente.

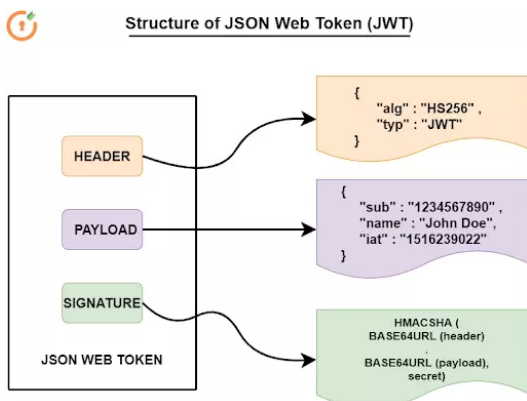


Figura 30: Funcionamiento JWT 1

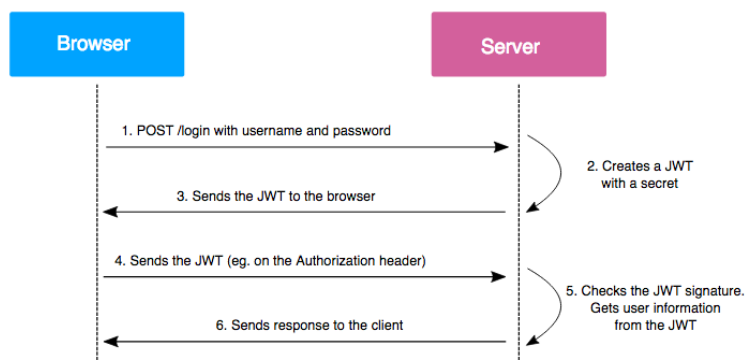


Figura 31: Funcionamiento JWT 2

Para hacer que el envío del *JWT* sea automático cada vez que el usuario va a refrescar la aplicación o navegar dentro de las *URLs* de la misma, el *framework* de *Angular* nos proporciona una solución mediante el uso de los llamados *Guards*. Estos no son más que fragmentos de código que se ejecutan en estos momentos clave y que pueden determinar la redirección llevada a cabo por el usuario dentro de la aplicación.

En la figura 32 podemos ver el código que se ejecuta cuando se disparan estos *Guards*, en el que podemos diferenciar tanto el envío del *JWT* al servidor como la actualización del modelo de datos del cliente en base a los datos recibidos.

```
30     canActivate(): Observable<boolean> | boolean | UrlTree {
31
32         return this.serverConnectionService.renewJWT()
33         .pipe(
34             tap( (res:renewJWTResult Interface) =>{
35                 if(res.status==false){
36                     this.router.navigateByUrl("access");
37                 }else{
38
39                     if(Model.getInitialLoad()==false){
40                         this.model.setUserData( UserData_ensambleFromSD(res.userSD) );
41                         Model.setInitialLoad(true);
42                     }
43                     if(Model.getInitialUpdate()==false){
44                         updateInformation();
45                         Model.setInitialUpdate(true);
46                     }
47                 }
48             }
49         ),
50         map( (res:renewJWTResult Interface)=>{
51             if(res.status==false){
52                 return false;
53             }else{
54                 return true;
55             }
56         }
57     ))
58 );
59
60 }
```

Figura 32: Ejemplo código Validar JWT

6.5.5 Creación de tres modelos de datos distintos

Puede sonar contradictorio que en una aplicación se necesiten un total de tres modelos de datos distintos, sin embargo, esto es necesario para el correcto funcionamiento del proyecto y evitar a su vez problemas de seguridad.

Antes de plantear crear estos modelos, el proyecto funcionaba únicamente con el modelo de datos que permanecía almacenado en la base de datos, de modo que estos viajaban integralmente desde dichos servidores hasta el cliente.

En este momento se propuso usar un modelo de datos distinto para el cliente y para el servidor, de modo que únicamente terminara en manos de los usuarios aquella información que no fuera sensible.

Sin embargo, debido a que el cliente debe realizar distintas transformaciones sobre los datos para facilitar la representación de los mismos y la interacción con el usuario, deducimos que sería necesaria crear un tercer modelo. Este último debía ser común tanto para el cliente como para el servidor y determinaría la estructura que mantendrían los datos cuando estos son transmitidos entre ambas partes.

Los tres modelos de datos que encontramos en el proyecto son:

- **Modelo DB:** Se trata del modelo de datos que se encuentra almacenado en la Base de Datos. El servidor trabaja con este modelo, puesto que no constituye una brecha de seguridad que el *backend* trate con los datos en bruto.
- **Modelo SD:** Modelo encargado de establecer la estructura que conformarán los datos cuando se produce un envío de información entre el cliente y el servidor.
- **Modelo Data:** Modelo de datos con el que trabaja el cliente, el cual se encuentra adaptado a la correcta representación de los datos y la interacción con el usuario.

En las figuras 33, 34 y 35 podemos ver una representación conceptual de los modelos de datos especificados anteriormente.

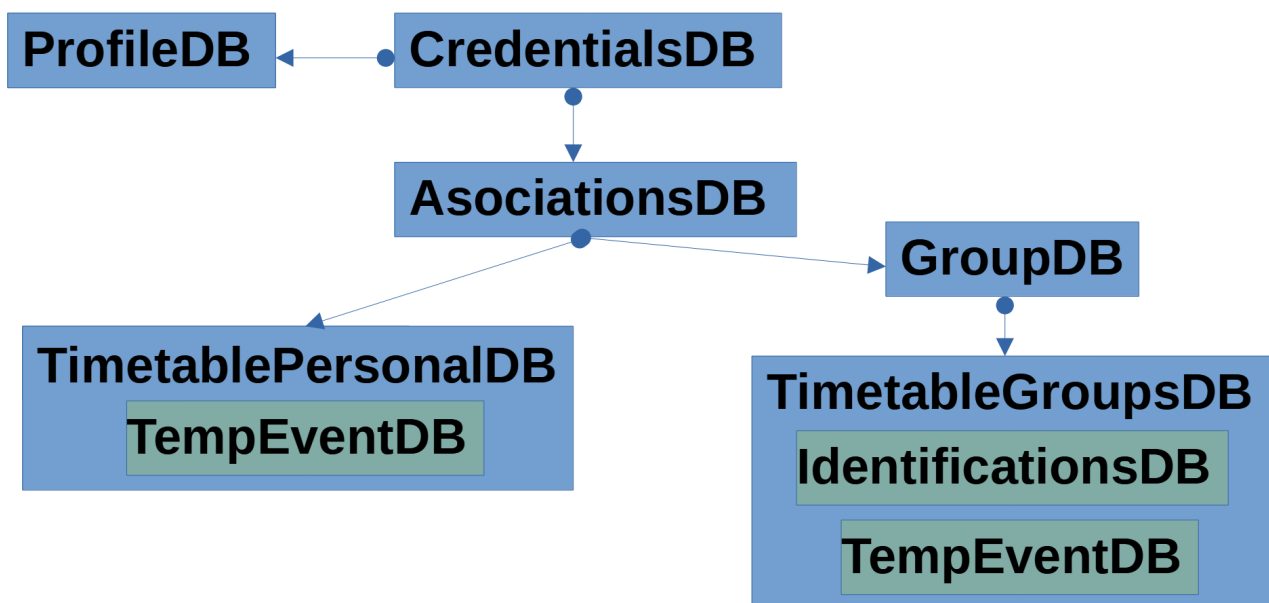


Figura 33: Modelo DB

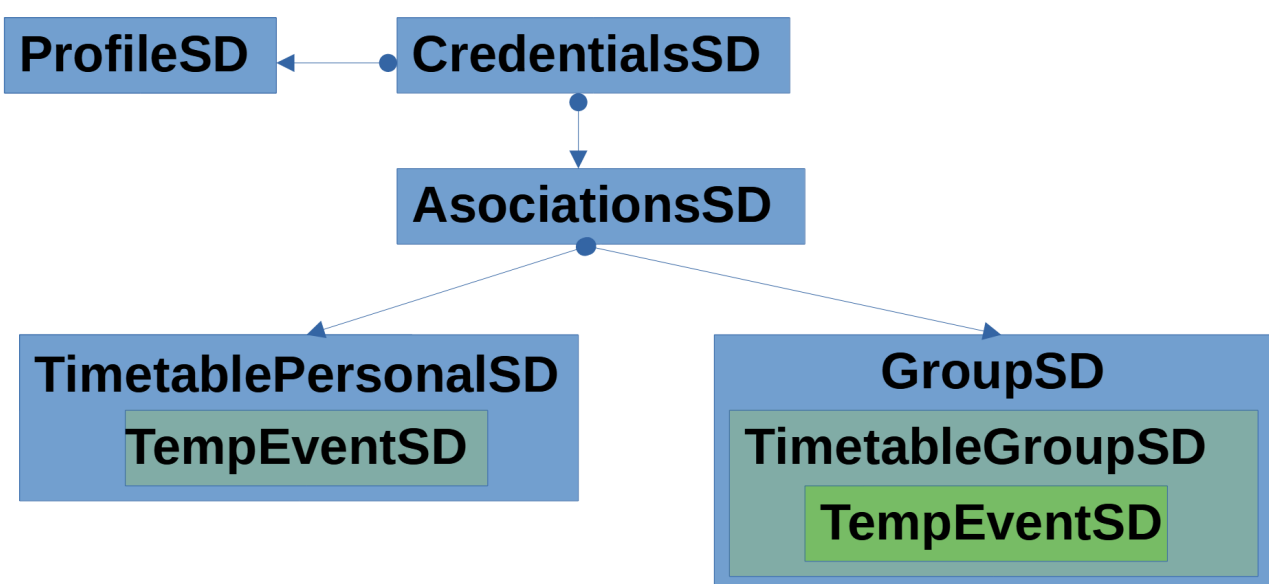


Figura 34: Modelo SD

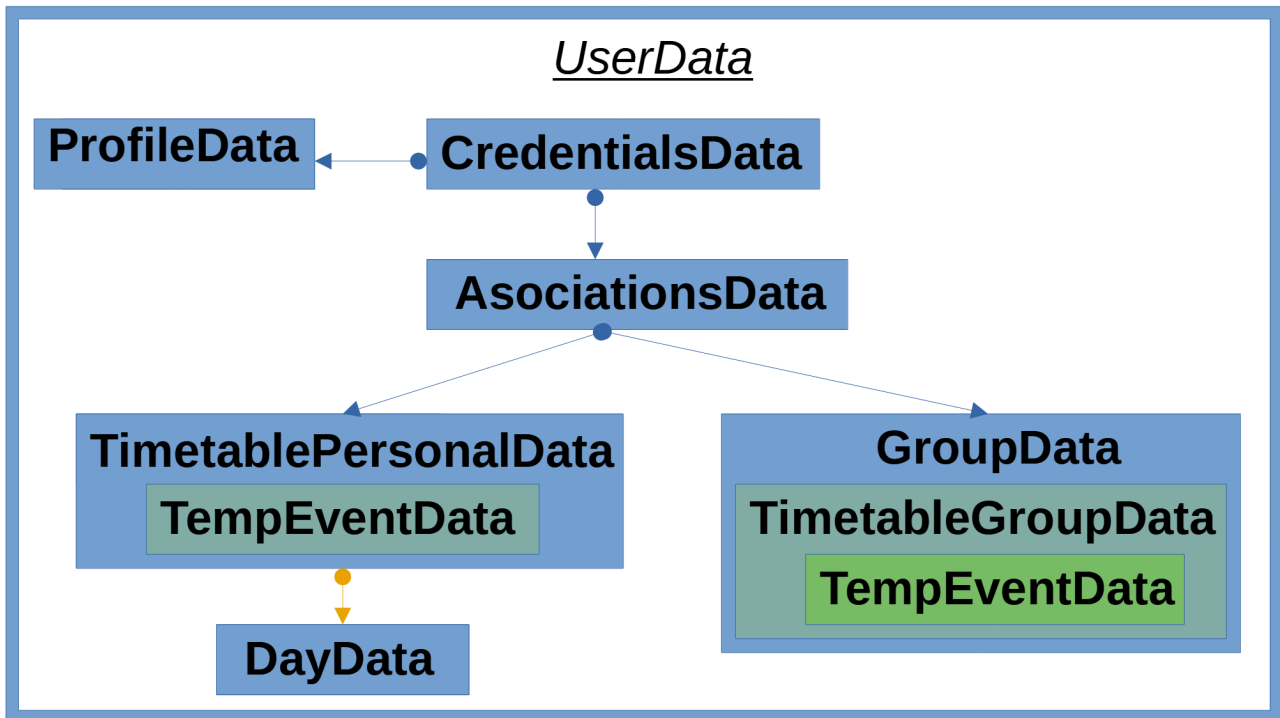


Figura 35: Modelo Data

Este último, referente a la estructura de datos manejada por el cliente, contiene más elementos dentro de su modelo (por eso se representa dentro de una paquete llamado *userData*), sin embargo, estos no interactúan con la base de datos y algunos se encuentran en fases de desarrollo para futuras líneas de trabajo.

Establecer el modelo SD conlleva que tanto *backend* como *frontend* deben tener funciones de ensamblado que adapten los datos que ellos trabajan a este último y viceversa. De esta manera nos aseguramos que ambas partes pueden traducir los datos en cuanto lo reciben y operar con ellos sin problemas.

En las figuras 37, 38 y 39 podemos ver los ficheros que se encargan de dicho ensamblado por parte del cliente y la función principal de ensamblado de los mismos:

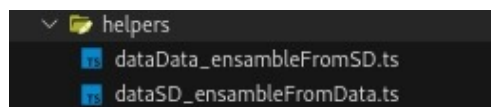


Figura 37: ficheros ensambleData

```

18 export const UserData_ensambleFromSD = (value:userSD.Interface):UserData => {
19
20   let result:UserData = new UserData();
21
22   result.setProfile( profileData_ensambleFromSD(value.profile) );
23   result.setAsociations( asociationsData_ensambleFromSD(value.asociations, value.timetablePersonallist, value.groupList) );
24   result.setTimetablePersonallist( timetablePersonallistData_ensambleFromSD(value.timetablePersonallist) );
25   result.setGroupList( groupListData_ensambleFromSD(value.groupList) );
26   return result;
27 }
    
```

Figura 36: Ejemplo código ensamble Data 1

```

17 export const UserSD_ensambleFromData = (value:UserData):userSD_Interface => {
18
19   let result:userSD_Interface = {
20     profile: profileSD_ensambleFromData(value.getProfile()),
21     asociations: asociationsSD_ensambleFromData(value.getAsociations()),
22     timetablePersonallist: timetablePersonallistSD_ensambleFromData(value.getTimetablePersonallist()),
23     groupList: groupListSD_ensambleFromData (value.getGroupList())
24   }
25   return result;
26 }

```

Figura 38: Ejemplo código ensamble Data 2

Por otra parte, en las figuras 39, 40 y 41 podemos ver el equivalente a las anteriores por la parte del servidor:

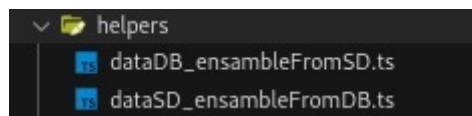


Figura 39: Ficheros ensambleDB

```

10 export const userSD_ensambleFromDB = ( profileDB:ProfileDB_Interface,
11                                       asociationsDB:AsociationsDB_Interface,
12                                       personalTimetableListDB:Array<TimetablePersonalDB_Interface>,
13                                       groupListDB:Array<GroupDB_Interface>,
14                                       groupTimetableListDB:Array<TimetableGroupDB_Interface>
15                                       ):userSD_Interface =>{
16
17   const result:userSD_Interface = {
18     profile: profileSD_ensambleFromDB(profileDB),
19     asociations: asociationsSD_ensambleFromDB(asociationsDB),
20     timetablePersonallist: timetablePersonallistSD_ensambleFromDB(personalTimetableListDB),
21     groupList: groupListSD_ensambleFromDB(groupListDB, groupTimetableListDB),
22   }
23   return result;
24 }

```

Figura 40: Elemplo código ensambleDB 1

```

10 export const userDB_ensambleFromSD = ( userID:String,
11                                       profileSD:profileSD_Interface,
12                                       asociationsSD:AsociatonsSD_Interface,
13                                       personalTimetableListSD:Array<TimetablePersonalSD_Interface>,
14                                       groupListSD:Array<GroupSD_Interface>,
15                                       ):UserDB_Interface =>{
16
17   let timetableGroupListTemp:Array<TimetableGroupDB_Interface>= new Array<TimetableGroupDB_Interface>();
18   for(let i=0 ; i<groupListSD.length ; i++){
19     timetableGroupListTemp.concat(timetableGroupListDB_ensambleFromSD(groupListSD[i].timetableGroupList));
20   }
21
22   const result:UserDB_Interface = {
23     profile: profileDB_ensambleFromSD(userID, profileSD),
24     asociations: asociationsDB_ensambleFromSD(userID, asociationsSD),
25     timetablePersonallist: timetablePersonallistDB_ensambleFromSD(personalTimetableListSD),
26     groupList: groupListDB_ensambleFromSD(groupListSD),
27     timetableGroupList:timetableGroupListTemp
28   }
29   return result;
30 }

```

Figura 41: Ejemplo código ensambleDB 2

6.5.6 Implementación de gestures

Uno de los elementos más importantes para hacer que una aplicación pueda ser atractiva para los usuarios y que estos la utilicen es implementar una interacción fluida y gratificante.

Actualmente los usuarios se encuentran muy familiarizados con el proceso de simplemente mover su pantalla o deslizar los dedos por ella para llevar a cabo acciones como navegar entre los menús de la aplicación o hacer scroll por el contenido mostrado. Esto hace que sea necesario implementar el uso de gestures dentro del proyecto.

Esto surgió en el momento que necesitábamos construir una forma de navegación entre las distintas tablas que el usuario podía visualizar, la cual debería ser intuitiva, fácil de llevar a cabo y que fuera acorde con los pasos actuales que ejerce la industria en el ámbito de interacción con el usuario.

De esta manera surgió la idea de implementar gestures que permitieran al usuario navegar entre dichas tablas simplemente haciendo un desplazamiento con su dedo sobre la pantalla.

Esto se implementó tanto en las tablas personales como en las tablas grupales, con la salvedad de que las primeras exigen realizar un scroll vertical mientras que las segundas lo implementan en horizontal.

Además de esto, para facilitar la intuición del usuario, se han agregado una serie de indicadores que muestran la tabla en la que se encuentra el usuario y cuántas hay disponibles para ver. En las figuras 42 y 43 podemos ver el resultado final.

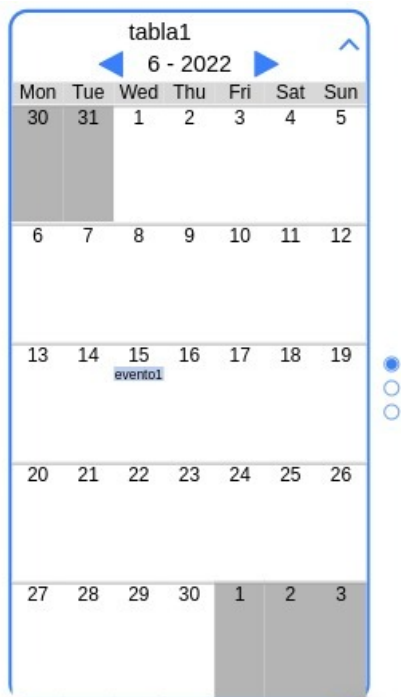


Figura 42: Ejemplo implementación gestures 1

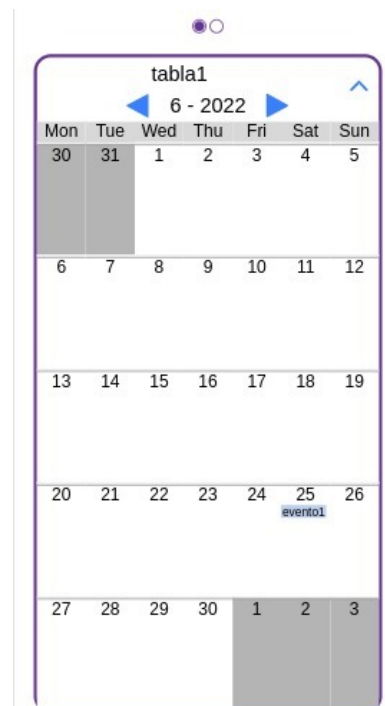


Figura 43: Ejemplo implementación gestures 2

Para que la interacción con el usuario sea correcta, hemos necesitado implementar dos funciones de control sobre la lectura del gesto llevada a cabo por el usuario, uno que se dispara cuando el gesto a concluido, y otro que de lanza siempre que existe un movimiento, aunque este aun no haya terminado.

En la figura 44 podemos ver la función encargada de disparar las dos funciones de control mencionadas anteriormente. Esta pertenece a las tablas de la sección personal, lo podremos notar por que uno de sus atributos especifica que unicamente es leído el movimiento en el eje Y.

```
40     ngAfterViewInit(): void {
41         this.gestureScroll= this.gestureController.create({
42             el: this.HTMLdiv.nativeElement,
43             gestureName: 'gestureScroll',
44             threshold: 0,
45             direction: "y",
46             onEnd: (ev:GestureDetail) => this.gestureScroll_onEnd(ev),
47             onMove: (ev:GestureDetail) => this.gestureScroll_onMove(ev)
48         }, true);
49         this.gestureScroll.enable(true);
50     }
```

Figura 44: Ejemplo código gestures

El uso de estas dos funciones conlleva no solo una navegación entre las tablas, sino la posibilidad de ejercer un movimiento de desplazamiento de las mismas con su posterior autocorrección por parte de la aplicación. Esto hace que se trate de un movimiento mucho mas natural y agradable para el usuario.

7º Trabajos relacionados

7.1 Versión antigua de GroupApp-client

La aplicación llamada *GroupApp-client* constituye la parte *frontend* del proyecto, sin embargo, esta no es la misma que se creó inicialmente para el desarrollo del mismo. En el momento que se decidió cambiar el *framework* del cliente migrando desde *Angular* a *Ionic-Angular* se necesitó crear un nuevo proyecto cliente desde cero (el cual sería el cliente actual).

La versión antigua del cliente tenía implementada toda la sección de acceso a la aplicación y las verificaciones necesarias para llevar a cabo una interacción fluida por parte del cliente, incluyendo la consistencia de datos mediante *JWT*.

Ionic no solo influye en la capacidad de desarrollar proyectos híbridos, sino también en los estilos normalizados que llevan a cabo los elementos proporcionados por el mismo. En las figuras 45 y 46 podemos ver las diferencias entre los estilos de ambas versiones.

GroupApp

Acceso

Email
test1@gmail.com

Password

Login

¿No tienes cuenta? [Crear una aquí](#)

Server status: **Online**

Figura 45: Ejemplo acceso antiguo

GroupApp

test1@gmail.com

ACCEDER

¿No tienes cuenta? [Crear una aquí](#)

Server status: Offline

Figura 46: Ejemplo acceso nuevo

A lo largo del presente documento se realizarán más referencias a esta antigua versión y se analizarán algunos aspectos de la misma, ya que supone un elemento importante dentro del proceso de desarrollo del proyecto.

8º Líneas de trabajo futuras

8.1 Interacción de usuarios sin registro

La necesidad de tener que pasar por un proceso de registro para poder utilizar las funcionalidades básicas de una aplicación puede llegar a conformar una gran barrera para que nuevos usuarios se sumen a la utilización de este.

Este dilema se nos presentó bastante avanzado el proyecto, de modo que las distintas secciones del mismo ya estaban creadas y configuradas en mayor o menor medida. La mejor solución para poder solventar esta barrera es hacer que todas aquellas funcionalidades que no precisan obligatoriamente de registro por parte del usuario puedan realizarse sin la necesidad de llevarlo a cabo.

Esto no es nuevo y lo realizan una gran cantidad de compañías en sus aplicaciones, algunas como *Facebook* te permiten poder ver buena parte del contenido subido por sus usuarios sin la necesidad de haberte identificado previamente, otras directamente están diseñadas para llevar a cabo toda su funcionalidad de forma anónima.

En nuestro caso consideramos las siguientes funcionalidades como necesarias para poder ser llevadas a cabo sin la necesidad de identificación:

- Poder llevar a cabo toda la funcionalidad de la sección de tablas personales. Esto incluye tanto poder crear nuestras propias tablas como nuestros eventos con todas las acciones que se pueden realizar con ellos.
- Visualizar en la aplicación en la aplicación los grupos existentes con toda la información pública que estos quieren transmitir. Sin embargo, no sería posible unirse a estos grupos sin registro previo.
- Llevar a cabo búsquedas tanto de los grupos existentes dentro de la base de datos como de otros usuarios.

Para llevar a cabo esto necesitaríamos construir una función de verificación que sea utilizada cuando el usuario quiera realizar interacciones que requieran de una verificación previa. Gracias al uso de *JWT* podemos comprobar cuando un usuario se encuentra verificado correctamente en la aplicación sin provocar por ello posibles brechas de seguridad.

Gracias a que el *Jason Web Token* se actualiza cada vez que es utilizado, puesto que el servidor nos envía uno nuevo, podemos diferenciar cada una de las interacciones que lleva el usuario y que requieren de una verificación.

Para que los usuarios pudieran interactuar con su sección personal necesitaremos almacenar los datos de forma nativa en nuestro dispositivo. Teniendo en cuenta que estamos utilizando una aplicación híbrida, nuestra mejor opción es crear un servicio que encapsule toda la funcionalidad de guardado local de datos.

En el proyecto ya existe el comienzo de implantación para llevar a cabo este aspecto, pues dentro de la sección *services* podemos encontrar un servicio llamado *local-data*, cuya función principal se basa en realizar esta encapsulación a las acciones de acceso a los datos locales.

En la figura 47 podemos ver como el principio de implementación de este servicio ya incluye variables para identificar la plataforma en la que nos encontramos, con el objetivo de poder optimizar las acciones de guardado y obtención de los datos de la mejor forma posible dependiendo de la plataforma en la que nos encontremos.

```
9  export class LocalDataService {
10
11
12     private localDataFolderPath:String = ".././Docs/";
13     private deprecatedFileName:String = "deprecatedData";
14     private deprecatedFile:File;
15
16     private identificationJWT:string="groupApp-JasonWebToken";
17
18
19     constructor(
20         private platform: Platform
21     ) {
22         this.deprecatedFile = new File();
23         this.platform.ready().then(() => {
24             });
25     }
26 }
```

Figura 47: Servicio local-data

8.2 Implementación del funcionamiento offline

Desligar el funcionamiento de una aplicación web por parte de los usuarios de la necesidad de mantener una conexión continua a internet es algo complicado y difícil de llevar a cabo de una forma satisfactoria.

Acciones como expulsar al usuario de la aplicación por el mero hecho de no tener conexión con los servidores únicamente contribuye a que estos dejen de usarla debido a contar con una interacción hacia los usuarios poco trabajada e insatisfactoria.

El objetivo es permitir que el usuario lleve a cabo el mayor número de acciones posibles de forma offline y hacer que estas se lleven a cabo tan rápidamente como el usuario vuelva a tener conexión. De esta manera haremos que prevalezca un sentimiento de que la aplicación no limita nuestras acciones debido a la conexión.

Para llevar a cabo esto es necesario, al igual que en el punto anterior, implementar un servicio que nos permita acceder a los datos guardados localmente de una forma eficiente dependiendo del dispositivo en el que nos encontremos. Esto es debido a que las acciones llevadas de forma offline necesitan registrarse localmente hasta que sean transmitidas al servidor, con el fin de no perder los datos si se reinicia la aplicación.

Dos de los primeros elementos pensados para desarrollar esta funcionalidad son: La sección del modelo denominada *Deprecated*, la cual podemos ver en la figura 48 y cuyo

objetivo es guardar aquellos elementos eliminados mientras no se disponía de conexión, y los atributos *temporalID* de los distintos elementos.

```

7  export class DeprecatedData {
8
9      private _timetablePersonalDeprecatedList: Array<TimetablePersonalDeprecated>;
10     private _groupDeprecatedList: Array<GroupDeprecated>
11     private _timetableGroupDeprecatedList: Array<TimetableGroupDeprecated>;
12     private _tempEventDeprecatedList: Array<TempEventDeprecated>;
13

```

Figura 48: Clase deprecated Data

Implementar esta funcionalidad requiere generar un *feedback* al usuario que le permita saber cuando se encuentra sin conexión y en que momento los elementos han sido registrados por el servidor. Todo esto debe realizarse sin notificaciones y de forma muy visual, pues de lo contrario corremos el riesgo de saturar a los usuarios.

8.3 Creación de los paquetes ConnectionService

Como hemos visto en apartados anteriores, el proyecto utiliza un total de tres modelos de datos distintos dependiendo de si nos encontramos en el cliente, en el servidor o si queremos realizar el paso de información entre ambos.

Debido a que el modelo utilizado para las comunicaciones debe ser conocido tanto por el cliente como por el servidor, una implementación futura que se vuelve más necesaria a medida que crece la aplicación es crear paquetes que encapsulen la funcionalidad de las comunicaciones junto con dicho modelo (de forma semejante a la que se realizó con el *DataBaseAccess* explicado anteriormente).

En este caso se necesitarían un total de dos paquetes:

- **ConnectionServer:** Paquete que se encontraría de lado del servidor y que encapsularía tanto el modelo *DS* como las funciones de recepción y envío de mensajes desde y hacia el cliente.
- **ConnectionClient:** Paquete del lado del cliente que encapsula el modelo *DS* y todas las funciones de envío y recepción de datos desde y hacia el servidor.

En la figura 49 visualizamos el esquema resultante de la arquitectura general de la aplicación tras implementar dichos paquetes.

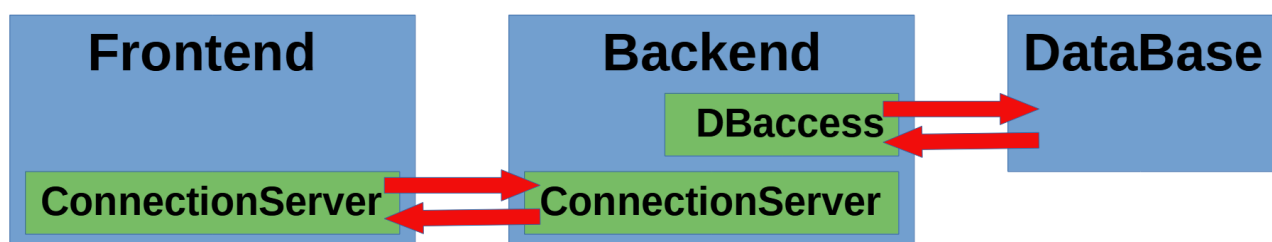


Figura 49: Creación de los paquetes ConnectionService

Una de las principales motivaciones para llevar a cabo este escapsulamiento es la capacidad de que el servidor pueda identificar la versión del paquete que tiene el cliente, y de esta manera poder adaptar el tratamiento de los datos en el caso de que se tratara de una versión anterior a la actual.

8.4 Creación de eventos avanzados

Los eventos constituyen el núcleo de la funcionalidad del proyecto, por lo que deben ser a la vez sencillos de utilizar pero tan potentes como el usuario pueda llegar a demandar. Teniendo en cuenta que un grupo puede estar administrado por una empresa o institución, es importante agregar una capa de funcionalidad extra a la hora de diseñar los eventos.

Una de las mejores formas para llevar a cabo esto es hacer que los eventos puedan programarse, de modo que sea suficiente con diseñar un solo evento pero que aparezcan todos aquellos que sean necesarios para cumplir la programación del primero.

Un ejemplo sería un grupo de clases en el que un evento es programado para cubrir la clase de matemáticas, la cual se imparte los martes y jueves de 10:00 a 11:00 entre los meses de septiembre a junio. Debe bastar con diseñar un solo evento para que el calendario se llene por completo de eventos hijos de este situados en los horarios indicados que se imparten dichas clases.

Además de esto, los eventos también deben tener implementadas funcionalidades extra como poder agregar fotos, videos, localizaciones mediante *GoogleMaps*, poder diseñar los estilos de los mismos como el color del fondo o los bordes etc.

Por último, poder linkear los eventos de los grupos en los que nos encontramos a nuestras propias tablas personales es algo muy util e importante. Esto es debido a que podríamos crear nuestro propio espacio en el que no solo nosotros nos organizamos, sino que dejamos a las personas de nuestros grupos que nso ayuden a organizarnos también.

En la imagen 50 vemos ver una aplicación cuya interfaz puede servir de modelo para implementar lo dicho en el presente punto.

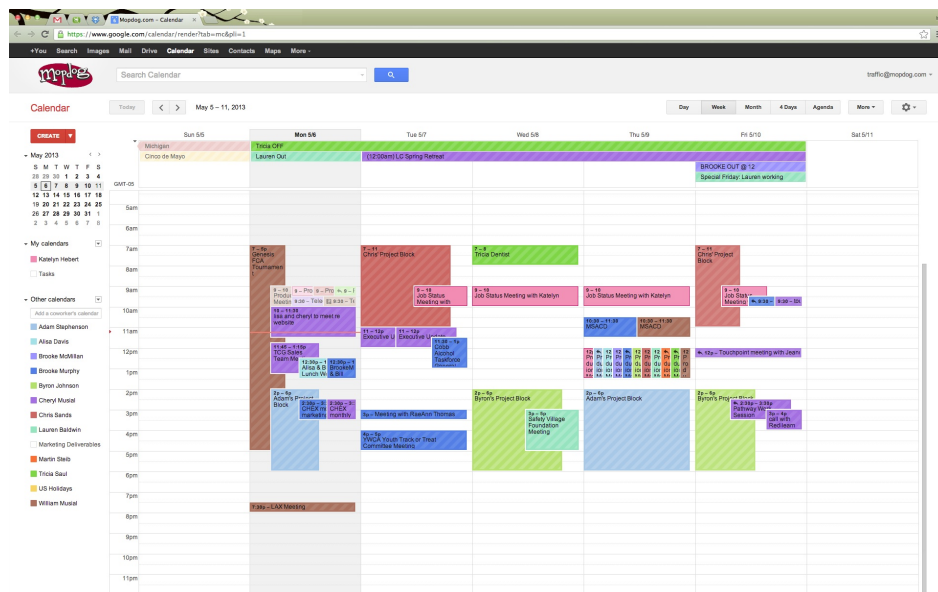


Figura 50: Ejemplo creación de eventos avanzados

8.5 Implementación de roles dentro de un grupo

Una característica innata de los grupos sociales es que estos suelen ser heterogéneos, lo cual nos puede llevar a la situación de que no todos los usuarios desempeñen los mismos roles dentro de un mismo grupo.

Esto puede hacer parte de la actividad desempeñada dentro de un grupo únicamente requiera ser visible o poder ser llevada a cabo por una parte de los integrantes del mismo y no por su totalidad.

Un ejemplo simple es un grupo destinado a la organización de un supermercado, donde pueden estar suscritos tanto clientes como empleados, los cuales a su vez pueden desempeñar muchos roles, como cajeros, reponedores, encargados de sección etc. En este caso, los eventos que puede visualizar o crear un encargado seguramente no sean los mismo que aquellos con los debe interactuar el personal de limpieza.

La solución es dotar a los grupos de un sistema en el cual los administradores del mismo puedan crear y gestionar distintos roles, los cuales son asignados a los usuarios que se encuentran suscritos al mismo. De esta manera, como dos roles pueden tener distintos privilegios dentro de un mismo grupo, se pueden crear un sistema con distintos tipos de usuarios.

GroupApp: Aplicación de gestión de eventos y grupos distribuida y centralizada

En la figura 51 podemos ver una referencia a este concepto perteneciente a la aplicación de chats online *discord*, la cual puede servir como una idea aproximada sobre este concepto aplicado a nuestro proyecto:

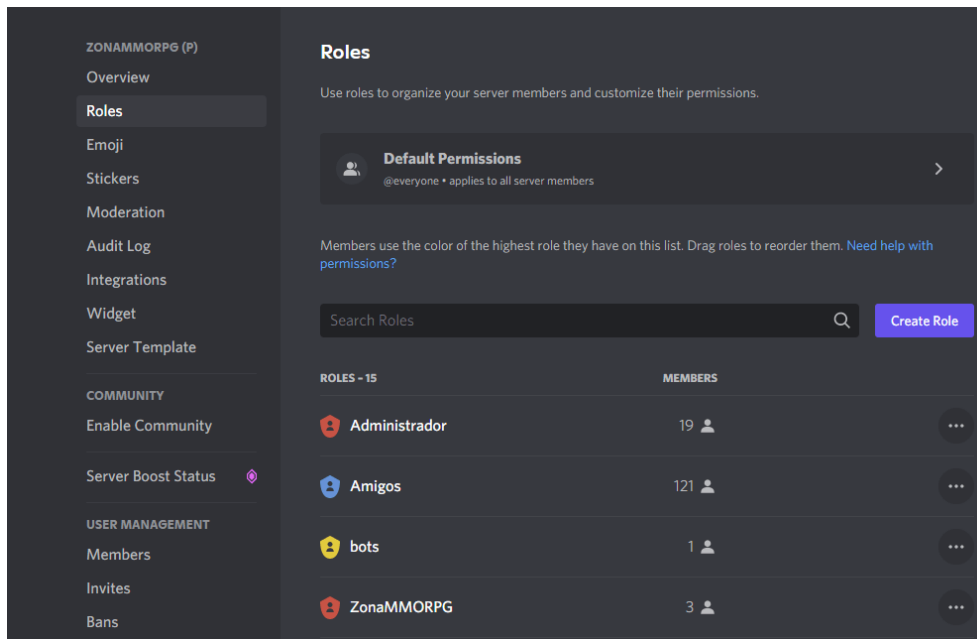


Figura 51: Ejemplo implementación de roles dentro de un grupo

9º Conclusiones

El desarrollo del presente proyecto se ha caracterizado por una gran cantidad de matices que lo han llevado a ser el proyecto más importante que a nivel personal he desarrollado hasta el momento. Escribir unas conclusiones generales no es sencillo, debido a la gran cantidad de elementos que interaccionan en tan poco espacio.

Primero analizaremos detallaremos uno a uno nuestras conclusiones respecto a los objetivos funcionales inicialmente propuestos:

- **Gestión de cuentas de usuario:** La aplicación es capaz de crear cuentas de usuario y de enlazar las mismas con las distintas colecciones que se dan dentro de la base de datos, además de realizar modificaciones. Podemos concluir que es un aspecto trabajado.
- **Gestión de tablas personales:** Un usuario puede crear todas las tablas que quiera, navegar entre ellas, modificarlas y añadir todos los eventos de quiera a cada una de dichas tablas. Podemos concluir que es un objetivo que se ha alcanzado plenamente.
- **Gestión de grupos y tablas grupales:** La aplicación acepta crear una cantidad ilimitada de grupos por parte de cualquier usuario que la utilice, además de que estas pueden tener todas las tablas grupales que quieran los usuarios. La información de los grupos se comparte correctamente entre todos sus miembros y estos pueden unirle libremente a los grupos. Su puede determinar que es un objetivo que se ha completado.
- **Gestión de búsquedas y descubrimientos:** Un usuario puede realizar tanto al búsqueda de otros usuarios en la aplicación como de los grupos que existen. Además de esto, puede unirse a nuevos grupos mediante la realización de dicha búsqueda. Se cumplen todas las acciones planteadas para este apartado, por lo que podemos determinar que es un objetivo alcanzado exitosamente.
- **Gestión de notificaciones:** El sistema muestra mensajes de notificaciones al usuario cuando acontece un evento que le concierne. Se trata de un objetivo funcional sencillo pero cumplido satisfactoriamente.

Por otra parte, podemos determinar nuestras conclusiones sobre los objetivos no funcionales:

- **Portabilidad:** Gracias a la implementación de Ionic el *Frontend* del proyecto esta consolidado como una aplicación híbrida, lo que quiere decir que puede utilizarse sin dificultad en una gran variedad de soportes físicos. Teniendo esto cuenta, podemos determinar que la aplicación consta de una gran portabilidad, concluyendo en que es un objetivo totalmente logrado.
- **Seguridad y protección de datos:** Existen una gran cantidad de brechas de seguridad que se pueden dar dentro de una aplicación web. Sin embargo, el uso

de cifrado a la hora de registrar contraseñas, el empleo de JWK y la inclusión de un modelo de datos exclusivo para el envío de mensajes entre el cliente y el servidor, denotan que se han introducido mecánicas y herramientas que en busca de la seguridad y protección de los datos.

- **Usabilidad:** La aplicación esta enfocada a mostrar la información de una forma clara y utilizar una interfaz amigable e intuitiva. El uso de gestures y la indicación de la tabla que se está visualizando son un ejemplo de trabajo en este objetivo, aunque recordemos que la interacción con el usuario es un apartado que requiere un gran desarrollo y siempre es mejorable.
- **Funcionamiento en tiempo real:** La aplicación consta del uso de bibliotecas como *Socket.IO* que le permiten realizar una comunicación cruzada entre el cliente y el servidor en tiempo real, por lo que este objetivo se ha logrado con éxito.

A parte de los objetivos marcados, podemos hacer un inciso en el uso de una gran cantidad de tecnologías que utilizamos a para desarrollar la funcionalidad de la aplicación. Podemos hacer una pequeña lista con lo que nos han aportado algunas de ellas:

- El uso de JWT nos ha permitido crear una integridad de datos en el cliente y hacer que este se sienta cómodo en la aplicación, además de añadir una capa de seguridad al uso de la misma.
- *Socket.IO* nos brinda la posibilidad de establecer una comunicación en tiempo real entre el cliente y el servidor, lo cual es fundamental debido a que el cliente necesita recibir notificaciones esporádicas. Esto es algo que no se consigue únicamente con las peticiones REST.
- *Ionic* no solo nos ha permitido crear una aplicación híbrida, sino trabajar con elementos normalizados a todos los estilos empleados en dispositivos móviles, lo cual es muy importante a la hora de mantener la cohesión en la aplicación.
- Emplear el sistema de gestión de paquetes *NPM* para crear nuestro propio paquete de gestión a la base de datos nos ha permitido familiarizarnos con la publicación de código público y el encapsulamiento de sección del proyecto que pueden ser reutilizables.

El enfoque inicial que seguía y continua siguiendo la aplicación es la versatilidad en el uso de la misma. Enfocarse en conseguir una aplicación que a su vez fuera fácil de usar e intuitiva para el usuario promedio, además de robusta y potente para las empresas requiere de un gran conocimiento sobre el área y un largo periodo de desarrollo.

Puede parecer a priori una meta sencilla, pero nada más lejos de la realidad. El fino equilibrio necesario para llegar a cumplir este objetivo solo puede ser obtenido a base del pulido que el equipo de desarrollo debe realizar apoyándose en su experiencia y conocimientos sobre el área.

Hemos de admitir que esto no nos pilla de imprevisto, pues ya sabíamos que crear un producto cuya misión final era llegar a estar tan pulido como para poder ser una aplicación

preparada para lanzar al mercado era algo que no podríamos hacer debido al tiempo que disponíamos para llevarlo a cabo.

Una de las evidencias es el apartado anterior, donde podemos ver como existen una gran cantidad de elementos que no se han implementado en el proyecto, pero que ya se han planeado y diseñado a nivel conceptual. Es difícil poner el punto final a un desarrollo donde verdaderamente pones algo de ti.

Sin embargo, podemos decir que el desarrollo de la aplicación ha sido exitoso gracias a que ha cumplido uno de sus objetivos vitales, servir como proceso educativo en el ámbito del desarrollo web a una persona que, haciendo autocrítica, disponía de unos conocimientos nulos respecto al área.

He de admitir que en lo que referente al aspecto didáctico, se trata de un planteamiento muy ambicioso, puesto que no se limita únicamente a un área concreta del desarrollo web, sino que toca todas las posibles.

El empleo de múltiples herramientas y el desarrollo tanto del *frontend* como del *backend*, además de aspectos secundarios como utilizar un sistema de versionado o el uso de múltiples bibliotecas como *Socket.IO* o *Moongoose* son un ejemplo de todos los aspectos que se han abarcado.

Pese a que esto ya trae consigo un gran trabajo, el elemento clave es que dicho aprendizaje se ha desarrollado de una forma totalmente autodidáctica, puesto que no ha sido necesario, en ningún momento, requerir al apoyo de terceras personas par llevar a cabo el desarrollo.

Este se puede considerar uno de los grandes logros, no únicamente de llevar a cabo el presente proyecto, sino de la propia titulación en sí misma. Conseguir que un alumno llegue a convertirse en un profesional autodidacta no es una tarea sencilla.

Esto aún se complica más si introducimos la otra parte de la ecuación, llegar a ser una persona resolutiva que es capaz de dar solución a los problemas gracias al uso de su ingenio y creatividad.

Hacer esto es difícil, requiere un proceso de aprendizaje continuo y enfrentamiento a la incertidumbre que únicamente puede llegar a solventarse gracias a cultivar, poco a poco, las capacidad de adaptación.

A día de hoy me alegro de que estas sean las conclusiones tanto de la elaboración mi TFG como de haber realizado mi titulación. Ser autodidacta y resolutivo son dos de las premisas básicas para ser un ingeniero, es nuestro pan de cada día, ser capaz de brindar una solución a los problemas que se plantan en nuestro camino.

Esto es un ámbito vocacional que requiere que la persona sienta pasión a la hora de dedicarse al mismo, de nada sirve que obtenga un título si no tienes la motivación por mejorar en un ámbito tan exigente.

GroupApp: Aplicación de gestión de eventos y grupos distribuida y centralizada

Bibliografía

Documentación sobre las herramientas utilizadas

Documentación sobre Typescript

- Pagina oficial de *Typescript*:
 - <https://www.typescriptlang.org/>
- *Typescript* introducción oficial:
 - <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>
- Artículo de comparación entre *Typescript* y *JavaScript*:
 - <https://radixweb.com/blog/typescript-vs-javascript>

Documentación Socket.IO

- Pagina web del repositorio NPM de *Socket.IO*:
 - <https://www.npmjs.com/package/socket.io>
- Pagina web oficial de *Socket.IO*:
 - <https://socket.io/>

Documentación sobre Git y GitHub

- Pagina oficial de *Git*:
 - <https://git-scm.com/>
- Pagina oficial del portal de acceso a *GitHub*
 - <https://github.com/>
- Tutorial básico *Git* y *GitHub*:
 - <https://bluuweb.github.io/tutorial-github/01-fundamentos/#enlaces>

Documentación sobre Jason Web Token

- Pagina web oficial de *Jason Web Token*:
 - <https://jwt.io/>
- Lista oficial de librerías que implementan *JWT*:
 - <https://jwt.io/libraries>

Documentación sobre Angular

- Pagina oficial de *Angular*:
 - <https://angular.io/>
- ¿Qué es *Angular*?:

- <https://angular.io/guide/what-is-angular>
- Artículo comparativa *Angular*, *React* y *Vue.js*:
 - <https://www.openinnova.es/angular-vs-react-vs-vue-espanol-cual-elegir/>

Documentación sobre Ionic

- Pagina oficial de *Ionic*:
 - <https://ionicframework.com/>

Documentación sobre HTML

- Tutoriales sobre *HTML*:
 - <https://lenguajehtml.com/html/>

Documentación sobre SASS y CSS

- Página web oficial de *SASS*:
 - <https://sass-lang.com/>
- Tutoriales sobre *CSS*:
 - <https://lenguajecss.com/css/>

Documentación sobre Node y NPM

- Página web oficial de *Node*:
 - <https://nodejs.org/en/>
- Página oficial del portal de acceso a *NPM*:
 - <https://www.npmjs.com/>
- Repositorio del paquete de creación propia para la gestión de la base de datos:
 - <https://www.npmjs.com/package/databaseservice-groupaplibrary>

Documentación Express

- Página web del repositorio *NPM* de *Express*:
 - <https://www.npmjs.com/package/express>
- Pagina oficial de *Express*:
 - <http://expressjs.com/>

Documentación MongoDB y Moongose

- Página oficial de *MongoDB*:
 - <https://www.mongodb.com/>
- Página web del repositorio *NPM* de *Moongose*:
 - <https://www.npmjs.com/package/mongoose>

Documentación sobre la planificación

- Pagina web donde se encuentra del diagrama de Gantt:
 - <https://plan.tomsplanner.es/#doc=VbcrTbCqwcvgjnQttvEv>

Documentación sobre los aspectos relevantes

- Pagina web del repositorio donde se encuentra el paquete *DataBaseService*;
 - <https://www.npmjs.com/package/databaseservice-groupapplibrary>

Glosario

- **Backend:** Referido a la parte de un sistema software distribuido que lleva a cabo las dependencias asignadas al servidor. Es el encargado de gestionar la información que proporciona el usuario recogida por el sitio web.
- **Frontend:** Permite a los usuarios acceder y solicitar las prestaciones y servicios del sistema de información subyacente.
- **Framework:** Es utilizado para desarrollar productos software y esta compuesto por un conjunto de herramientas y módulos que pueden ser reutilizados para varios proyectos.
- **HTML:** Lenguaje de Hipertexto que define tanto el significado como la estructura del diseño web que implementaremos en nuestra aplicación.
- **CSS:** Lenguaje que maneja el diseño y presentación de las páginas web, es decir, cómo lucen cuando un usuario las visita.
- **SASS:** Es un preprocesador CSS, es decir, es una herramienta que nos permite generar, de manera automática, hojas de estilo, añadiéndoles características que no tiene CSS, y que son propias de los lenguajes de programación
- **Gestures:** Referente a los gestos identificados por un dispositivo informático, los cuales son realizados por el usuario de dicho dispositivo.
- **Offline:** Referente a que está disponible o se realiza sin conexión a internet o a otra red de datos.
- **Online:** Referente a que está disponible o se realiza a través de internet o de otra red de datos.
- **GroupAPP:** Nombre del proyecto.
- **GroupApp-Client:** Referente a la parte *Frontend* del proyecto.
- **GroupApp-Servidor:** Referente a la parte *Backend* del proyecto.
- **MongoDB:** Sistema de base de datos no relacional orientado a documentos y de código abierto.
- **Node:** Entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript.
- **Ionic:** SDK de código abierto completo para el desarrollo de aplicaciones móviles híbridas.

- **Moongose:** Mongoose es una biblioteca de programación orientada a objetos de JavaScript que crea una conexión entre MongoDB y el marco de la aplicación web Express.
- **Express:** Express es un framework web transigente, escrito en JavaScript y alojado dentro del entorno de ejecución NodeJS.
- **NPM:** Responde a las siglas de Node Package Manager o manejador de paquetes de node, es la herramienta por defecto de JavaScript para la tarea de compartir e instalar paquetes.
- **Angular:** Framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página.
- **Typescript:** TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases.
- **Socket.IO:** Biblioteca basada en eventos para aplicaciones web en tiempo real. Permite la comunicación bidireccional en tiempo real entre clientes web y servidores.
- **Gmail:** Servicio de correo electrónico gratuito proporcionado por el motor de búsqueda Google.