



---

# APLICACIÓN ANDROID. CLIMAPP

---



13 DE OCTUBRE DE 2022  
LUIS ALFONSO HUERTAS DELGADO  
MIGUEL JARA ARROYO  
PEDRO DEL CASTILLO GÓMEZ  
JORGE DEL CASTILLO GÓMEZ

# ÍNDICE

<b>Introducción.....</b>	<b>7</b>
<b>Propuesta inicial.....</b>	<b>8</b>
Descripción de la idea.....	8
Público objetivo.....	8
¿A quién va dirigido?.....	9
Funcionalidad principal.....	10
<b>Análisis.....</b>	<b>11</b>
Casos de usos estructurales.....	11
Casos de usos no estructurales.....	11
Conjunto de casos de uso.....	12
Requisitos no funcionales.....	13
<b>Metodologías de desarrollo.....</b>	<b>15</b>
Proceso de desarrollo.....	15
Planificación del proyecto.....	15
Proceso de planificación.....	15
Distribución de casos de uso.....	16
Análisis de distribución.....	17
Planificación inicial.....	19
Planificación modificada.....	24
Análisis de la planificación.....	32
Camino Crítico.....	39
Configuración del seguimiento de la planificación.....	41
Señor Blanco (Equipo 1).....	46
Proceso seguido.....	46
Análisis del progreso.....	46
Planificación tareas CU01 - Añadir Evento de Municipio.....	47
Planificación tareas CU02 - Añadir Evento de Ruta Montaña.....	49
Planificación tareas CU03 - Añadir Usuario.....	51
Planificación tareas CU04 - Añadir una barra de búsqueda y filtrado de ubicaciones.....	53
Distribución de asignación de incidencias.....	55
Contribución de los líderes de equipo.....	55
Señor Marrón (Equipo 2).....	56
Proceso seguido.....	56
Análisis del progreso.....	56
Planificación tareas CU05 - Añadir preferencias desde el menú de AppBar (ajustes...).....	57
Planificación tareas CU06 - Consultar el tiempo detallado de una ubicación.....	58
Planificación tareas CU07 - Modificar un evento.....	59
Planificación tareas CU08 - Consultar el tiempo meteorológico en la ubicación actual.....	61

Proporción de Épicas, Tareas y Subtareas.....	62
Señor Naranja (Equipo 3).....	63
Proceso seguido.....	63
Análisis del progreso.....	64
Planificación tareas CU09 - Consultar tiempo meteorológico en la ubicación actual.....	65
Planificación tareas CU10 - Modificar idioma y tema a modo oscuro.....	67
Planificación tareas CU11 - Consultar lista de eventos.....	69
Planificación tareas CU12 - Consultar un evento.....	71
Distribución de incidencias por iteración.....	73
Contribución de los líderes de equipo.....	74
Señor Azul (Equipo 1).....	74
Proceso seguido.....	74
Análisis del progreso.....	75
Planificación tareas CU13 - Iniciar sesión.....	75
Contribución de los líderes de equipo.....	77
Proporción de tareas por caso de uso.....	77
Desarrollo del progreso del proyecto.....	79
Síntesis de la planificación por roles.....	79
Desarrollo del Equipo 1 (Sr Blanco).....	81
CU01 - Añadir Evento de Municipio.....	81
Implementación de CU01.....	82
Integración de CU01.....	82
CU02 - Añadir Evento de Ruta Montaña.....	82
Implementación de CU02.....	83
Integración de CU02.....	83
CU03 - Añadir Usuario.....	83
Implementación de CU03.....	83
Integración de CU03.....	84
CU04 - Añadir una barra de búsqueda y filtrado de ubicaciones.....	84
Implementación de CU04.....	84
Integración de CU04.....	84
Desarrollo del Equipo 2 (Sr Marrón).....	85
CU05 - Añadir preferencias desde el menú de AppBar (ajustes ...).....	85
Implementación de CU05.....	85
Integración de CU05.....	85
CU06 - Consultar el tiempo detallado de una ubicación.....	86
Implementación de CU06.....	86
Integración de CU06.....	86
CU07 - Modificar un evento.....	86
Implementación de CU07.....	87
Integración de CU07.....	87
CU08 - Eliminar un evento.....	87
Implementación de CU08.....	87
Integración de CU08.....	88

Desarrollo del Equipo 3 (Sr Naranja).....	88
CU09 - Consultar el tiempo meteorológico en la ubicación actual.....	88
Implementación de CU09.....	88
Integración de CU09.....	89
CU10 - Modificar idioma y tema a modo oscuro.....	89
Implementación de CU10.....	89
Integración de CU10.....	89
CU11 - Consultar lista de eventos.....	90
Implementación de CU11.....	90
Integración de CU11.....	90
CU12 - Consultar un evento.....	90
Implementación de CU12.....	91
Integración de CU12.....	91
Desarrollo del Equipo 4 (Sr Azul).....	91
CU13 - Iniciar sesión.....	91
Implementación de CU13.....	91
Integración de CU13.....	92
CU14 - Cerrar sesión.....	92
Implementación de CU14.....	92
Integración de CU14.....	93
CU15 - Modificar usuario.....	93
Implementación de CU15.....	93
Integración de CU15.....	93
CU16 - Eliminar usuario.....	94
Implementación de CU16.....	94
Integración de CU16.....	94
<b>Diseño de la interfaz de usuario.....</b>	<b>95</b>
Mapa Navegación: pantallas, patrones y diagrama.....	95
Diagrama de casos de uso.....	95
Diagrama de navegación.....	96
Mockup.....	97
Grafo de navegación.....	100
Patrones de navegación aplicados.....	101
Patrón de lista y detalle.....	101
Patrón de cajón de navegación.....	101
Patrón de botones y objetivos sencillos.....	101
<b>Diseño arquitectónico.....</b>	<b>102</b>
Decisiones tomadas.....	102
Diagrama de componentes.....	102
Patrones arquitectónicos.....	103
<b>Gestión del entorno.....</b>	<b>103</b>
Gestión de la configuración.....	104
Entorno utilizado.....	104
Herramienta de implementación: Android Studio.....	104

Estructura de un proyecto en Android Studio.....	105
Componentes de la interfaz de Android Studio.....	106
Funcionamiento de una App en Android Studio.....	107
Compilación en Gradle.....	108
Integración de Jira en Android Studio.....	108
Ramas utilizadas en el proyecto.....	119
<b>Integración continua.....</b>	<b>119</b>
Definición de disciplinas.....	120
Implementación.....	120
Integración y Testeo.....	120
Integración de cada subsistema.....	121
Integración del sistema.....	121
<b>Implementación.....</b>	<b>122</b>
Modelo de datos.....	122
Detalles de implementación.....	124
Patrones de Diseño.....	125
Patrón Singleton.....	125
Patrón DAO.....	126
Refactorización.....	128
Patrón Repository.....	128
Patrón Model - View - ViewModel (MVVM).....	131
Aspectos novedosos.....	133
Gestión de la API.....	133
Obtención de Localización.....	133
Gestión de Permisos.....	134
Carga de municipios y montañas desde JSON.....	134
Menú de Hamburguesa.....	134
Filtro de Eventos.....	135
Usuario único.....	135
Implementación de Spinners en diversos campos.....	135
Filtrado localizaciones.....	136
Modo Oscuro.....	136
<b>Gestión de la calidad del software.....</b>	<b>137</b>
Pruebas.....	137
Pruebas implementadas.....	139
Equipo 1 (Señor Blanco).....	140
CU1 - Añadir Evento de Municipio.....	140
Test unitario.....	140
Test funcional.....	140
CU2 - Añadir Evento de Montaña.....	141
Test unitario.....	141
Test funcional.....	141
CU3 - Añadir Usuario.....	141
Test unitario.....	141

Test funcional.....	142
CU4 - Añadir barra de búsqueda y filtrado de ubicaciones.....	142
Test unitario.....	142
Test funcional.....	142
Equipo 2 (Señor Marrón).....	143
CU5 - Añadir preferencias desde el menú AppBar.....	143
Test unitario.....	143
Test funcional.....	143
CU6 - Consultar tiempo detallado de una ubicación.....	144
Test unitario.....	144
Test funcional.....	144
CU7 - Modificar un evento.....	145
Test unitario.....	145
Test funcional.....	145
CU8 - Eliminar un evento.....	145
Test unitario.....	145
Test funcional.....	146
Equipo 3 (Señor Naranja).....	147
CU9 - Consultar tiempo meteorológico en la ubicación actual.....	147
Test unitario.....	147
Test funcional.....	147
CU10 - Modificar idioma y tema a modo oscuro.....	148
Test unitario.....	148
Test funcional.....	148
Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y tras comprobar mediante un assert que el color del nombre de usuario es el del valor del azul oscuro (R.color.Azul_osc), selecciona la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.....	148
CU11 - Consultar lista de eventos.....	149
Test unitario.....	149
Test funcional.....	149
Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.....	149
CU12 - Consultar un evento.....	149
Test unitario.....	149
Test funcional.....	149
Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.....	150
Equipo 4 (Señor Azul).....	150
CU13 - Iniciar sesión.....	150
Test unitario.....	150
Test funcional.....	150

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.....	151
<b>CU14 - Cerrar sesión.....</b>	<b>151</b>
Test unitario.....	151
Test funcional.....	151
<b>CU15 - Modificar usuario.....</b>	<b>151</b>
Test unitario.....	151
Test funcional.....	151
<b>CU16 - Eliminar usuario.....</b>	<b>152</b>
Test unitario.....	152
Test funcional.....	152
Arreglo de errores.....	152
<b>Análisis de la calidad del código.....</b>	<b>153</b>
Características de SonarCloud.....	154
Repositorios utilizados.....	154
Pasos a seguir.....	154
Detección de fallos.....	155
Problemas solventados.....	155
Análisis del Sr. Blanco.....	155
Se puede observar cómo existen 580 Code Smells o incidencias de mantenibilidad. Las dos incidencias resueltas por el Sr. Blanco son las siguientes:.....	156
Análisis del Sr. Marrón.....	157
Análisis del Sr. Azul.....	158
Se puede observar cómo existen 576 Code Smells o incidencias de mantenibilidad. Las dos incidencias resueltas por el Sr. Azul son las siguientes:.....	158
Análisis del Sr. Naranja.....	159
El análisis base sobre el que se parte para resolver las dos incidencias del Sr. Naranja es el siguiente:.....	159
Se puede observar cómo existen 578 Code Smells o incidencias de mantenibilidad. Las dos incidencias resueltas por el Sr. Blanco son las siguientes:.....	159
Reflexión.....	161

# Introducción

## Motivación

En los últimos años, la plataforma Android ha gobernado notablemente el mercado móvil, de tal manera que iOS nunca ha sido capaz de recuperar el status con el que se inició en el “mundillo” tecnológico. Además, junto con el crecimiento exponencial del “Big Data”, apareció un nuevo formato de aplicaciones móviles que accedían a grandes cantidades de datos. Como consecuencia, los servicios que proporcionaban grandes fuentes de datos debían incorporar la forma de brindar datos con un acceso libre y seguro, conocido como “Open Data”.

En cuanto al nicho de mercado, apenas existen propuestas novedosas de aplicaciones meteorológicas, puesto que la mayoría de dispositivos Android incorporan estas aplicaciones de fábrica. Del mismo modo, los usuarios no parecen ser tan exigentes con este tipo de aplicaciones al ser conformistas con lo que disponen.

Hoy en día, las aplicaciones móviles no se centran únicamente en una tarea, sino que aglutinan diversas funcionalidades relacionadas entre sí. Por otro lado, el hardware móvil ha evolucionado hasta tal punto en que los desarrolladores tienen un gran abanico de opciones para desarrollar una IU con una alta usabilidad y accesibilidad.

## Principales objetivos

El propósito del equipo de desarrollo es afianzar conocimientos: cómo utilizar Android Studio, cómo tratar el gran flujo de información devuelto por la API y la forma en la que se presentan los datos al usuario. Asimismo, se va a focalizar en el diseño de la interfaz de usuario, haciéndola atractiva, novedosa y cumpliendo los requisitos de usabilidad y accesibilidad. Se pretende aportar al mercado un servicio de buena calidad a primera vista.

## Resumen

En los siguientes apartados, hablaremos de la propuesta inicial del proyecto con una descripción y su funcionalidad inicial. A continuación, se detalla el marco de desarrollo, en otras palabras, la planificación y el proceso seguido para cumplir con los objetivos propuestos. La metodología empleada por el equipo es el “Proceso Unificado”, un marco de desarrollo iterativo e incremental basado en casos de uso y centrado en la arquitectura del sistema. Por último, se presentará el funcionamiento interno de la aplicación, así como los *mockup* (interfaz de usuario primigenia) y la navegación entre las diferentes pantallas.

# Propuesta inicial

## Descripción de la idea

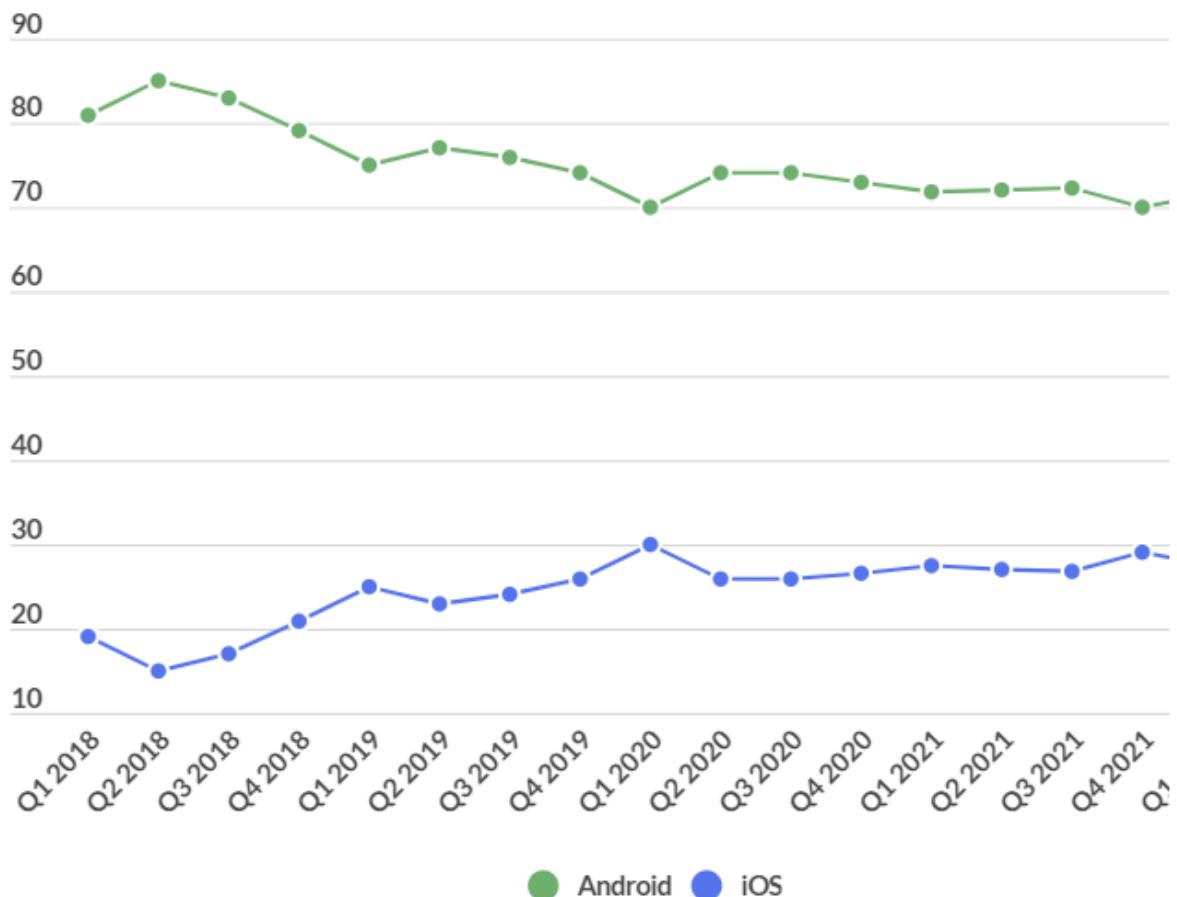
El objetivo principal de este trabajo es el desarrollo de una aplicación móvil Android llamada **ClimApp** centrada en la visualización del tiempo meteorológico de una localización en España, dicha información proporcionada por la API de AEMET.

El usuario puede crear eventos personalizados ligados a una ubicación, los cuales tendrán asignado las condiciones meteorológicas a tiempo real en dicha localización. De igual importancia, se favorece la interacción de los usuarios con el medio ambiente, permitiéndoles crear eventos tanto en municipios como en montañas.

## Público objetivo

En este apartado, se especifica a qué público va dirigida nuestra aplicación mediante estudios y gráficas. Antes de adentrarnos en el “quid” de la cuestión, cabe destacar el motivo por el que se ha elegido Android como plataforma sobre la que se implementa la aplicación. Según [StatCounter](#), Android ha dominado el 70% de la cuota de mercado por delante de iOS en creces, como se muestra en la siguiente gráfica:

**Android vs iOS global market share (%)**



*Imagen 1. Gráfica de la cuota de mercado entre iOS y Android desde 2018*

Asimismo, no es una sorpresa que Android gobierne el mercado de dispositivos móviles. Según Google, el número de usuarios con dispositivos Android ha aumentado notablemente. En la siguiente tabla, se especifica la evolución de los usuarios activos durante los últimos años en billones como unidad.

### **Android annual active users 2012 to 2021 (bn)**

Year	Users (bn)
2012	0.5
2013	0.7
2014	1
2015	1.4
2016	1.7
2017	2
2018	2.3
2019	2.5
2020	2.8
2021	3

*Source: Google*

*Imagen 2. Evolución de usuarios activos de Android desde 2012, según Google*

### **¿A quién va dirigido?**

La categoría de apps del tiempo son el 0,4% del total, con más de 11.500 disponibles y un promedio de valoración de 4,0 (frente al 4,1 del resto de categorías contando solo las apps con más de 100 valoraciones). El 12% de las apps del tiempo se han descargado más de 50.000 veces (9,5% de media para el resto de apps).

Además dentro de la tienda Play Store de Google, la aplicaciones del tiempo estando dentro de las categorías más usadas, es la que atrae al menor número de usuarios, como se puede

ver en el siguiente enlace, el 32% de los usuarios se descarga una aplicación climatológica:  
[Estadísticas para este 2022 en la tienda de Google Play](#)

Esto es debido a que la totalidad de los dispositivos android ya vienen instalados con una aplicación climatológica, esto deriva en que solo los usuarios que quieren una experiencia mejor que a la que les dan sus dispositivos buscan una alternativa a la aplicación preinstalada, estos usuarios no buscan una aplicación que les diga el tiempo sin más, sino que además buscan funcionalidades extras y podríamos decir que son un tipo de usuarios que buscan una aplicación climatológica más específica.

Es por esto que el **prototipo de usuario** al que ya he dirigido nuestra aplicación será fundamentalmente una persona mayor de edad, entre los 18 y 50 años, la cual realice actividades o eventos de una manera regular, y que sea importante para dichos eventos conocer la situación meteorológica de la ubicación en la que se producen, como por ejemplo eventos deportivos, bodas y banquetes al aire libre, etc.

## Funcionalidad principal

Para poder utilizar las diferentes funcionalidades de la aplicación, un usuario debe registrarse con unas credenciales. Una vez se registre e inicie sesión, podrá :

- En la pantalla principal, visualizar el tiempo meteorológico de la ubicación actual del dispositivo.
- Visualizar el listado de eventos en municipios o montañas creados por el usuario.
- Gestionar un evento, así como su creación, edición y borrado, mediante las preferencias (fecha, nombre, descripción, color, ubicación, etc).
- Consultar en detalle el tiempo meteorológico de una ubicación específica.
- Editar los datos personales y borrar la cuenta de usuario.

# Análisis

En este apartado se especifican los casos de usos estructurales y no estructurales del proyecto.

## Casos de usos estructurales

- CU-01: Añadir Evento de Municipio.
- CU-02: Añadir Evento de Ruta Montaña.
- CU-03: Añadir Usuario.
- CU-04: Añadir una barra de búsqueda y filtrado de ubicaciones.
- CU-05: Añadir preferencias desde el menú de AppBar (ajustes ...)

## Casos de usos no estructurales

- CU-06: Consultar el tiempo detallado de una ubicación.
- CU-07: Modificar un evento.
- CU-08: Eliminar un evento.
- CU-09: Consultar tiempo meteorológico en la ubicación actual
- CU-10: Modificar idioma y tema a modo oscuro
- CU-11: Consultar lista de eventos.
- CU-12: Consultar un evento.
- CU-13: Iniciar sesión.
- CU-14: Cerrar sesión.
- CU-15: Modificar usuario.
- CU-16: Eliminar usuario.

## Conjunto de casos de uso

En la siguiente tabla se describen superficialmente los casos de usos, separados en dos bloques: el primer bloque, dedicado a los casos de uso de la planificación inicial; el segundo bloque centrado en los casos de uso añadidos en la quinta semana.

ID	Nombre	Descripción
<b>Primer bloque</b>		
CU-01	Añadir Evento de Municipio.	El usuario podrá añadir un nuevo evento de municipio en una fecha determinada, estando asignado a una condición meteorológica en tiempo real de la ubicación del municipio.
CU-02	Añadir Evento de Ruta Montaña.	El usuario podrá añadir un nuevo evento de montaña en una fecha determinada, estando asignado a una condición meteorológica en tiempo real de la ubicación de la montaña.
CU-03	Añadir Usuario.	El sistema permite al usuario registrarse con unas credenciales.
CU-04	Añadir una barra de búsqueda y filtrado de ubicaciones.	El sistema deberá incorporar una barra de búsqueda para buscar una ubicación con su tiempo meteorológico asignado.
CU-06	Consultar el tiempo detallado de una ubicación.	El usuario podrá visualizar el tiempo meteorológico de una ubicación específica en tiempo real
CU-07	Modificar un evento.	El usuario podrá modificar los datos de un evento, así como su nombre, descripción, fecha y ubicación.
CU-08	Eliminar un evento.	El usuario podrá borrar un evento creado y configurado previamente en la lista de eventos.
CU-09	Consultar el tiempo meteorológico en la ubicación actual.	El usuario podrá consultar los detalles del tiempo asociado a la localización/municipio más cercano.
CU-10	Modificar idioma y tema a modo oscuro	El usuario podrá realizar una serie de modificaciones en la aplicación para configurar esta a su gusto como cambiar el idioma, el tema a modo oscuro, etc.
CU-11	Consultar lista de eventos.	El usuario podrá visualizar el listado completo de eventos de municipios y montañas creados

		por su parte.
CU-12	Consultar un evento.	El usuario podrá visualizar en detalle los datos pertinentes de un evento de municipio o montaña.
CU-13	Iniciar sesión.	El sistema permite iniciar sesión con las credenciales asociadas a su cuenta de usuario.
<b>Segundo bloque</b>		
CU-05	Añadir preferencias desde el menú de AppBar (ajustes ...)	El sistema deberá incorporar un apartado de configuración que permite al usuario personalizar su experiencia con la aplicación, así como el modo oscuro, preferencias, etc.
CU-14	Cerrar sesión.	El sistema permite cerrar sesión a un usuario.
CU-15	Modificar usuario.	El usuario podrá modificar su nombre de usuario, contraseña asociadas a su cuenta en un apartado “perfil”.
CU-16	Eliminar usuario.	En el apartado “perfil”, el usuario podrá eliminar su cuenta de la aplicación y cerrar sesión.

## Requisitos no funcionales

Los requisitos no funcionales son aquellos que no hacen referencia a las funciones específicas del sistema, sino a las propiedades que este debe satisfacer. Suponen restricciones en su implementación. Para su definición vamos a agruparlos en varias categorías:

1. **Usabilidad:** Característica relacionada con el modo de interacción entre el usuario y la aplicación.

- **RNF 1:** La aplicación debe estar diseñada de manera que se adapte a la resolución al tamaño de pantalla de cualquier dispositivo móvil.
- **RNF 2:** Los usuarios habituados a las aplicaciones meteorológicas deben entender al instante el funcionamiento de la aplicación. El tiempo para una persona no habituada no debe ser superior a 5 minutos.
- **RNF 3:** La aplicación debe seguir los patrones establecidos por Google Material Design.
- **RNF 4:** El sistema proporcionará **feedback** constante al usuario durante el uso de la aplicación, bien informando de un error o confirmando una acción realizada.
- **RNF 5:** Para aquellas tareas que no finalicen inmediatamente se deberá mostrar una barra de progreso que informe de su estado actual.

- **RNF 6:** La aplicación debe ser apta para usuarios de habla inglesa o española, y permitir una fácil adaptación a cualquier otro idioma.

## 2. Eficiencia:

Capacidad para realizar adecuadamente una función.

- **RNF 7:** La aplicación no debe tardar más de 3 segundos en iniciarse.
- **RNF 8:** No demorar más de 6 segundos en la carga de información con las predicciones meteorológicas tras haber seleccionado una localidad, para que el usuario no se desespere.

## 3. Consistencia:

Característica que define a algo sólido y estable.

- **RNF 9:** La aplicación no debe bloquearse en ningún momento mientras se esté utilizando.

## 4. Integridad:

Capacidad para mostrar algo sin ser distorsionado.

- **RNF 10:** La información mostrada debe ser auténtica y coherente con los datos que la Agencia Española de Meteorología nos proporciona a través de su API.

## 5. Mantenibilidad:

- Facilidad para ser modificado.
- **RNF 11:** El código de la aplicación debe estar bien estructurado, utilizando patrones arquitectónicos, de modo que si hay que añadir o corregir alguna funcionalidad tardemos el menor tiempo posible en solucionarlo.

## 6. Compatibilidad:

Cualidad de un elemento que le permite poder trabajar con otro correctamente.

- **RNF 12:** Se espera que la aplicación sea compatible con el 85% de los dispositivos móviles existentes que utilicen Android, para ello usará la versión de Android 6.0.

## 7. Disponibilidad:

Probabilidad de que la aplicación falle.

- **RNF 13:** Se espera que la aplicación se encuentre disponible al menos el 99,95% de las veces que sean utilizadas por los usuarios, teniendo una probabilidad de fallo de como máximo un 0,05%.

## 8. Seguridad:

Capacidad que tiene la aplicación de proteger a sus usuarios.

- **RNF 14:** Se espera que la aplicación garantice la privacidad de los datos del usuario frente a amenazas externas de robo de datos o pérdida de estos gracias a mecanismos como el respaldo de estos mediante una copia de seguridad periódica y la restricción a su acceso a únicamente el administrador.

# Metodologías de desarrollo

## Proceso de desarrollo

Para llevar a cabo la realización del proyecto se ha utilizado el método denominado como **Proceso Unificado** (SCRUM), que consiste en una metodología ágil gracias a la cual se pueden desarrollar proyectos pudiendo realizar modificaciones en este fácilmente.

Así mismo, este proceso divide su desarrollo en **fases** utilizando el concepto de Casos de Uso para representar a los requisitos. Las fases se desarrollarán de forma incremental mediante una serie de **iteraciones**.

En este proceso, los **casos de uso** se crean e implementan a través de **cuatro fases**:

1. **Inicio:** Se especifica o declara el requisito en concreto, madurando dicha idea hasta un concepto más sólido.
2. **Elaboración:** El requisito pasa por un proceso de análisis y diseño, que amplían el concepto y cómo se debería implementar.
3. **Construcción:** El requisito se implementa en código software para ser utilizado en el programa.
4. **Transición:** Se prueba la versión final del requisito para saber si el código software es correcto, y en caso afirmativo se integra este junto con el resto en el programa.

## Planificación del proyecto

### Proceso de planificación

En la planificación de este proyecto, existirán **cuatro roles** que influyen de forma distinta la creación de casos de uso. Estos roles son:

1. **Jefe de proyecto:** Es el encargado principal de gestionar el desarrollo del proyecto.
2. **Desarrollador de software Senior:** Es un programador de Software relativamente experimentado.
3. **Desarrollador de software Junior:** Es un programador de software que cuenta con poca experiencia.
4. **Arquitecto de software:** Es un experto en la Arquitectura de Software.

Cada uno de estos roles tendrá que realizar una serie de tareas que les serán asignadas al comienzo del proyecto, las cuales ocupan cierta cantidad de tiempo designada durante la planificación.

## Distribución de casos de uso

Para seguir este proceso, los casos de uso especificados se desarrollan e implementan a lo largo de una serie de **iteraciones**, cuya duración es **de una semana**, en las que los cuatro roles distinguidos llevarán a cabo las distintas disciplinas de desarrollo de los casos de uso.

Estas disciplinas permitirán marcar la vida de desarrollo de los **casos de uso**, y estarán relacionados con las cuatro fases existentes en el método de Proceso Unificado. Las disciplinas son las siguientes:

- **Modelado del negocio** (Previa al inicio)
- **Requisitos** (Relacionada con la fase de inicio)
- **Análisis y Diseño** (Relacionada con la fase de elaboración)
- **Implementación** (Relacionada con la fase de construcción)
- **Test e Integración** (Relacionada con la fase de transición)
- **Despliegue** (Posterior a la transición)

De esta forma, cada rol realiza las siguientes disciplinas:

- **Jefe del proyecto:** Realiza la disciplina de Modelado del negocio (BM) y también tareas de gestión del proyecto.
- **Arquitecto del software:** Realiza las disciplinas Requisitos (R), Análisis y Diseño (A&D) e Implementación (Imp). También colaborará en situaciones extraordinarias en la disciplina de Test e Integración (T&I).
- **Desarrollador de Software Junior:** Realiza las disciplinas de Implementación (Imp), Test e integración (T&I) y Despliegue (D).
- **Desarrollador de Software Senior:** Realiza las disciplinas de Requisitos (R), Análisis y Diseño (A&D), Implementación (Imp), Test e integración (T&I) y Despliegue (D).

Para resumir y poder visualizar mejor esto, realizamos la siguiente tabla:

Tareas de rol:	BM	R	A&D	Imp	T&I	D
JP	X					
AS		X	X	X		
DS		X	X	X	X	X
DJ				X	X	X

## Análisis de distribución

Para calcular el tiempo total que tendrá disponible cada rol y cada caso de uso en número de horas se tendrá en cuenta una jornada laboral de 8 horas. Trabajando 5 días a la semana y teniendo en cuenta las 7 iteraciones (semanas), con lo que se obtiene la siguiente cantidad de tiempo:

	Jornada laboral	Jefe (20 %)	Arquitecto (100%)	Senior (100%)	Junior (50%)	Todos	Productividad (93%)	Tareas de proyecto (15%)	Tiempo casos de Uso
Semanal	40,00	8,00	40,00	40,00	20,00	108,00	100,44	15,066	85,374
Total	280,00	56,00	280,00	280,00	140,00	756,00	703,08	105,462	597,618

Esto nos da que la jornada laboral tiene **280 horas por semana**, de las cuales, según la **implicación** de cada **rol** en el proyecto se obtiene el tiempo que emplean:

- Jefe del proyecto: **20%**
- Arquitecto del Software: **100%**
- Desarrollador de software Senior: **100%**
- Desarrollador de software Junior: **50%**

Con esto, el tiempo total empleado en el proyecto es de **756 horas**.

Según las investigaciones proporcionadas por el periódico [diariodeSevilla](#), en España, existe una estimación de **productividad** general de un **93%** del tiempo trabajado.

*Tiempo productivo en el proyecto: 93% de 756 h ≈ 703,08 horas*

Con lo cual contamos con 703,08 horas de trabajo real, de las cuales el **15%** se dedica a **tareas de gestión del proyecto (Tareas de Project)**, y el resto del tiempo (**85%**) a la creación de los **casos de uso**, por lo que tenemos **597,618 horas** para realizar todos los casos de uso.

Tiempo Para CU por Iteración									
	Total	Business Modeling (BM)	Total	Definición de Requisitos (R)	Análisis y Diseño (A&D)	Implementación (Imp)	Test & Integración (T&I)	Despliegue (D)	Total
Porcentaje		5%		10,00%	20,00%	45,00%	20,00%	5,00%	100,00%
Tiempo bruto Semanal	85,37	4,27	81,11	8,11	16,22	36,50	16,22	4,06	81,11
Tiempo bruto Total	597,618	29,8809	567,7371	56,77371	113,54742	255,481695	113,54742	28,386855	567,7371
Tiempo por cada CU	49,8015	2,490075	47,311425	4,7311425	9,462285	21,29014125	9,462285	2,36557125	47,311425

En esta tabla se pueden observar los **porcentajes de tiempo** que se van a dedicar a cada uno de los **casos de uso**.

- **Definición de requisitos:** 10%
- **Análisis y Diseño:** 20%
- **Implementación:** 45%
- **Test e Integración:** 20%
- **Despliegue:** 5%

El tiempo que tenemos para cada caso de uso es calculado con:  $\frac{\text{Tiempo bruto total}}{\text{Nº Casos de Uso}}$

De este tiempo, tanto el **Jefe del proyecto** como el **Arquitecto del software** emplearán tiempo para las **tareas de Modelado de Negocio**, lo cual se ha estimado que ocupará un **5%** del tiempo.

Por lo cual las **Tareas de Gestión + Modelado de negocio** ocupan un total de

$$105,46 \text{ (gestión)} + 29,89 \text{ (BM)} = 135.35 \text{ horas}$$

Con lo que nos quedan **47,31 horas** para realizar el resto de fases de los CU.

Trabajador	Horas Semanales	Productividad	Horas Totales Semanales	Horas Totales	Gestión	BM	Horas CU Totales	Horas CU Semanales
Jefe	8,00	93%	7,44	52,08	22,2	29,88	0	0
Arquitecto	40,00	93%	37,2	260,4			260,40	37,20
Senior	40,00	93%	37,2	260,4			260,4	37,20
Junior	20,00	93%	18,6	130,2			130,2	18,60

En la tabla anterior podemos ver que los trabajadores, sumando la columna “Horas Totales Semanales”, disponen de **93 horas semanales en total** para realizar los casos de uso, así como las distintas cantidades de tiempo que dedicará cada rol en el proyecto.

## Planificación inicial

Se ha priorizado que sea el arquitecto de software el que tenga tiempo restante, como se puede ver en cada una de las siguientes tablas, para así dedicárselo a la gestión del proyecto.

El proyecto se desarrollará a lo largo de **7 iteraciones**, cada una de una semana de duración, las cuales se organizará de la siguiente manera:

Planificación							
CU1	R + A&D	Imp		T&I			D
CU2	R + A&D	Imp		T&I			D
CU3	R + A&D	Imp		T&I			D
CU4	R + A&D		Imp		T&I		D
CU6	R	A&D		Imp	T&I		D
CU7	R	A&D		Imp	T&I		D
CU8	R	A&D		Imp		T&I	D
CU9			R + A&D		Imp	T&I	D
CU10			R + A&D		Imp	T&I	D
CU11			R + A&D		Imp	T&I	D
CU12			R + A&D			Imp+T&I	D
CU13			R + A&D			Imp	T&I+D
Iteraciones	IT 1	IT 2	IT3	IT 4	IT 5	IT 6	IT 7
Fases	INICIO	ELABORACIÓN	CONSTRUCCIÓN				TRANSICIÓN

De esta forma, cada una de las iteraciones se han planificado en función de las distintas **fases** que pueden realizar cada uno de los **roles**.

Con lo cual la planificación por iteración quedaría de la siguiente manera:

#### Fase de Inicio:

ITERACIÓN 1												
Rol	Arquitecto				Desarrollador Senior					Desarrollador Junior		
Horas Iniciales	37,20				37,20					18,60		
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas					Horas		
CU1	4,73				9,46							
CU2	4,73				9,46							
CU3	4,73	5,37			4,09							
CU4	4,73	9,46										
CU6					4,73							
CU7					4,73							
CU8					4,73							
CU9												
CU10												
CU11												
CU12												
CU13												
Horas finales	3,45				0,00					18,60		

#### Fase de Elaboración:

ITERACIÓN 2												
Rol	Arquitecto				Desarrollador Senior					Desarrollador Junior		
Horas Iniciales	37,20				37,20					18,60		
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas					Horas		
CU1							2,69			18,60		
CU2							21,29					
CU3			8,07				13,22					
CU4												
CU6			9,46									
CU7			9,46									
CU8			9,46									
CU9												
CU10												
CU11												
CU12												
CU13												
Horas finales	0,75				0,00					0,00		

**Fases de Construcción:**

ITERACIÓN 3												
Rol	Arquitecto				Desarrollador Senior					Desarrollador Junior		
Horas Iniciales	37,20				37,20					18,60		
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas					Horas		
CU1												
CU2												
CU3												
CU4							2,69			18,60		
CU6												
CU7												
CU8												
CU9		4,73					9,46					
CU10		4,73					9,46					
CU11		4,73					9,46					
CU12		4,73	3,33				6,13					
CU13		4,73	9,46									
Horas finales	0,76				0,00					0,00		

ITERACIÓN 4												
Rol	Arquitecto				Desarrollador Senior					Desarrollador Junior		
Horas Iniciales	37,20				37,20					18,60		
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas					Horas		
CU1							9,46					
CU2							9,46					
CU3							9,46					
CU4												
CU6			2,69							18,60		
CU7			21,29									
CU8			12,47			8,82						
CU9												
CU10												
CU11												
CU12												
CU13												
Horas finales	0,75				0,00					0,00		

ITERACIÓN 5												
Rol	Arquitecto				Desarrollador Senior					Desarrollador Junior		
Horas Iniciales	37,20				37,20					18,60		
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas					Horas		
CU1												
CU2												
CU3												
CU4									9,46			
CU6									9,46			
CU7									9,46			
CU8												
CU9				2,69							18,60	
CU10				21,29								
CU11				12,47				8,82				
CU12												
CU13												
Horas finales	0,75				0,00					0,00		

#### Fases de Transición:

ITERACIÓN 6												
Rol	Arquitecto				Desarrollador Senior					Desarrollador Junior		
Horas Iniciales	37,20				37,20					18,60		
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas					Horas		
CU1												
CU2												
CU3												
CU4												
CU6												
CU7												
CU8										9,46		
CU9							0,32				9,14	
CU10							9,46					
CU11							9,46					
CU12				12,79			8,50	9,46				
CU13				21,29								
Horas finales	3,12				0,00					0,00		

ITERACIÓN 7												
Rol	Arquitecto				Desarrollador Senior					Desarrollador Junior		
Horas Iniciales	37,20				37,20					18,60		
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas					Horas		
CU1												2,37
CU2												2,37
CU3												2,37
CU4												2,37
CU6												2,37
CU7												2,37
CU8												2,37
CU9								0,36				2,01
CU10								2,37				
CU11								2,37				
CU12								2,37				
CU13								9,46	2,37			
Horas finales	37,20				17,90					0,00		

## Planificación modificada

Posteriormente, el **cliente** añadió cuatro casos de uso más para un total de **dieciséis**, uno de ellos **estructural** (**CU-05**), y los otros **tres no estructurales** (CU-14, CU-15 y CU-16). Puesto que no hay suficiente tiempo para realizar estas nuevas funcionalidades con la actual planificación del proyecto, se han añadido **2 iteraciones** más en la fase de **transición**, las cuales han permitido solucionar esta falta de tiempo.

Esto da lugar a una planificación compuesta por:

- Una iteración en la fase de **Inicio**
- Una iteración en la fase de **Elaboración**
- Cinco iteraciones en la fase de **Construcción**
- Dos iteraciones en la fase de **Transición**

De esta forma, la planificación por iteración después de realizar las modificaciones sería:

Planificación									
CU1	R + A&D	Imp	T&I				D		
CU2	R + A&D	Imp	T&I				D		
CU3	R + A&D	Imp	T&I				D		
CU4	R + A&D	Imp				T&I		D	
CU6	R	A&D	Imp			T&I		D	
CU7	R	A&D	Imp				T&I	D	
CU8	R	A&D	Imp				T&I	D	
CU9		R + A&D	Imp				T&I	D	
CU10		R + A&D	Imp				T&I	D	
CU11		R + A&D	Imp				T&I	D	
CU12		R + A&D			Imp			T&I + D	
CU13		R + A&D			Imp			T&I + D	
CU5				R + A&D	Imp	T&I		D	
CU14				R	A&D	Imp		T&I + D	
CU15				R	A&D	Imp		T&I + D	
CU16				R	A&D	Imp		T&I + D	
Iteraciones	IT 1	IT 2	IT3	IT 4	IT 5	IT 6	IT 7	IT 8	IT 9
Fases	INICIO	ELABORACIÓN	CONSTRUCCIÓN					TRANSICIÓN	

Debido a que las cuatro primeras iteraciones tienen la misma planificación que en la [inicial](#), son las iteraciones de la IT5 a la IT9 las que deben ser realizadas de nuevo, dejando la planificación modificada como la siguiente:

**Fases de Construcción:**

ITERACIÓN 5												
Rol	Arquitecto				Desarrollador Senior				Desarrollador Junior			
Horas Iniciales	37,20				37,20				18,60			
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas				Horas			
CU1												
CU2												
CU3												
CU4												
CU6												
CU7												
CU8												
CU9				2,69						18,60		
CU10				21,29								
CU11				12,47			8,82					
CU12												
CU13												
CU5					4,73	9,46						
CU14					4,73							
CU15					4,73							
CU16					4,73							
Horas finales	0,75				0,00				0,00			

### ITERACIÓN 6

Rol	Arquitecto				Desarrollador Senior				Desarrollador Junior			
Horas Iniciales	37,20				37,20				18,60			
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas				Horas			
CU1												
CU2												
CU3												
CU4												
CU6												
CU7												
CU8												
CU9												
CU10												
CU11												
CU12							21,29					
CU13			5,38				15,91					
CU5			2,69							18,60		
CU14		9,46										
CU15		9,46										
CU16		9,46										
Horas finales	0,75				0,00				0,00			

### ITERACIÓN 7

Rol	Arquitecto				Desarrollador Senior				Desarrollador Junior			
Horas Iniciales	37,20				37,20				18,60			
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas				Horas			
CU1												
CU2												
CU3												
CU4										9,46		
CU6							0,32			9,14		
CU7												
CU8												
CU9												
CU10												
CU11												
CU12												
CU13												
CU5							9,46					
CU14							21,29					
CU15			15,16				6,13					
CU16			21,29									
Horas finales	0,75				0,00				0,00			

**Fases de Transición:**

ITERACIÓN 8												
Rol	Arquitecto				Desarrollador Senior				Desarrollador Junior			
Horas Iniciales	37,20				37,20				18,60			
Fase	BM	R	A&D	Imp	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas				Horas			
CU1												2,36
CU2												2,36
CU3												2,36
CU4												
CU6												
CU7									9,46			
CU8									9,46			
CU9									9,46			
CU10									7,40			2,06
CU11												9,46
CU12												
CU13												
CU5												
CU14												
CU15												
CU16												
Horas finales	37,20				1,42				0,00			

ITERACIÓN 9													
Rol	Arquitecto				Desarrollador Senior				Desarrollador Junior				
Horas Iniciales	37,20				37,20				18,60				
Fase	BM	R	A&D	Imp	T&I	R	A&D	Imp	T&I	D	Imp	T&I	D
Caso de Uso	Horas				Horas				Horas				
CU1													
CU2													
CU3													
CU4										2,36			
CU6										2,36			
CU7										2,36			
CU8										2,36			
CU9										2,36			
CU10										2,36			
CU11										2,36			
CU12										2,36		9,46	
CU13									0,32	2,36		9,14	
CU5										2,36			
CU14				3,26					6,20	2,36			
CU15				9,46						2,36			
CU16				9,46						2,36			
Horas finales	15,02				0,00				0,00				

Como podemos ver, en esta **última iteración** ha sido necesario que el **arquitecto** ayude a los dos roles de desarrolladores a realizar ciertas tareas de **test e integración**.

También hay que tener en cuenta que todo el **tiempo que le sobra** al arquitecto es tiempo que dedica a **tareas de gestión Project** en conjunto con el jefe de proyecto.

Una vez finalizados todos los cálculos y cuadradas las horas del proyecto, se procede a **migrar el mismo a Microsoft Project**, una herramienta que nos permite detallar más información acerca de fechas, horas laborales, planificación, diagramas, tareas... etc.

Para ello, primero es necesario obtener la jornada laboral de cada uno de los roles basándose en su implicación en el desarrollo. Esta es la siguiente:

	Jornada laboral	Jefe (20 %)	Arquitecto (100%)	Senior (100%)	Junior (50%)	Todos	Productividad (93%)	Tareas de proyecto (15%)	Tiempo casos de Uso
Semanal	40,00	8,00	40,00	40,00	20,00	108,00	100,44	15,066	85,374
Total	280,00	56,00	280,00	280,00	140,00	756,00	703,08	105,462	597,618

- Jornada diaria del **Jefe de Proyecto** = 8 horas semanales/5 días = 1,6 horas.
- Jornada diaria de **Arquitecto Software** = 40 horas semanales/5 días = 8 horas.
- Jornada diaria de **Desarrollador Software Senior** = 40 h semanales/5 días = 8 horas.
- Jornada diaria de **Desarrollador Software Junior** = 20 h semanales/5 días = 4 horas

A estas horas les aplicamos el porcentaje de productividad de 93% para calcular el tiempo que realmente le dedican al proyecto:

- Jefe de Proyecto: 1,488h
- Arquitecto Software: 7,44h
- Desarrollador Software Senior: 7,44h
- Desarrollador Software Junior: 3,72

En Microsoft Project se han realizado **2 tipos de planificaciones**, una **planificación por iteraciones** y una posterior **planificación por casos de uso**.

		Modo de	Nombre de tarea	Trabajo	Duración	Comienzo	Fin	Predecesoras
1		➡	▫ Planificación APP ASEEE	882,64 horas	45 días	lun 31/08/20	vie 30/10/20	
2		➡	▫ Iteracion 01- Inicio	81,71 horas	5 días	lun 31/08/20	vie 04/09/20	
3		➡	▫ Management disciplines	6,5 horas	5 días	lun 31/08/20	vie 04/09/20	
4	🌐	➡	▫ Project Management	3,26 horas	5 días	lun 31/08/20	vie 04/09/20	
11	🌐	➡	▫ Environment	1,62 horas	4 días	mar 01/09/20	vie 04/09/20	
12	🌐	➡	▫ Prepare Environment for Project	0,54 horas	1 día	mar 01/09/20	mié 02/09/20	
	🌐	➡	Jefe de Proyecto	0,54 horas		mar 01/09/20	mié 02/09/20	
13	🌐	➡	▫ Prepare Environment and guidelines for an Iteration	0,54 horas	1 día	mié 02/09/20	jue 03/09/20	12
	🌐	➡	Jefe de Proyecto	0,54 horas		mié 02/09/20	jue 03/09/20	
14	🌐	➡	▫ Support Environment During an Iteration	0,54 horas	1 día	vie 04/09/20	vie 04/09/20	13
	🌐	➡	Arquitecto Software	0,54 horas		vie 04/09/20	vie 04/09/20	
15	🌐	➡	▫ Configuration & Change Management	1,62 horas	4 días	mar 01/09/20	vie 04/09/20	
16	🌐	➡	▫ Plan Project Configuration and Change Control	0,54 horas	1 día	mar 01/09/20	jue 03/09/20	
	🌐	➡	Jefe de Proyecto	0,54 horas		jue 03/09/20	jue 03/09/20	
17	🌐	➡	▫ Create Project CM Environments	0,54 horas	1 día	jue 03/09/20	vie 04/09/20	16
	🌐	➡	Jefe de Proyecto	0,54 horas		vie 04/09/20	vie 04/09/20	
18	🟠	➡	▫ Manage Environments during the Iteration	0,54 horas	1 día	vie 04/09/20	vie 04/09/20	17CC
	🌐	➡	Arquitecto Software	0,54 horas		vie 04/09/20	vie 04/09/20	
19		➡	▫ Use Cases Disciplines	75,21 horas	5 días	lun 31/08/20	vie 04/09/20	
20	🌐	➡	▫ Business Modeling	4,26 horas	4 días	lun 31/08/20	jue 03/09/20	
21	🌐	➡	▫ Assess Business Status	1,42 horas	1 día	lun 31/08/20	lun 31/08/20	
	🌐	➡	Jefe de Proyecto	1,42 horas		lun 31/08/20	lun 31/08/20	
22	🌐	➡	▫ Identify Business Processes	1,42 horas	2 días	lun 31/08/20	mar 01/09/20	21
	🌐	➡	Jefe de Proyecto	1,42 horas		lun 31/08/20	mar 01/09/20	
23	🟠	➡	▫ Design and Refine Business Process Realizations	1,42 horas	3 días	mar 01/09/20	jue 03/09/20	22CC
	🌐	➡	Jefe de Proyecto	1,42 horas		mar 01/09/20	jue 03/09/20	
24		➡	▫ Requirements	33,11 horas	3 días	lun 31/08/20	mié 02/09/20	
25	➡		Manage Changing Requirements	0 horas	1 día	lun 31/08/20	lun 31/08/20	
26		➡	▫ RE - UC 1	4,73 horas	1 día	lun 31/08/20	lun 31/08/20	
29		➡	▫ RE - UC 2	4,73 horas	2 días	lun 31/08/20	mar 01/09/20	
32		➡	▫ RE - UC 3	4,73 horas	2 días	lun 31/08/20	mar 01/09/20	
35		➡	▫ RE - UC 4	4,73 horas	2 días	mar 01/09/20	mié 02/09/20	
38		➡	▫ RE - UC 6	4,73 horas	1 día	lun 31/08/20	lun 31/08/20	
41		➡	▫ RE - UC 7	4,73 horas	2 días	lun 31/08/20	mar 01/09/20	
44		➡	▫ RE - UC 8	4,73 horas	1 día	mar 01/09/20	mar 01/09/20	
47	🌐	➡	▫ Analysis and Design	37,84 horas	5 días	lun 31/08/20	vie 04/09/20	
48		➡	▫ A&D - UC 1	9,46 horas	4 días	lun 31/08/20	jue 03/09/20	
51		➡	▫ A&D - UC 2	9,46 horas	5 días	lun 31/08/20	vie 04/09/20	
54		➡	▫ A&D - UC 3	9,46 horas	5 días	lun 31/08/20	vie 04/09/20	
57		➡	▫ A&D - UC 4	9,46 horas	5 días	lun 31/08/20	vie 04/09/20	
60		➡	▫ Iteracion 02 - Elaboracion	100,36 horas	5 días	lun 07/09/20	vie 11/09/20	
95		➡	▫ Iteracion 03 - Construcion	99,62 horas	5 días	lun 14/09/20	vie 18/09/20	
145		➡	▫ Iteración 04 - Construcion	100,4 horas	5 días	lun 21/09/20	vie 25/09/20	
189		➡	▫ Iteracion 05 - Construcion	100,4 horas	5 días	lun 28/09/20	vie 02/10/20	
230		➡	▫ Iteracion 06 - Construcion	100,4 horas	5 días	lun 05/10/20	vie 09/10/20	
264		➡	▫ Iteracion 07 - Construcion	100,4 horas	5 días	lun 12/10/20	vie 16/10/20	
308		➡	▫ Iteración 08 - Transicion	98,99 horas	5 días	lun 19/10/20	vie 23/10/20	
360		➡	▫ Iteracion 09 - Transicion	100,36 horas	5 días	lun 26/10/20	vie 30/10/20	

Como se puede observar se ha planificado el proyecto basándose en la intervención de los roles en cada iteración.

Por ello, en cada iteración se encuentran **2 secciones** dedicadas a:

- **Management disciplines:** Esta sección está dedicada para las tareas de gestión del proyecto, en ella trabajan tanto el jefe de proyecto como el arquitecto software, y se divide en 3 subsecciones, Project Management, Environment y Configuration & Change Management.
  - **Use Cases Disciplines:** Esta sección está utilizada para las distintas fases de los casos de uso, en ella trabaja todo el equipo y sus posibles subsecciones son las siguientes:
    - **Business Modeling:** Esta sección se encarga por completo del jefe del proyecto, entrando dentro de lo que conocemos como el modelado del negocio.
    - **Requirements:** Esta sección corresponde con los requerimientos de cada caso de uso.
    - **Analysis and Design:** Esta sección corresponde con el análisis y el diseño de cada caso de uso del proyecto.
    - **Implementation:** Esta subsección corresponde con la implementación de cada caso de uso
    - **Test y Integration:** En estas dos subsecciones corresponden con la fase de test e integración del Excel
    - **Deployment:** Esta subsección corresponde con el despliegue, y es la última que se realiza antes de que el caso de uso esté completamente desarrollado en la aplicación.

Cada iteración corresponde con una semana, por lo que en esta planificación se han ido repartiendo las horas de cada trabajador en cada una de las tareas que había que realizar para dicha integración, de tal forma que la repartición de horas, en función de los siguientes días y respetando su jornada laboral, se puede ver en la siguiente imagen:

28 sep '20				
L	M	X	J	V
20,08h	20,08h	20,08h	20,08h	20,08h
0h	0h	0,18h	1,48h	2,23h
20,08h	20,08h	19,9h	18,6h	17,85h
1,48h	1,48h	1,3h		
1,48h	0,65h			
1,48h	0,65h			
	0,83h	1,3h		
	0,83h	1,3h		
7,44h	7,44h	4,04h		
		3,4h	6,06h	
		3,4h	6,06h	
11,16h	11,16h	11,16h	12,54h	17,85h
6,41h	3,72h	3,72h	3,72h	3,72h
4,75h	7,44h	7,44h	1,66h	
0h	0h	0h	7,16h	14,13h

En cuanto a la planificación por casos de uso, esta es la siguiente:

		Modo de	Nombre de tarea	Trabajo	Duración	Comienzo	Fin	reg
1		▶ Planificación APP ASEE		882,64 horas	45 días	lun 31/08/20	vie 30/10/20	
2		▶ MANAGMENT		88,32 horas	45 días	lun 31/08/20	vie 30/10/20	
3		▶ Iteracion 01- Inicio		6,5 horas	5 días	lun 31/08/20	vie 04/09/20	
20		▶ Iteracion 02 - Elaboracion		3,85 horas	5 días	lun 07/09/20	vie 11/09/20	
21		▶ Management Disciplines		3,85 horas	5 días	lun 07/09/20	vie 11/09/20	
22		▶ Project Management		1,54 horas	4 días	lun 07/09/20	jue 10/09/20	
25		▶ Environment		1,54 horas	4 días	mar 08/09/20	vie 11/09/20	
28		▶ Configuration & Change Management		0,77 horas	3 días		mié vie 11/09/20	09/09/20
30		▶ Iteracion 03 - Construccion		3,9 horas	5 días	lun 14/09/20	vie 18/09/20	
40		▶ Iteración 04 - Construccion		3,89 horas	5 días	lun 21/09/20	vie 25/09/20	
50		▶ Iteracion 05 - Construccion		3,89 horas	5 días	lun 28/09/20	vie 02/10/20	
60		▶ Iteracion 06 - Construcion		3,89 horas	5 días	lun 05/10/20	vie 09/10/20	
70		▶ Iteracion 07 - Construccion		3,89 horas	5 días	lun 12/10/20	vie 16/10/20	
80		▶ Iteración 08 - Transicion		40,35 horas	5 días	lun 19/10/20	vie 23/10/20	
90		▶ Iteracion 09 - Transicion		18,16 horas	5 días	lun 26/10/20	vie 30/10/20	
100		▶ USES CASES		756,56 horas	45 días	lun 31/08/20	vie 30/10/20	
101		▶ General tasks		38,34 horas	43 días	lun 31/08/20	nié 28/10/20	
130		▶ UC01		44,94 horas	19 días	lun 31/08/20	jue 24/09/20	
131		▶ RE - UC 1		4,73 horas	1 día	lun 31/08/20	lun 31/08/20	
134		▶ A&D - UC 1		9,46 horas	4 días	lun 31/08/20	jue 03/09/20	
137		▶ IMP - UC 1		21,29 horas	5 días	lun 07/09/20	vie 11/09/20	
140		▶ INT - UC 1		4,73 horas	2 días	lun 21/09/20	nar 22/09/20	
143		▶ TEST - UC 1		4,73 horas	3 días	nar 22/09/20	jue 24/09/20	
146		▶ UC02		44,94 horas	20 días	lun 31/08/20	vie 25/09/20	
162		▶ UC03		44,94 horas	20 días	lun 31/08/20	vie 25/09/20	
178		▶ UC04		44,94 horas	33 días	lun 31/08/20	nié 14/10/20	
194		▶ UC05		44,94 horas	15 días	lun 28/09/20	vie 16/10/20	
210		▶ UC06		44,94 horas	35 días	lun 31/08/20	vie 16/10/20	
226		▶ UC07		44,94 horas	37 días	lun 31/08/20	nar 20/10/20	
242		▶ UC08		44,94 horas	37 días	nar 01/09/20	nié 21/10/20	
258		▶ UC09		44,94 horas	29 días	lun 14/09/20	jue 22/10/20	
274		▶ UC10		44,94 horas	30 días	lun 14/09/20	vie 23/10/20	
290		▶ UC11		44,94 horas	29 días	lun 14/09/20	jue 22/10/20	
306		▶ UC12		44,94 horas	32 días	nar 15/09/20	nié 28/10/20	
322		▶ UC13		44,16 horas	34 días	nar 15/09/20	vie 30/10/20	
338		▶ UC14		44,94 horas	22 días	lun 28/09/20	nar 27/10/20	
354		▶ UC15		44,9 horas	24 días	lun 28/09/20	jue 29/10/20	
370		▶ UC16		44,94 horas	25 días	lun 28/09/20	vie 30/10/20	
386		▶ Despliegues		37,76 horas	10 días	lun 19/10/20	vie 30/10/20	
387		▶ IT 08 - Package Product (Use Cases 1, 2, 3)		7,08 horas	5 días		lun vie 23/10/20	19/10/20
391		▶ IT 09 Package Product (Use Cases 4, 5, ... 15, 16)		30,68 horas	3 días		mié vie 30/10/20	28/10/20

Para realizarla, se ha tenido en cuenta la planificación previa por iteraciones y se han agrupado todas las intervenciones de los roles en esta por caso de uso.

En concreto, se compone de **3 secciones**:

- **Management:** Dedicada a la **gestión del proyecto** y el **modelado de negocio**, con las intervenciones del jefe del proyecto y el Arquitecto del Software
- **Use Cases:** Contiene las **disciplinas realizadas** de los casos de uso, en la fecha y hora que se realizaron y los roles que intervinieron.
- **Despliegues:** Contiene los **despliegues** de cada caso de uso, con la fecha y hora de su realización

## Análisis de la planificación

En primer lugar, la planificación ha requerido **más horas** de las que realmente se disponen para completarse, esto es debido a que en la planificación inicial, los cálculos se realizaron en torno a **7 semanas y 12 casos de uso**, obteniendo un total de **41,23 horas** por cada **caso de uso**. Sin embargo, fue necesario modificar la planificación del proyecto expandiendo 2 semanas su duracion.

Se ha decidido que el **Arquitecto del Software** se encargará de aquellas tareas que no debía realizar por su cualificación, pero que no se disponía del tiempo requerido para llevarlas a cabo. En este caso se trata de la disciplina de **Test e Integración**, pues a lo largo de las **últimas iteraciones** era necesario completarla para llevar a cabo el despliegue.

Considerando la falta de horas para realizar todo el trabajo, el **arquitecto software** priorizará trabajar en las disciplinas dedicadas a los **casos de uso**, y tras su realización, dedicará el tiempo sobrante de cada iteración en realizar **horas de gestión**. En la siguiente tabla se observan las **13,16 horas** de gestión que realizará el arquitecto tras finalizar tu contribución en las tareas dedicadas a los casos de uso.

Trabajador	Horas Gestión
Jefe	37,08
Arquitecto	55,22
Tiempo restante	13,16

Respecto a la distribución de horas de los distintos roles en las distintas disciplinas, el **Jefe de proyecto** es el que se encarga de todo el tiempo de **Business Modeling**, tras ello, dedica sus horas restantes a la **gestión**.

El **arquitecto de software** por tanto se dedica a las disciplinas de Requisitos, Análisis y Diseño, Implementación y extraordinariamente en la última iteración realizará Tests e Integración. Como se comentó previamente, invertirá las horas restantes que tenga en cada iteración en tareas de **gestión**.

El **desarrollador senior**, dedica todas sus horas a las siguientes disciplinas de los casos de uso: Requisitos, Análisis y Diseño, Implementación, Test e Integración y Despliegue.

Finalmente el **desarrollador junior**, dedica sus horas a las siguientes disciplinas de los casos de uso: Implementación, Test e Integración y Despliegue.

A continuación se muestra la distribución de horas que se ha seguido en las disciplinas de los casos de uso para cada uno de los **roles**, exceptuando el Jefe de proyecto.

#### Datos:

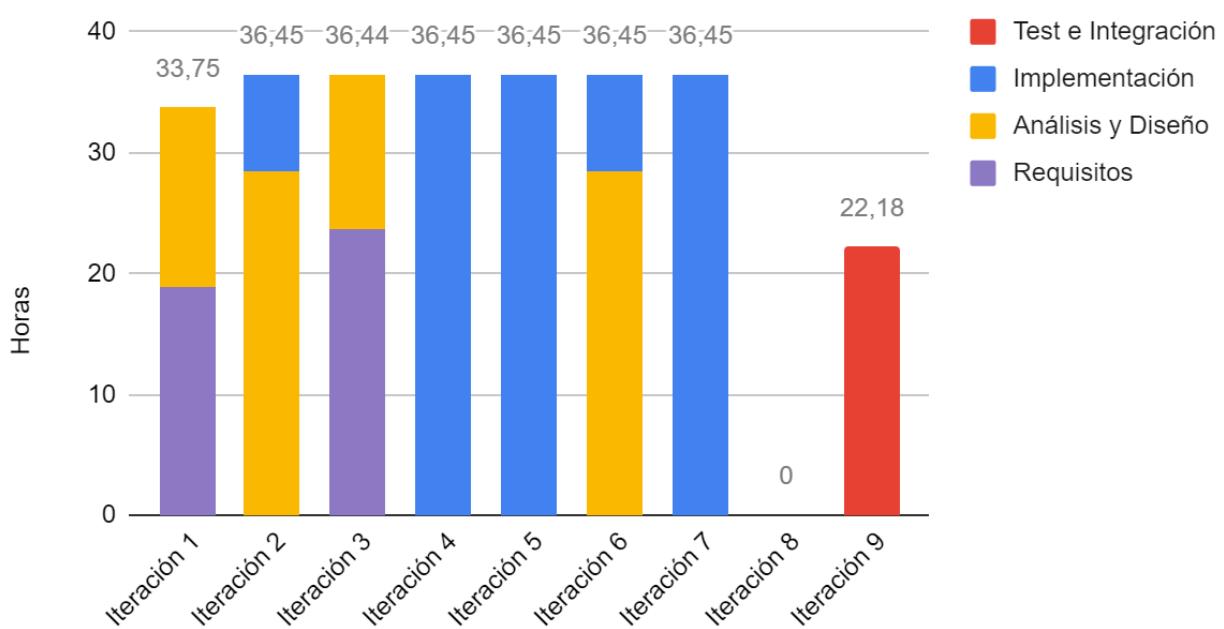
Iteraciones	Arquitecto Software				
	Requisitos	Análisis y Diseño	Implementación	Test e Integración	Horas totales
<i>Iteración 1</i>	18,92	14,83	0,00	0	33,75
<i>Iteración 2</i>	0,00	28,38	8,07	0	36,45
<i>Iteración 3</i>	23,65	12,79	0,00	0	36,44
<i>Iteración 4</i>	0,00	0,00	36,45	0	36,45
<i>Iteración 5</i>	0,00	0,00	36,45	0	36,45
<i>Iteración 6</i>	0,00	28,38	8,07	0	36,45
<i>Iteración 7</i>	0,00	0,00	36,45	0	36,45
<i>Iteración 8</i>	0,00	0,00	0,00	0	0,00
<i>Iteración 9</i>	0,00	0,00	0,00	22,18	22,18
<b>Total</b>	<b>42,57</b>	<b>84,38</b>	<b>125,49</b>	<b>22,18</b>	<b>274,62</b>

Iteraciones	Desarrollador Senior					
	Requisitos	Análisis y Diseño	Implementación	Test e Integración	Despliegue	Horas totales
<i>Iteración 1</i>	14,19	23,01	0,00	0,00	0,00	37,20
<i>Iteración 2</i>	0,00	0,00	37,20	0,00	0,00	37,20
<i>Iteración 3</i>	0,00	34,51	2,69	0,00	0,00	37,20
<i>Iteración 4</i>	0,00	0,00	8,82	28,38	0,00	37,20
<i>Iteración 5</i>	18,92	9,46	8,82	0,00	0,00	37,20
<i>Iteración 6</i>	0,00	0,00	37,20	0,00	0,00	37,20
<i>Iteración 7</i>	0,00	0,00	27,42	9,78	0,00	37,20
<i>Iteración 8</i>	0,00	0,00	0,00	35,78	0,00	35,78
<i>Iteración 9</i>	0,00	0,00	0,00	6,52	30,68	37,20
<b>Total</b>	<b>33,11</b>	<b>66,98</b>	<b>122,15</b>	<b>80,46</b>	<b>30,68</b>	<b>333,38</b>

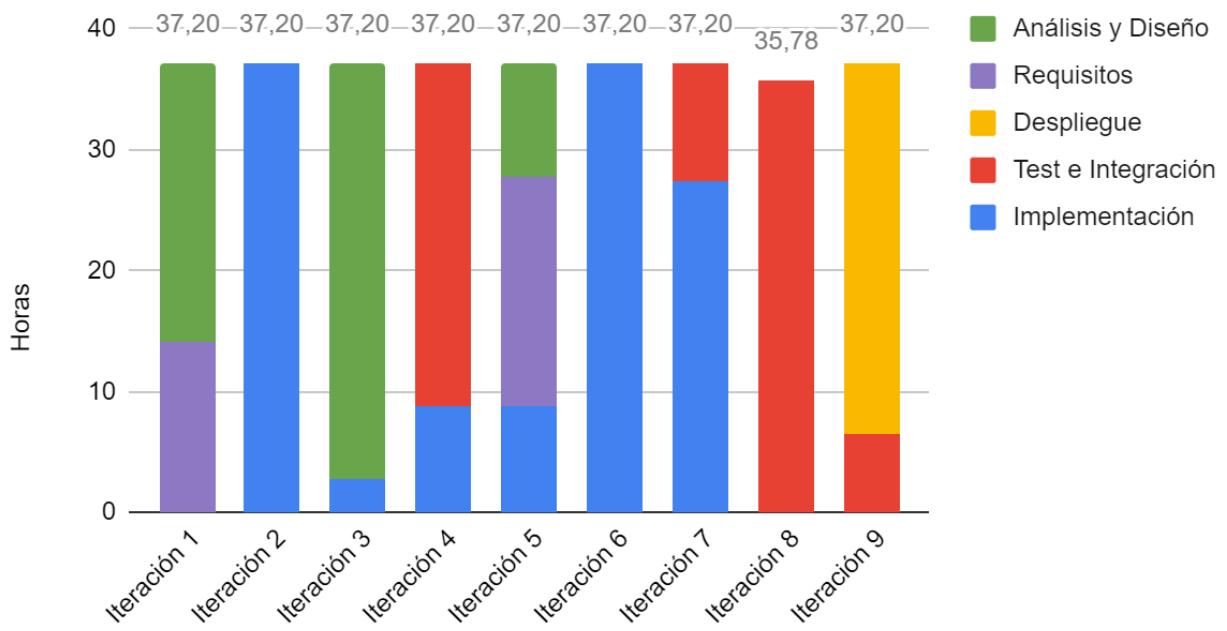
Desarrollador Junior				
Iteraciones	Implementación	Test e Integración	Despliegue	Horas totales
<i>Iteración 1</i>	0,00	0,00	0,00	0,00
<i>Iteración 2</i>	18,60	0,00	0,00	18,60
<i>Iteración 3</i>	18,60	0,00	0,00	18,60
<i>Iteración 4</i>	18,60	0,00	0,00	18,60
<i>Iteración 5</i>	18,60	0,00	0,00	18,60
<i>Iteración 6</i>	18,60	0,00	0,00	18,60
<i>Iteración 7</i>	0,00	18,60	0,00	18,60
<i>Iteración 8</i>	0,00	11,52	7,08	18,60
<i>Iteración 9</i>	0,00	18,60	0,00	18,60
<b>Total</b>	93,00	48,72	7,08	148,80

**Gráficos:**

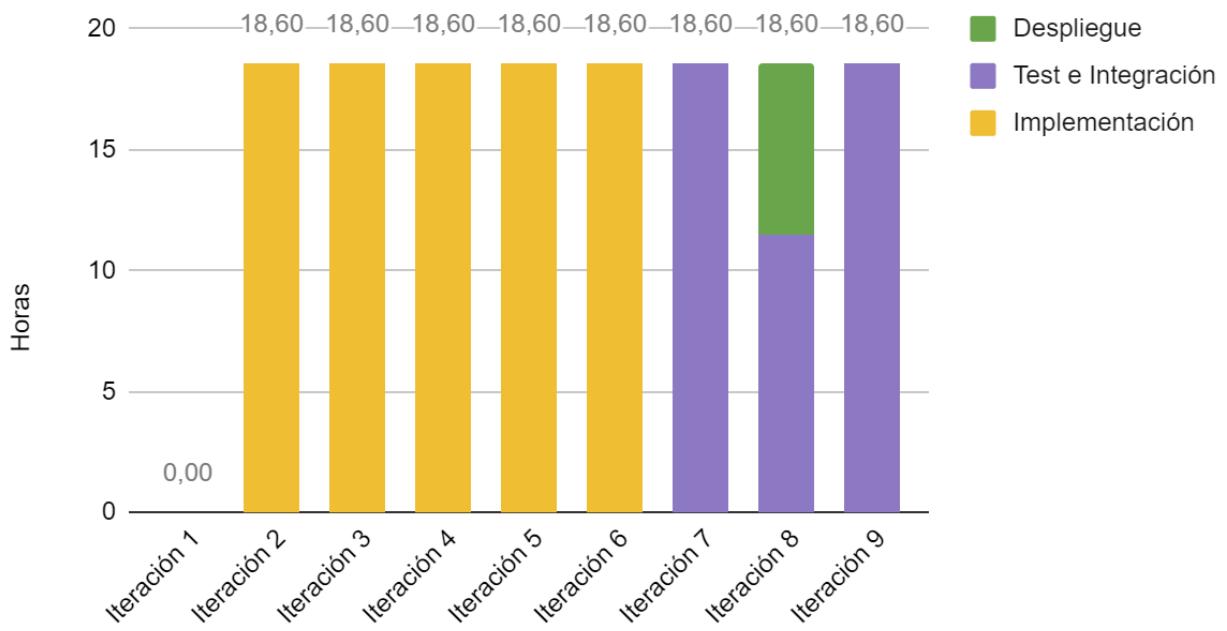
### Horas acumuladas de cada Disciplina en los Casos de Uso por iteración del Arquitecto software



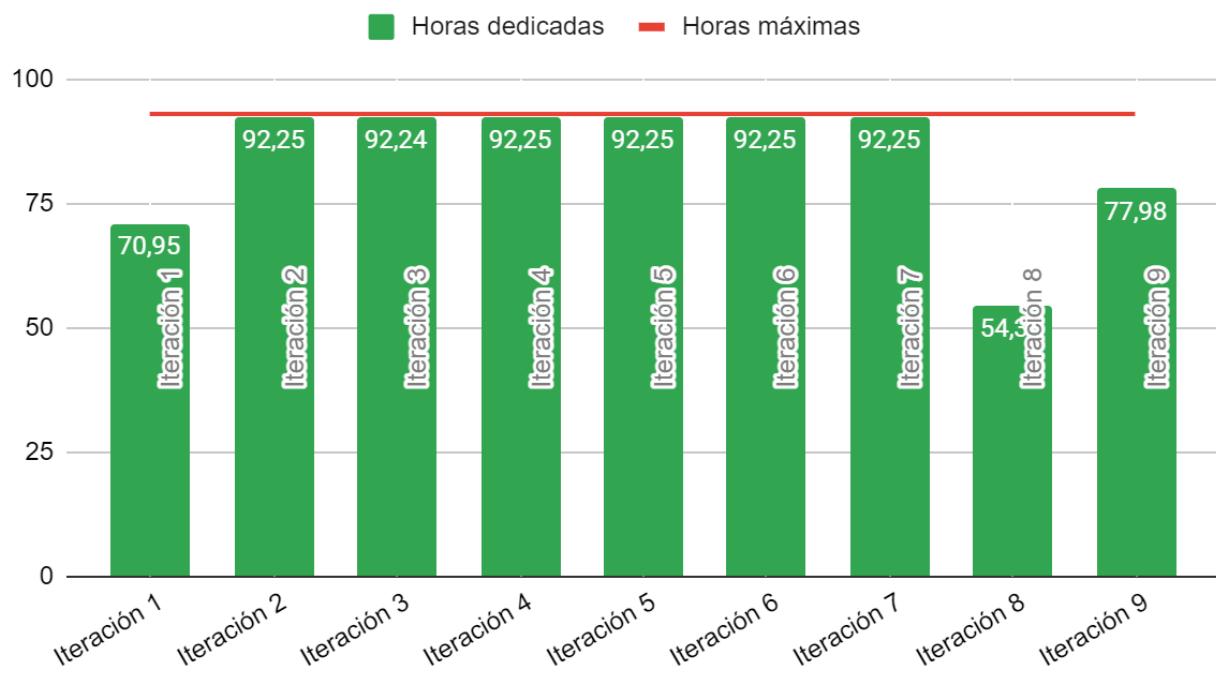
## Horas acumuladas de cada Disciplina en los Casos de Uso por iteración del Desarrollador Senior



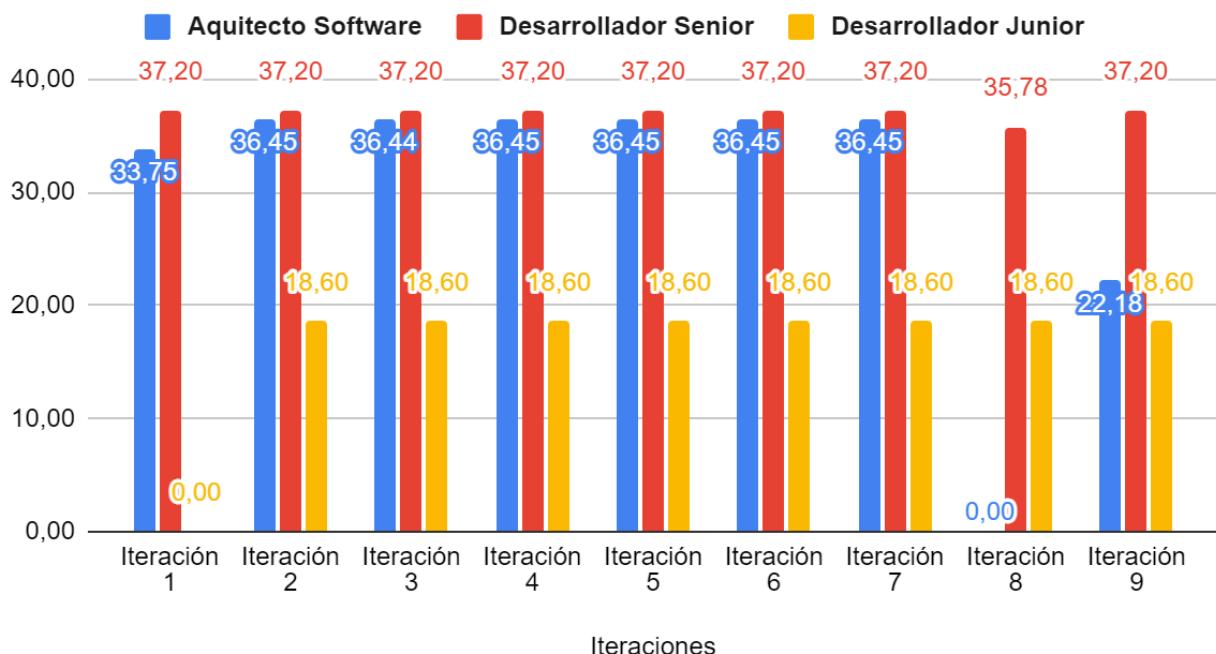
## Horas acumuladas de cada Disciplina por iteración del Desarrollador Junior



## Horas dedicadas en cada iteración



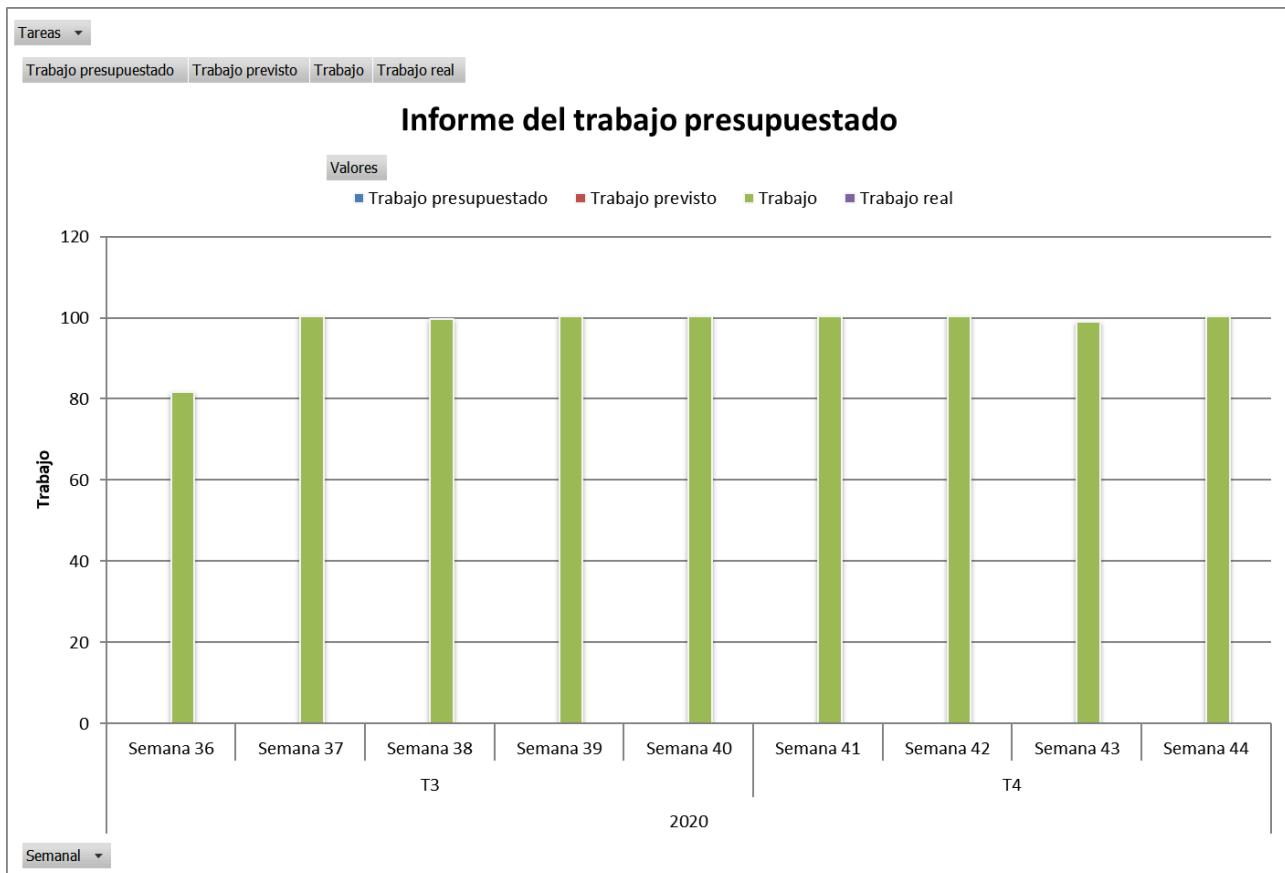
## Horas productivas por Rol



De esta forma, las **horas máximas** que se han dedicado al proyecto respecto de las que se **disponen** en primer lugar son las siguientes:

Iteraciones	Total	
	Horas máximas	Horas dedicadas
<b>Iteración 1</b>	93	70,95
<b>Iteración 2</b>	93	92,25
<b>Iteración 3</b>	93	92,24
<b>Iteración 4</b>	93	92,25
<b>Iteración 5</b>	93	92,25
<b>Iteración 6</b>	93	92,25
<b>Iteración 7</b>	93	92,25
<b>Iteración 8</b>	93	54,38
<b>Iteración 9</b>	93	77,98
<b>Total</b>	837	756,80

Con estos datos, se puede comprobar el **rendimiento** que se ha tenido por cada iteración:



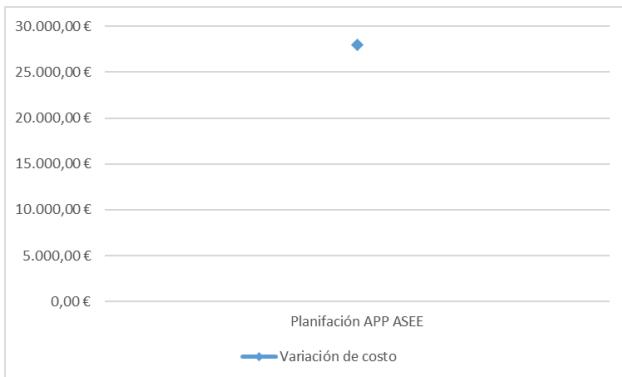
Se puede apreciar una bajada de trabajo en horas en la primera semana (semana 36), debido a que el desarrollador senior no puede trabajar en sus disciplinas por las dependencias de tareas. Respecto a las semanas posteriores, las horas de trabajo se mantienen constantes (100 horas).

A continuación, se muestra una gráfica de coste de los recursos de la planificación:

## SOBRECOSTOS

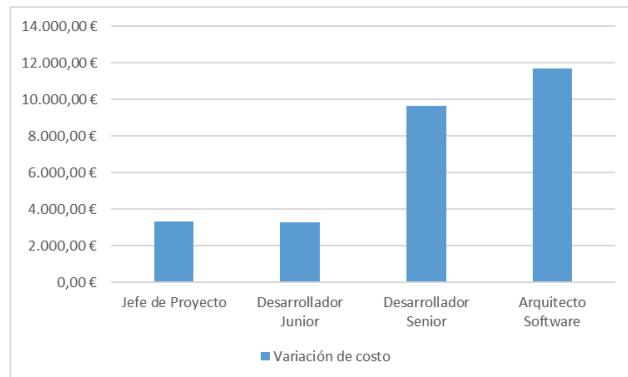
### VARIACIÓN DE COSTO DE TAREA

Variación de costos para todas las tareas de nivel superior en el proyecto.



### VARIACIÓN DE COSTO DE RECURSOS

Variación de costo de todos los recursos de trabajo.



Nombre	% completado	Costo	Costo de línea base	Variación de costo
Planificación APP ASEEE	0%	27.955,33 €	0,00 €	27.955,33 €

Nombre	Costo	Costo de línea base	Variación de costo
Jefe de Proyecto	3.325,54 €	0,00 €	3.325,54 €
Desarrollador Junior	3.273,60 €	0,00 €	3.273,60 €
Desarrollador Senior	9.667,94 €	0,00 €	9.667,94 €
Arquitecto Software	11.688,25 €	0,00 €	11.688,25 €

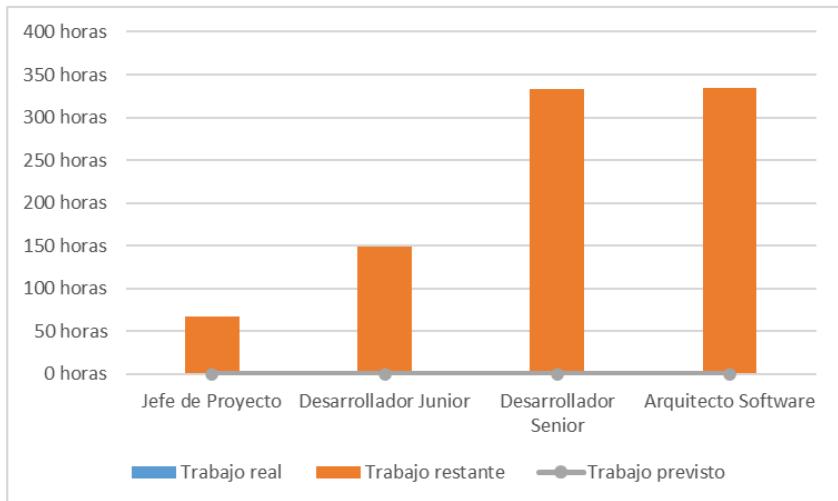
El coste total del proyecto se acerca a los 28.000 €, presupuesto que se desglosa según los recursos humanos. El jefe del proyecto cobra 3.300 € durante 9 semanas desde el inicio del proyecto; El desarrollador junior gana en torno 3.300 €, menos de la mitad del costo del desarrollador senior, el cual se encuentra cercano a los 10.000€. Por último, el presupuesto para cubrir el costo del arquitecto software es el más alto, 11.000 €.

A continuación, se muestra una descripción gráfica de las horas trabajadas por los roles en total durante toda la planificación del proyecto.

# VISIÓN GENERAL DE LOS RECURSOS

## ESTADÍSTICAS DE RECURSOS

Estado de trabajo de todos los recursos de trabajo.



## ESTADO DE LOS RECURSOS

Trabajo restante para todos los recursos de trabajo

Nombre	Comienzo	Fin	Trabajo restante
Jefe de Proyecto	lun 31/08/20	vie 30/10/20	66,51 horas
Desarrollador Junior	lun 07/09/20	vie 30/10/20	148,8 horas
Desarrollador Senior	lun 31/08/20	vie 30/10/20	333,38 horas
Arquitecto Software	lun 31/08/20	vie 30/10/20	333,95 horas

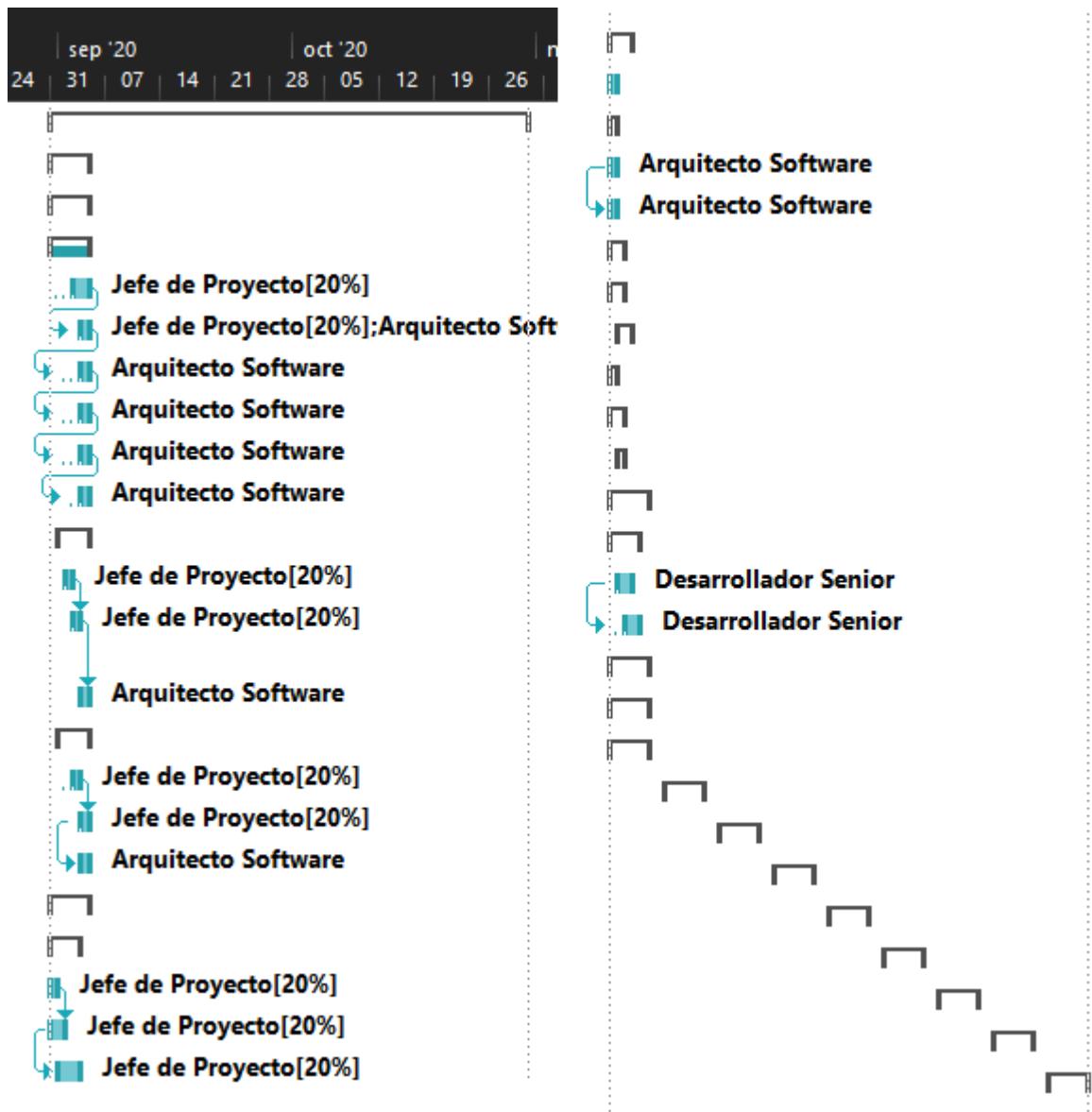
En este informe se puede observar el trabajo restante que les queda por hacer a cada uno de los trabajadores desde la fecha de comienzo **31/08/20** hasta el **30/10/20** como esta es una fecha pasada, el trabajo previsto está a 0, pero aun así, nos podemos hacer una idea de la carga de trabajo que soportará cada uno de los roles.

Como podemos ver cada rol trabaja en función del porcentaje de trabajo asignado al principio, el **jefe de proyecto** trabaja un **20%**, el **desarrollador junior** un **50%** y el **desarrollador senior** y el **arquitecto software** trabajaban el **100%** de la jornada laboral.

## Camino Crítico

Por último, dentro del análisis de la planificación nos encontramos ante el camino crítico. Este nos informará si hay algún punto crítico dentro del proyecto ya sea por culpa de horas mal planificadas o sobrecarga de trabajo.

Como se puede ver en estas dos fotos anteriores, hay zonas críticas debido a las horas, algunas acciones empiezan y acaban el mismo día (Business Modeling) la cual es predecesora de las disciplinas de los casos de usos.



## Configuración del seguimiento de la planificación

La planificación del proyecto se ha trasladado a **Jira**, una herramienta centrada en la **gestión de proyectos**, seguimiento de errores y del estado de desarrollo, la administración de tareas y la gestión de requisitos.

Así mismo, permite una profunda documentación sobre seguimientos y errores que contendría el proyecto y gestión de workflows ideales para proyectos software ágiles, como SCRUM. Es una herramienta flexible que facilita la gestión de proyectos pequeños, medianos o muy extensos.

Se ha decidido utilizar este software porque aporta un medio para la realización de la planificación de un proyecto de forma compartida y en tiempo real gracias a la plataforma web online . Así como porque presenta una interfaz relativamente sencilla e intuitiva. Además, los informes, las estadísticas y los progresos se muestran en **tiempo real**. De este modo, todos los implicados están siempre al día.

En cuanto a la sintaxis, cada caso de uso se ha creado como una **incidencia épica**, mientras que cada una de las tareas que se corresponden a cada caso de uso se han diseñado como **tareas**. Sin embargo, aquellas tareas que son compartidas por más de un rol contienen una **subtarea** con la contribución de cada rol.

Respecto a las iteraciones, estas vendrán definidas como un Sprint que contendrá todas las tareas implicadas en el progreso de dicha iteración. Por añadidura, todas las tareas de un caso de uso estarán enlazadas a la tarea épica del caso de uso susodicho y pertenecerán a un sprint concreto, correspondiente a una de las 9 iteraciones o semanas que durará el proyecto.

En la siguiente imagen se puede observar en detalle una tarea épica (caso de uso Añadir Evento Municipio):



ProyectoGA04 / PGA04-1

## CU01 - Añadir Evento de Municipio

[Editar](#) [Comentar](#) [Asignar](#) [Más](#) [Por Hacer](#) [En progreso](#) [Hecho](#)

[Detalles](#)

Tipo: [Épica](#)

Prioridad: [High](#)

Etiquetas: [CU01](#)

Nombre de épica: CU01

Estado:

**POR HACER** ([Ver Flujo de Trabajo](#))

Resolución: Sin resolver

[Descripción](#)

El usuario podrá añadir un nuevo evento de municipio en una fecha determinada, estando asignado a una condición meteorológica en tiempo real de la ubicación del municipio.

[Adjuntos](#)

...

Suelte los archivos para adjuntarlos o explorar.

[Incidencias en épica](#)

+

<a href="#">PGA04-2</a>	IT1 - UC01 - RE - Analyze the Problem Understanding Stakeholders Needs	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Marron</a>
<a href="#">PGA04-16</a>	IT1 - UC01 - RE - Define and Refine the System Requirements	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Marron</a>
<a href="#">PGA04-44</a>	IT1 - UC01 - A&D - Analyze Behavior and Define a Candidate Architecture	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Naranja</a>
<a href="#">PGA04-48</a>	IT1 - UC01 - A&D - Design the Database and Components	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Naranja</a>
<a href="#">PGA04-92</a>	IT2 - UC01 - IMP - Implement Components	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">Sin asignar</a>
<a href="#">PGA04-104</a>	IT2 - UC01 - IMP - Test Components	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Naranja</a>
<a href="#">PGA04-140</a>	IT4 - UC01 - INT - Integrate Each Subsystem	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Naranja</a>
<a href="#">PGA04-141</a>	IT4 - UC01 - INT - Integrate the System	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Naranja</a>
<a href="#">PGA04-142</a>	IT4 - UC01 - TEST - Test and Evaluate	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Naranja</a>
<a href="#">PGA04-143</a>	IT4 - UC01 - TEST - Achieve Acceptable Mission and Improve Test Assets	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Naranja</a>
<a href="#">PGA04-167</a>	IT8 - UC01 - D - Produce Deployment Unit	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Azul</a>
<a href="#">PGA04-169</a>	IT8 - UC01 - D - Manage Acceptance Test (At Installation Site)	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Azul</a>
<a href="#">PGA04-170</a>	IT8 - UC01 - D - Develop Support Material	<input checked="" type="checkbox"/> <a href="#">POR HACER</a> <a href="#">GA04Azul</a>

En la próxima imagen se puede ver una Tarea (**Incidencia**) con atribuciones de los roles (**Subtareas**):

ProyectoGA04 / PGA04-02

## IT2 - UC01 - IMP - Implement Components

Editar Comentar Asignar Más Por Hacer En progreso Hecho

**Detalles**

Tipo: Tarea Estado: POR HACER (Ver Flujo de Trabajo)  
Prioridad: High Resolución: Sin resolver  
Etiquetas: CU01 IMP IT02  
Enlace de épica: CU01  
Sprint: Iteración 2 - Elaboración

**Descripción**

Implementar los componentes

**Adjuntos**

Suelte los archivos para adjuntarlos o explorar.

**Enlace a Incidencias**

blocks

PGA04-104 IT2 - UC01 - IMP - Test Components POR HACER

is blocked by

PGA04-48 IT1 - UC01 - A&D - Design the Database and Components POR HACER

**Sub-Tareas**

1. IT2 - UC01 - IMP - Implement Components - DJ	POR HACER	GA04Azul	0%
2. IT2 - UC01 - IMP - Implement Components - DS	POR HACER	GA04Naranja	0%

En la imagen siguiente se puede ver una Iteración (**sprint**) en la que se desarrollan tareas:

Iteración 1 - Inicio		18 incidencias	Iniciar sprint
	...		
<input checked="" type="checkbox"/>	PGA04-2 IT1 - UC01 - RE - Analyze the Problem Understanding Stakeholders Needs		
<input checked="" type="checkbox"/>	PGA04-16 IT1 - UC01 - RE - Define and Refine the System Requirements		
<input checked="" type="checkbox"/>	PGA04-25 IT1 - UC02 - RE - Analyze the Problem Understanding Stakeholder Needs		
<input checked="" type="checkbox"/>	PGA04-27 IT1 - UC02 - RE - Define and Refine the System Requirements		
<input checked="" type="checkbox"/>	PGA04-29 IT1 - UC03 - RE - Analyze the Problem Understanding Stakeholder Needs		
<input checked="" type="checkbox"/>	PGA04-31 IT1 - UC03 - RE - Define and Refine the System Requirements		
<input checked="" type="checkbox"/>	PGA04-39 IT1 - UC04 - RE - Analyze the Problem Understanding Stakeholder Needs		
<input checked="" type="checkbox"/>	PGA04-43 IT1 - UC04 - RE - Define and Refine the System Requirements		
<input checked="" type="checkbox"/>	PGA04-44 IT1 - UC01 - A&D - Analyze Behavior and Define a Candidate Architecture		
<input checked="" type="checkbox"/>	PGA04-48 IT1 - UC01 - A&D - Design the Database and Components		
<input checked="" type="checkbox"/>	PGA04-51 IT1 - UC02 - A&D - Analyze Behavior and Define a Candidate Architecture		
<input checked="" type="checkbox"/>	PGA04-55 IT1 - UC02 - A&D - Design the Database and Components		
<input checked="" type="checkbox"/>	PGA04-4 IT1 - UC03 - A&D - Analyze Behavior and Define a Candidate Architecture		
<input checked="" type="checkbox"/>	PGA04-67 IT1 - UC03 - A&D - Design the Database and Components		
<input checked="" type="checkbox"/>	PGA04-68 IT1 - UC04 - A&D - Analyze Behavior and Define a Candidate Architecture		
<input checked="" type="checkbox"/>	PGA04-69 IT1 - UC04 - A&D - Design the Database and Components		
<input checked="" type="checkbox"/>	PGA04-209 IT1 - UC06 - RE - Analyze the Problem Understanding Stakeholder Needs		
<input checked="" type="checkbox"/>	PGA04-214 IT1 - UC06 - RE - Define and Refine the System Requirements		

Cada una de las incidencias (épicas o de tipo tarea) tienen asociada una prioridad. Las incidencias de los casos de usos estructurales vendrán definidas con una prioridad **High**, mientras que las incidencias relativas a los casos de uso no estructurales **Medium**.

En lo referente al nombre/título de cada tarea, se especificará en JIRA con la siguiente sintaxis:

*IT[Nº de Iteración] - UC[Nº de caso de uso] - [fase de disciplina] - [Rol encargado]*

A partir del contexto anterior, es momento de especificar que equipos de desarrollo se han planificado para realizar el proyecto y qué tareas se han repartido en cada equipo. Adicionalmente, se informará en qué consisten dichas tareas y cómo se realizan, estimando el tiempo que tienen el/los miembros del equipo asignados a hacerla a lo largo de las iteraciones.

Para llevar a cabo la planificación de cada disciplina de los casos de uso, estas disciplinas se han descompuesto en tareas de la siguiente manera:

- **Definición de requisitos:**
  - Análisis del problema tratando de comprender a los Stakeholders.
  - Definición y redefinición de los requisitos del sistema.
- **Análisis y Diseño:**
  - Analizar el comportamiento y definir una arquitectura software candidata.
  - Diseñar la base de datos y los componentes del sistema.
- **Implementación:**
  - Implementar los componentes.
  - Testear esos componentes (comprobar su funcionamiento).
- **Integración y testeo**
  - **Integración:**
    - Integrar cada subsistema al completo.
    - Integrar todos los subsistemas al sistema.
  - **Testeo:**
    - Testear y evaluar el resultado.
    - Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos.
- **Despliegue:**
  - Producir la Unidad de despliegue
  - Gestionar las pruebas de instalación del sistema en el sitio deseado
  - Desarrollar material de soporte.

Estas serán codificadas como **tareas**, en el caso de ser compartidas por más de un rol del equipo, contendrán unas **subtareas** con la contribución de cada rol. Para aquellas tareas que representan de manera uniforme un caso de uso, se definen como **épicas**.

En cuanto a las iteraciones, estas se contabilizarán con los **sprints**, que contendrán las distintas tareas que se realizarán dentro de dicha iteración.

De esta forma, todas las tareas de un caso de uso serán **asignadas a la Incidencia** correspondiente a dicho caso de uso, y pertenecerán a un sprint concreto, correspondiente a **una de las 9 iteraciones** que durará el proyecto.

Por tanto, dentro de Jira se ha seguido el siguiente procedimiento:

1. Se crea una épica para el caso de uso que se esté realizando
  - 1.1. Como nombre de épica se utiliza la nomenclatura CUXX.
  - 1.2. En el resumen se utiliza CUXX - Nombre\_del\_Caso.
  - 1.3. Se incluye una pequeña descripción del caso de uso.
  - 1.4. Se añade una etiqueta idéntica al nombre de la épica.
2. Dentro de cada épica, se crea una incidencia de tipo tarea por cada tarea de proyecto correspondiente al caso de uso
  - 2.1. Como resumen de la tarea se utiliza la nomenclatura ITXX - UCXX - Fase - Nombre\_de\_la\_Tarea
  - 2.2. Se asignan las etiquetas correspondientes al caso de uso, la iteración, la persona asignada a la tarea (en caso de que sólo sea una), y la fase
  - 2.3. Se asigna al responsable de la tarea, en caso de que sólo haya uno, y se añaden la cantidad de horas previstas.
    - 2.3.1. En caso de que la tarea no tenga un sólo responsable, se crean las subtareas necesarias para asignar el tiempo y las personas necesarias a dicha tarea, siguiendo el mismo procedimiento que para las tareas.
3. Se revisa que las tareas y las épicas están creadas correctamente
  - 3.1. Se comprueba que las horas son las adecuadas.
  - 3.2. Se comprueba que la tarea está asignada a la persona correspondiente.
  - 3.3. Se revisan las etiquetas y el sprint asignado.

## Señor Blanco (Equipo 1)

### Proceso seguido

El señor blanco ha liderado al equipo 1, donde cada rol es interpretado por los siguientes miembros:

- Jefe del proyecto (JP): Sr. Blanco
- Arquitecto Software (AS): Sr. Marrón
- Desarrollador Senior (DS): Sr. Naranja
- Desarrollador Junior (DJ): Sr. Azul

El **Sr. Blanco** (JP) se encargará de las tareas de modelado de negocio (BM) y gestión del proyecto software, en otras palabras, se ocupará de asignar las tareas a realizar a cada rol.

El **Sr. Marrón** (AS) se ocupará de las disciplinas de Requisitos, análisis y Diseño e Implementación. El **Sr. Naranja** (DS) se centra en todas las disciplinas que abarcan el desarrollo completo de cada caso de uso; desde la Definición de Requisitos, Análisis y Diseño, Implementación, Integración y Test y Despliegue. Por último, el **Sr. Azul** (DJ) ejecuta las disciplinas de Implementación, Test e Integración y Despliegue.

Respecto los casos de uso, el equipo 1 se encarga de la totalidad de disciplinas de los siguientes casos de uso (en negrita, los estructurales):

- **CU01**: Añadir Evento de Municipio.
- **CU02**: Añadir Evento de Ruta Montaña.
- **CU03**: Añadir Usuario.
- **CU04**: Añadir una barra de búsqueda y filtrado de ubicaciones.

Por último, las incidencias relacionadas con los cinco casos de uso estructurales dispondrán de una prioridad alta (**High**).

### Análisis del progreso

En este apartado se detalla la distribución de las tareas realizadas por cada rol a lo largo de las 9 iteraciones, según las disciplinas que puedan realizar cada uno de ellos.

Antes de dar principio, es imprescindible aclarar que el tiempo inicial disponible del arquitecto software es de 37.20 horas semanales. En el caso de que realice sus tareas y quede tiempo sobrante, este se lo dedicará a tareas de gestión.

## Planificación tareas CU01 - Añadir Evento de Municipio

Es el primer caso de uso estructural del proyecto; el usuario podrá añadir un nuevo evento de municipio en una fecha determinada, estando asignado a una condición meteorológica en tiempo real de la ubicación del municipio. Los roles que participarán en el desarrollo de este caso de uso son todos los miembros del equipo, exceptuando el Sr. Blanco (JP).

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 1** y finaliza en la **iteración 8**, con el despliegue del paquete de requisitos CU01, CU02 y CU03.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos** (RE) en la iteración 1 es el Arquitecto software (Sr. Marrón).

En cuanto al **Análisis y Diseño** (A&D), esta disciplina comienza en la IT1 y es completada por el Desarrollador Senior (Sr. Naranja).

La **Implementación** (Imp), como consecuencia de ser un caso de uso estructural, se realizará en la primera iteración de elaboración IT2. Los roles encargados de completar las tareas de esta disciplina son el Desarrollador Junior y el Desarrollador Senior (Sr. Azul y Sr. Naranja).

- La tarea de implementación “Implementar los componentes” es compartida por los DJ y DS.

El **Test e Integración** (T&I) se realiza en la iteración IT4 por el Desarrollador Senior (Sr. Naranja).

En última instancia, el **Despliegue** (D) se completa en la IT8 por el Desarrollador Junior (Sr. Azul) en el paquete de casos de uso CU01, CU02 y CU03.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT1	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT1	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	DS	4h 43m
		Diseñar la base de datos y los componentes del sistema		4h 43m
IT2	Imp	Implementar los componentes	DS	1h 20m
		Testear esos componentes (comprobar su funcionamiento)	DJ	2d 2h 36m
		Testear y evaluar el resultado	DS	1h 21m
IT4	I&T	Integrar cada subsistema al completo	DS	2h 21m
		Integrar todos los subsistemas al sistema		2h 22m
		Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos	DS	2h 21m
		Testear y evaluar el resultado		2h 22m
IT8	D	Producir la Unidad de despliegue	DJ	47m
		Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
		Desarrollar material de soporte		47m

## Planificación tareas CU02 - Añadir Evento de Ruta Montaña

Es el segundo caso de uso estructural del proyecto; El usuario podrá añadir un nuevo evento de montaña en una fecha determinada, estando asignado a una condición meteorológica en tiempo real de la ubicación de la montaña. Los roles que participarán en el desarrollo de este caso de uso son todos los miembros del equipo, exceptuando el Sr. Blanco (JP).

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 1** y finaliza en la **iteración 8**, con el despliegue del paquete de requisitos CU01, CU02 y CU03.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos** (RE) en la iteración 1 es el Arquitecto software (Sr. Marrón).

En cuanto al **Análisis y Diseño** (A&D), esta disciplina comienza en la IT1 y es completada por el Desarrollador Senior (Sr. Naranja).

La **Implementación** (Imp), como consecuencia de ser un caso de uso estructural, se realizará en la primera iteración de elaboración IT2. El rol encargado de completar las tareas de esta disciplina es el Desarrollador Senior (Sr. Naranja).

El **Test e Integración** (T&I) se realiza en la iteración IT4 por el Desarrollador Senior (Sr. Naranja).

En última instancia, el **Despliegue** (D) se completa en la IT8 por el Desarrollador Junior (Sr. Azul) en el paquete de casos de uso CU01, CU02 y CU03.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea		Rol	Tiempo
IT1	RE		Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
			Definición y redefinición de los requisitos del sistema		2h 22m
IT1	A&D		Analizar el comportamiento y definir una Arquitectura software candidata	DS	4h 43m
			Diseñar la base de datos y los componentes del sistema		4h 43m
IT2	Imp		Implementar los componentes	DS	1d 2h 38m
			Testear esos componentes (comprobar su funcionamiento)		1d 2h 39m
IT4	I&T	INT	Integrar cada subsistema al completo	DS	2h 21m
			Integrar todos los subsistemas al sistema		2h 22m
		TEST	Testear y evaluar el resultado	DS	2h 21m
			Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m
IT8	D		Producir la Unidad de despliegue	DJ	47m
			Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
			Desarrollar material de soporte		47m

## Planificación tareas CU03 - Añadir Usuario

Es el tercer caso de uso estructural del proyecto; El sistema permite al usuario registrarse con unas credenciales, de forma que en la base de datos se inserta un nuevo objeto con la información relativa al nuevo perfil creado en la aplicación. Con estas credenciales, el usuario podrá utilizar la funcionalidad de la aplicación. Los roles que participarán en el desarrollo de este caso de uso son todos los miembros del equipo, exceptuando el Sr. Blanco (JP).

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 1** y finaliza en la **iteración 8**, con el despliegue del paquete de requisitos CU01, CU02 y CU03.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos (RE)** en la iteración 1 es el Arquitecto software (Sr. Marrón).

En cuanto al **Análisis y Diseño (A&D)**, esta disciplina comienza en la IT1 y es completada por el Desarrollador Senior (Sr. Naranja), en añadidura del Arquitecto Software (Sr. Marrón) para completar las tareas.

- La tarea de Análisis y Diseño “Analizar el comportamiento y definir una Arquitectura software candidata” es compartida por los AS y DS.

La **Implementación (Imp)**, como consecuencia de ser un caso de uso estructural, se realizará en la primera iteración de elaboración IT2. Los encargados de completar las tareas de esta disciplina son el Desarrollador Senior (Sr. Naranja) y el Arquitecto Software (Sr. Marrón).

- La tarea de Implementación “Implementar los componentes” es compartida por los AS y DS.

El **Test e Integración (T&I)** se realiza en la iteración IT4 por el Desarrollador Senior (Sr. Naranja).

En última instancia, el **Despliegue (D)** se completa en la IT8 por el Desarrollador Junior (Sr. Azul) en el paquete de casos de uso CU01, CU02 y CU03.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT1	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT1	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	DS	4h 5m
		Diseñar la base de datos y los componentes del sistema	AS	38m
		Implementar los componentes	AS	38m
IT2	Imp	Testear esos componentes (comprobar su funcionamiento)	DS	6h 36m
		Integrar cada subsistema al completo	AS	1d 4m
IT4	I&T	Integrar todos los subsistemas al sistema	DS	6h 36m
		Testear y evaluar el resultado	DS	2h 21m
		Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m
		Producir la Unidad de despliegue	DJ	2h 21m
IT8	D	Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
		Desarrollar material de soporte		47m
				47m

## Planificación tareas CU04 - Añadir una barra de búsqueda y filtrado de ubicaciones

Es el cuarto caso de uso estructural del proyecto; El sistema deberá incorporar una barra de búsqueda para buscar una ubicación con su tiempo meteorológico asignado. Los roles que participarán en el desarrollo de este caso de uso son todos los miembros del equipo, exceptuando el Sr. Blanco (JP).

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 1** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos (RE)** en la iteración 1 es el Arquitecto software (Sr. Marrón).

En cuanto al **Análisis y Diseño (A&D)**, esta disciplina comienza en la IT1 y es completada por el Arquitecto Software (Sr. Marrón).

La **Implementación (Imp)**, como consecuencia de ser un caso de uso estructural, se realizará en la primera iteración de elaboración IT3. Los encargados de completar las tareas de esta disciplina son el Desarrollador Senior (Sr. Naranja) y el Desarrollador Junior (Sr. Azul).

- La tarea de Implementación “Implementar los componentes” es compartida por los DJ y DS.
- La tarea de Implementación “Testear esos componentes (comprobar su funcionamiento)” es compartida por los DJ y DS.

El **Test e Integración (T&I)** se realiza en la iteración IT7 por el Desarrollador Junior (Sr. Azul).

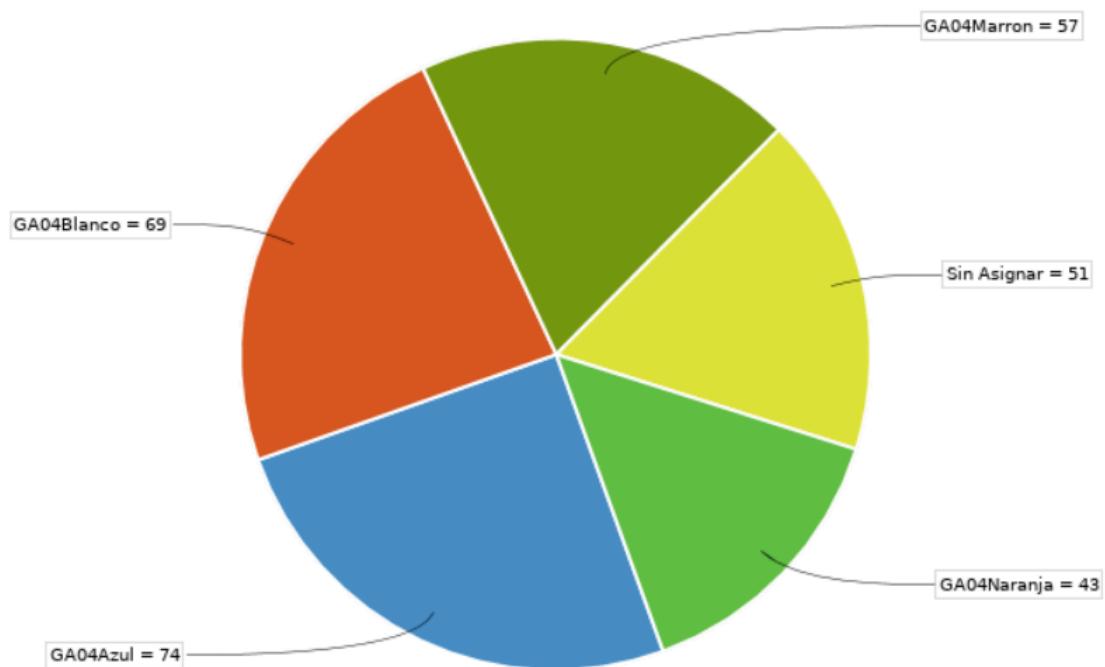
En última instancia, el **Despliegue (D)** se completa en la IT9 por el Desarrollador Junior (Sr. Azul) en el paquete de casos de uso CU04...CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo	
IT1	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m	
		Definición y redefinición de los requisitos del sistema		2h 22m	
IT1	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	AS	4h 43m	
		Diseñar la base de datos y los componentes del sistema		4h 43m	
IT3	Imp	Implementar los componentes	DS	1h 20m	
		Testear esos componentes (comprobar su funcionamiento)	DJ	1d 1h 18m	
			DS	1h 21m	
			DJ	1d 1h 18m	
IT7	I&T	INT	Integrar cada subsistema al completo	DJ	2h 21m
			Integrar todos los subsistemas al sistema		2h 22m
		TEST	Testear y evaluar el resultado	DJ	2h 21m
			Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m
IT9	D	Producir la Unidad de despliegue	DJ	47m	
		Gestionar las pruebas de instalación del sistema en el sitio deseado		47m	
		Desarrollar material de soporte		47m	

## Distribución de asignación de incidencias

**EXPLICA ESTO PQ NO SE SI AL NO TENER RESPONSABLE ESTÁN BIEN QUE SALGAN ALGUNAS SIN ASIGNAR**



### Tabla de datos

	Incidencias	%
GA04Azul	74	25%
GA04Blanco	69	23%
GA04Marron	57	19%
Sin Asignar	51	17%
GA04Naranja	43	14%

## Contribución de los líderes de equipo

Por último, sobre la planificación anterior establecida en Jira se crean dos gráficos: en el primero, se puede observar la contribución de cada usuario a las tareas asignadas a cada caso de uso.

**– FALTA – Necesito que todas las incidencias estén hechas :)**

## Señor Marrón (Equipo 2)

### Proceso seguido

El señor marrón ha liderado al equipo 2, donde cada rol es interpretado por los siguientes miembros:

- Jefe del proyecto (JP): Sr. Marrón
- Arquitecto Software (AS): Sr. Naranja
- Desarrollador Senior (DS): Sr. Azul
- Desarrollador Junior (DJ): Sr. Blanco

El **Sr. Marrón**(JP) se encargará de las tareas de modelado de negocio (BM) y gestión del proyecto software, en otras palabras, se ocupará de asignar las tareas a realizar a cada rol.

El **Sr. Naranja**(AS) se ocupará de las disciplinas de Requisitos, análisis y Diseño e Implementación. El **Sr. Azul**(DS) se centra en todas las disciplinas que abarcan el desarrollo completo de cada caso de uso; desde la Definición de Requisitos, Análisis y Diseño, Implementación, Integración y Test y Despliegue. Por último, el **Sr. Blanco**(DJ) ejecuta las disciplinas de Implementación, Test e Integración y Despliegue.

Respecto los casos de uso, el **equipo 2** se encarga de la totalidad de disciplinas de los siguientes casos de uso (en negrita, los estructurales):

- **CU05:** Añadir preferencias desde el menú de AppBar (ajustes...)
- CU06: Consultar el tiempo detallado de una ubicación.
- CU07: Modificar un evento.
- CU08: Eliminar un evento..

Por último, las incidencias relacionadas con el caso de uso estructural (CU05) dispondrás de una prioridad alta (**High**).

### Análisis del progreso

En este apartado se detalla la distribución de las tareas realizadas por cada rol a lo largo de las 9 iteraciones, según las disciplinas que puedan realizar cada uno de ellos.

Antes de dar principio, es imprescindible aclarar que el tiempo inicial disponible del arquitecto software es de 37.20 horas semanales. En el caso de que realice sus tareas y quede tiempo sobrante, este se lo dedicará a tareas de gestión.

## Planificación tareas CU05 - Añadir preferencias desde el menú de AppBar (ajustes...)

Se trata de un caso de uso estructural; el sistema deberá incorporar un apartado de configuración que permite al usuario personalizar su experiencia con la aplicación, así como el modo oscuro, preferencias...etc.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 5** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos del CU04 al CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos (RE)** en la iteración 5 es el Desarrollador Senior (Sr. Azul).

En cuanto al **Análisis y Diseño (A&D)**, esta disciplina comienza en la IT5 y es completada por el Desarrollador Senior (Sr. Azul).

La **Implementación (Imp)**, como consecuencia de ser un caso de uso estructural, se realizará cuanto antes posible, es decir, en la iteración 6. Los roles encargados de completar las tareas de esta disciplina son el Desarrollador Junior y el Arquitecto Software (Sr. Blanco y Sr. Azul).

- Las tareas de implementación “Implementar los componentes” y “Testear los componentes” son compartidas por los DJ y AS.

El **Test e Integración (T&I)** se realiza en la iteración IT7 por el Desarrollador Senior (Sr. Azul).

En última instancia, el **Despliegue (D)** se completa en la IT9 por el Desarrollador Senior (Sr. Azul) en el paquete de casos de uso CU04 al CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT5	RE	Análisis del problema tratando de comprender a los Stakeholders	DS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT5	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	DS	4h 43m
		Diseñar la base de datos y los componentes del sistema		4h 43m
IT6	Imp	Implementar los componentes	AS	1h 20m
			DJ	1d 1h 18m
		Testear esos componentes (comprobar su	AS	1h 20m

			funcionamiento)	DJ	1d 1h 18m
IT7	I&T	INT	Integrar cada subsistema al completo	DS	2h 21m
			Integrar todos los subsistemas al sistema		2h 22m
		TEST	Testear y evaluar el resultado	DS	2h 21m
			Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m
IT9	D		Producir la Unidad de despliegue	DS	47m
			Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
			Desarrollar material de soporte		47m

#### Planificación tareas CU06 - Consultar el tiempo detallado de una ubicación

Es el primer caso de uso no estructural del proyecto; El usuario podrá visualizar el tiempo meteorológico de una ubicación específica en tiempo real.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 1** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos (RE)** en la iteración 1 es el Desarrollador Senior (Sr. Azul).

En cuanto al **Análisis y Diseño (A&D)**, esta disciplina comienza en la IT2 y es completada por el Arquitecto Software (Sr. Naranja).

La **Implementación (Imp)**, se realizará en la segunda iteración de construcción IT4. Los encargados de completar las tareas de esta disciplina son el Desarrollador Junior (Sr. Blanco) y el Arquitecto Software (Sr. Naranja).

- La tarea de Implementación “Implementar los componentes” es compartida por los DJ y AS.

El **Test e Integración (T&I)** se realiza en la iteración IT7 por el Desarrollador Senior(Sr. Azul) y el Desarrollador Junior (Sr. Blanco).

- Todas las tareas de esta disciplina son compartidas por los DS y DJ.

En última instancia, el **Despliegue (D)** se completa en la IT9 por el Desarrollador Senior (Sr. Azul) en el paquete de casos de uso CU04...CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea		Rol	Tiempo
IT1	RE		Análisis del problema tratando de comprender a los Stakeholders	DS	2h 21m
			Definición y redefinición de los requisitos del sistema		2h 22m
IT2	A&D		Analizar el comportamiento y definir una Arquitectura software candidata	AS	4h 43m
			Diseñar la base de datos y los componentes del sistema		4h 43m
IT4	Imp		Implementar los componentes	AS	1h 20m
			Testear esos componentes (comprobar su funcionamiento)	DJ	2d 2h 36m
			Testear esos componentes (comprobar su funcionamiento)	AS	1h 21m
IT7	I&T	INT	Integrar cada subsistema al completo	DS	4m
				DJ	2h 16m
			Integrar todos los subsistemas al sistema	DS	4m
				DJ	2h 17m
		TEST	Testear y evaluar el resultado	DS	4m
				DJ	2h 16m
			Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos	DS	4m
				DJ	2h 17m
IT9	D		Producir la Unidad de despliegue	DS	47m
			Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
			Desarrollar material de soporte		47m

#### Planificación tareas CU07 - Modificar un evento

En este caso de uso no estructural del proyecto; El usuario podrá modificar los datos de un evento, así como su nombre, descripción, fecha y ubicación.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 1** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos (RE)** en la iteración 1 es el Desarrollador Senior (Sr. Azul).

En cuanto al **Análisis y Diseño (A&D)**, esta disciplina comienza en la IT2 y es completada por el Arquitecto Software (Sr. Naranja).

La **Implementación (Imp)**, se realizará en la segunda iteración de construcción IT4. El encargado de completar las tareas de esta disciplina es el Arquitecto Software (Sr. Naranja).

El **Test e Integración (T&I)** se realiza en la iteración IT8 por el Desarrollador Senior(Sr. Azul).

En última instancia, el **Despliegue (D)** se completa en la IT9 por el Desarrollador Senior (Sr. Azul) en el paquete de casos de uso CU04...CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea		Rol	Tiempo
IT1	RE		Análisis del problema tratando de comprender a los Stakeholders	DS	2h 21m
			Definición y redefinición de los requisitos del sistema		2h 22m
IT2	A&D		Analizar el comportamiento y definir una Arquitectura software candidata	AS	4h 43m
			Diseñar la base de datos y los componentes del sistema		4h 43m
IT4	Imp		Implementar los componentes	AS	1d 2h 38m
			Testear esos componentes (comprobar su funcionamiento)		1d 2h 39m
IT8	I&T	INT	Integrar cada subsistema al completo	DS	2h 21m
			Integrar todos los subsistemas al sistema		2h 22m
		TEST	Testear y evaluar el resultado		2h 21m
			Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m

IT9	D		Producir la Unidad de despliegue	DS	47m
			Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
			Desarrollar material de soporte		47m

Planificación tareas CU08 - Consultar el tiempo meteorológico en la ubicación actual.

En este caso de uso no estructural del proyecto; El usuario podrá consultar los detalles del tiempo asociado a la localización/municipio más cercano.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 1** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos (RE)** en la iteración 1 es el Desarrollador Senior (Sr. Azul).

En cuanto al **Análisis y Diseño (A&D)**, esta disciplina comienza en la IT2 y es completada por el Arquitecto Software (Sr. Naranja).

La **Implementación (Imp)**, se realizará en la segunda iteración de construcción IT4. Los encargados de completar las tareas de esta disciplina son el Desarrollador Senior (Sr. Azul) y el Arquitecto Software (Sr. Naranja).

- Ambas tareas de Implementación son compartidas por los DS y AS.

El **Test e Integración (T&I)** se realiza en la iteración IT8 por el Desarrollador Senior(Sr. Azul).

En última instancia, el **Despliegue (D)** se completa en la IT9 por el Desarrollador Senior (Sr. Azul) en el paquete de casos de uso CU04...CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT1	RE	Análisis del problema tratando de comprender a los Stakeholders	DS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT2	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	AS	4h 43m
		Diseñar la base de datos y los componentes del sistema		4h 43m

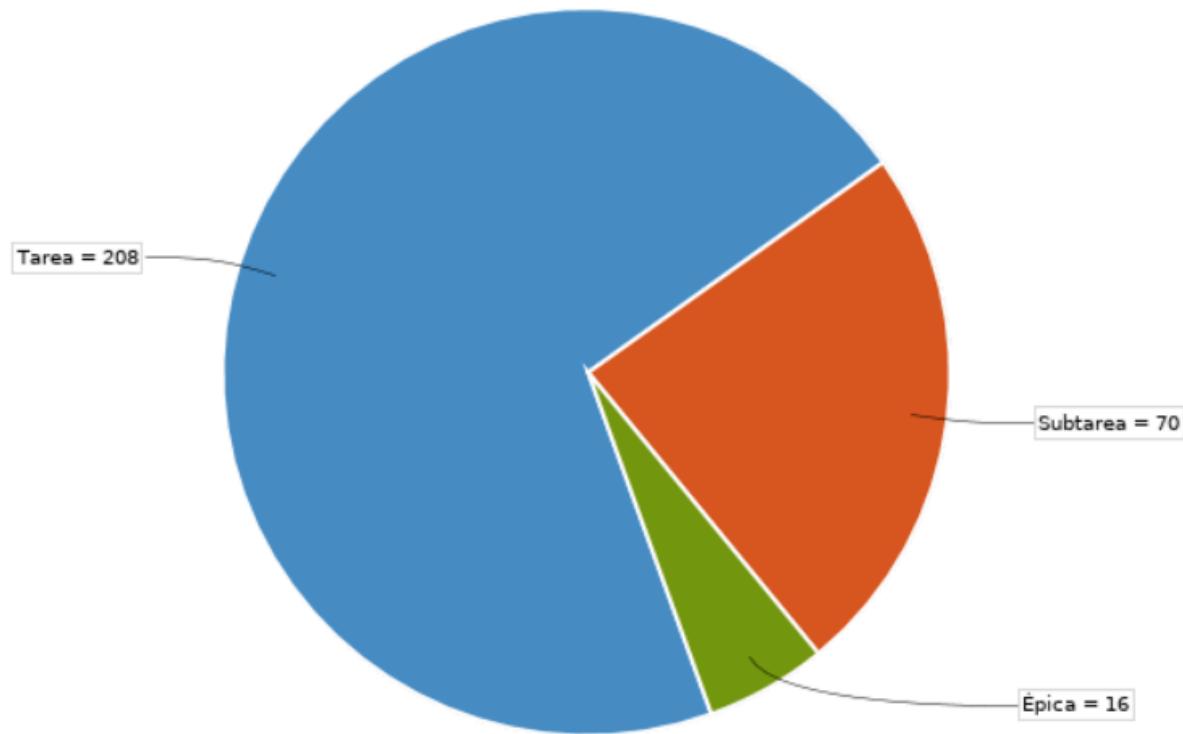
IT4	Imp		Implementar los componentes	AS	6h 13m	
				DS	4h 24m	
IT8	I&T	INT	Testear esos componentes (comprobar su funcionamiento)	AS	6h 13m	
				DS	4h 24m	
IT8		TEST	Integrar cada subsistema al completo	DS	2h 21m	
			Integrar todos los subsistemas al sistema		2h 22m	
		TEST	Testear y evaluar el resultado		2h 21m	
			Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m	
IT9	D		Producir la Unidad de despliegue	DS	47m	
			Gestionar las pruebas de instalación del sistema en el sitio deseado		47m	
			Desarrollar material de soporte		47m	

## Proporción de Épicas, Tareas y Subtareas

En el siguiente gráfico de tipo tarta se presenta la distribución existente en el proyecto de los distintos tipos de incidencias: épicas ( Casos de Uso), tareas y subtareas.

Se puede apreciar que la mayor parte del gráfico está compuesto por Tareas, seguido de un número alto de Subtareas.

Podemos observar que hay 16 incidencias de tipo Épica, las cuales referencian los casos de uso del proyecto.



A continuación se muestra el porcentaje de cada uno de estos tipos de incidencia:

### Tabla de datos

Incidencias		%
Tarea	208	70%
Subtarea	70	23%
Épica	16	5%

### Señor Naranja (Equipo 3)

Proceso seguido

El señor naranja ha liderado al equipo 3, donde cada rol es interpretado por los siguientes miembros:

- Jefe del proyecto (JP): Sr. Naranja
- Arquitecto Software (AS): Sr. Azul
- Desarrollador Senior (DS): Sr. Blanco
- Desarrollador Junior (DJ): Sr. Marrón

El **Sr. Naranja** (JP) se encargará de las tareas de modelado de negocio (BM) y gestión del proyecto software, en otras palabras, se ocupará de asignar las tareas a realizar a cada rol.

El **Sr. Azul** (AS) se ocupará de las disciplinas de Requisitos, análisis y Diseño e Implementación. El **Sr. Blanco** (DS) se centra en todas las disciplinas que abarcan el desarrollo completo de cada caso de uso; desde la Definición de Requisitos, Análisis y Diseño, Implementación, Integración y Test y Despliegue. Por último, el **Sr. Marrón** (DJ) ejecuta las disciplinas de Implementación, Test e Integración y Despliegue.

Respecto los casos de uso, el equipo 3 se encarga de la totalidad de disciplinas de los siguientes casos de uso (en negrita, los estructurales):

- **CU09:** Consultar tiempo meteorológico en la ubicación actual
- **CU10:** Modificar idioma y tema a modo oscuro
- **CU11:** Consultar lista de eventos.
- **CU12:** Consultar un evento.

Por último, las incidencias relacionadas con los cinco casos de uso estructurales dispondrán de una prioridad alta (**High**).

#### Análisis del progreso

En este apartado se detalla la distribución de las tareas realizadas por cada rol a lo largo de las 9 iteraciones, según las disciplinas que puedan realizar cada uno de ellos.

Antes de dar principio, es imprescindible aclarar que el tiempo inicial disponible del arquitecto software es de 37.20 horas semanales. En el caso de que realice sus tareas y quede tiempo sobrante, este se lo dedicará a tareas de gestión.

## Planificación tareas CU09 - Consultar tiempo meteorológico en la ubicación actual

Es el noveno caso de uso del proyecto; el usuario podrá consultar los detalles del tiempo asociado a la localización/municipio más cercano. Los roles que participarán en el desarrollo de este caso de uso son todos los miembros del equipo, exceptuando el Sr. Naranja (JP).

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 3** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos** (RE) en la iteración 3 es el Arquitecto software (Sr. Azul).

En cuanto al **Análisis y Diseño** (A&D), esta disciplina comienza en la IT3 y es completada por el Desarrollador Senior (Sr. Blanco).

La **Implementación** (Imp), esta disciplina comienza en la IT5 y es realizada por el Desarrollador Senior y el Arquitecto Software (Sr. Blanco y Sr. Azul).

- La tarea de implementación “Implementar los componentes” es compartida por los DS y AS.
- La tarea de implementación “Testear los componentes” es compartida por los DS y AS.

El **Test e Integración** (T&I) se realiza en la iteración IT8 por el Desarrollador Senior (Sr. Blanco).

En última instancia, el **Despliegue** (D) se completa en la IT9 por el Desarrollador Senior (Sr. Blanco) en el paquete de casos de uso CU04 ... CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT3	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT3	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	DS	4h 43m
		Diseñar la base de datos y los componentes del sistema		4h 43m
IT5	Imp	Implementar los componentes	DS	1d 1h 18m
		Testear esos componentes (comprobar su funcionamiento)	AS	1h 20m
			DS	1d 1h 18m
			AS	1h 21m
IT8	I&T	Integrar cada subsistema al completo	DS	2h 21m
		Integrar todos los subsistemas al sistema		2h 22m
		Testear y evaluar el resultado	DS	2h 21m
		Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m
IT9	D	Producir la Unidad de despliegue	DS	47m
		Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
		Desarrollar material de soporte		47m

## Planificación tareas CU10 - Modificar idioma y tema a modo oscuro

Es el décimo caso de uso del proyecto; el usuario podrá realizar una serie de modificaciones en la aplicación para configurar esta a su gusto como cambiar el idioma, el tema a modo oscuro, etc.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 3** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos** (RE) en la iteración 3 es el Arquitecto software (Sr. Azul).

En cuanto al **Análisis y Diseño** (A&D), esta disciplina comienza en la IT3 y es completada por el Desarrollador Senior (Sr. Blanco).

La **Implementación** (Imp), esta disciplina comienza en la IT5 y es realizada por el Arquitecto Software (Sr. Azul).

El **Test e Integración** (T&I) se realiza en la iteración IT8 por el Desarrollador Senior y el Desarrollador Junior (Sr. Blanco y Sr. Marrón).

En última instancia, el **Despliegue** (D) se completa en la IT9 por el Desarrollador Senior (Sr. Blanco) en el paquete de casos de uso CU04 ... CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT3	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT3	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	DS	4h 43m
		Diseñar la base de datos y los componentes del sistema		4h 43m
IT5	Imp	Implementar los componentes	AS	1d 2h 38m
		Testear esos componentes (comprobar su funcionamiento)	AS	1d 2h 38m
IT8	I&T	Integrar cada subsistema al completo	DS	1h 51m
			DJ	30m
		Integrar todos los subsistemas al sistema	DS	1h 51m
			DJ	31m
		Testear y evaluar el resultado	DS	1h 51m
			DJ	30m
IT9	D	Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos	DS	1h 51m
			DJ	31m
		Producir la Unidad de despliegue	DS	47m
		Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
		Desarrollar material de soporte		47m

## Planificación tareas CU11 - Consultar lista de eventos

Es el undécimo caso de uso del proyecto; el usuario podrá visualizar el listado completo de eventos de municipios y montañas creados por su parte.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 3** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos** (RE) en la iteración 3 es el Arquitecto software (Sr. Azul).

En cuanto al **Análisis y Diseño** (A&D), esta disciplina comienza en la IT3 y es completada por el Desarrollador Senior (Sr. Blanco).

La **Implementación** (Imp), esta disciplina comienza en la IT5 y es realizada por el Desarrollador Senior y el Arquitecto Software (Sr. Blanco y Sr. Azul).

- La tarea de implementación “Implementar los componentes” es compartida por los DS y AS.
- La tarea de implementación “Testear los componentes” es compartida por los DS y AS.

El **Test e Integración** (T&I) se realiza en la iteración IT8 por el Desarrollador Junior (Sr. Marrón).

En última instancia, el **Despliegue** (D) se completa en la IT9 por el Desarrollador Senior (Sr. Blanco) en el paquete de casos de uso CU04 ... CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT3	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT3	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	DS	4h 43m
		Diseñar la base de datos y los componentes del sistema		4h 43m
IT5	Imp	Implementar los componentes	DS	4h 24m
		Testear esos componentes (comprobar su funcionamiento)	AS	6h 13m
			DS	4h 24m
			AS	6h 13m
IT8	I&T	Integrar cada subsistema al completo	DJ	2h 21m
		Integrar todos los subsistemas al sistema		2h 22m
		Testear y evaluar el resultado	DJ	2h 21m
		Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m
IT9	D	Producir la Unidad de despliegue	DS	47m
		Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
		Desarrollar material de soporte		47m

## Planificación tareas CU12 - Consultar un evento

Es el duodécimo caso de uso del proyecto; el usuario podrá visualizar en detalle los datos pertinentes de un evento de municipio o montaña.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 3** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04...CU16.

El encargado de realizar las tareas correspondientes a la **Definición de Requisitos (RE)** en la iteración 3 es el Arquitecto software (Sr. Azul).

En cuanto al **Análisis y Diseño (A&D)**, esta disciplina comienza en la IT3 y es realizada por el Desarrollador Senior y el Arquitecto Software (Sr. Blanco y Sr. Azul).

- La tarea de Análisis y diseño “Analizar el comportamiento y definir una Arquitectura software candidata” es compartida por los DS y AS.

La **Implementación (Imp)**, comienza en la IT6 y es realizada por el Desarrollador Senior (Sr. Blanco).

El **Test e Integración (T&I)** se realiza en la iteración IT9 por el Desarrollador Junior (Sr. Marrón).

En última instancia, el **Despliegue (D)** se completa en la IT9 por el Desarrollador Senior (Sr. Blanco) en el paquete de casos de uso CU04 ... CU16.

En la siguiente tabla se puede ver la distribución de manera resumida:

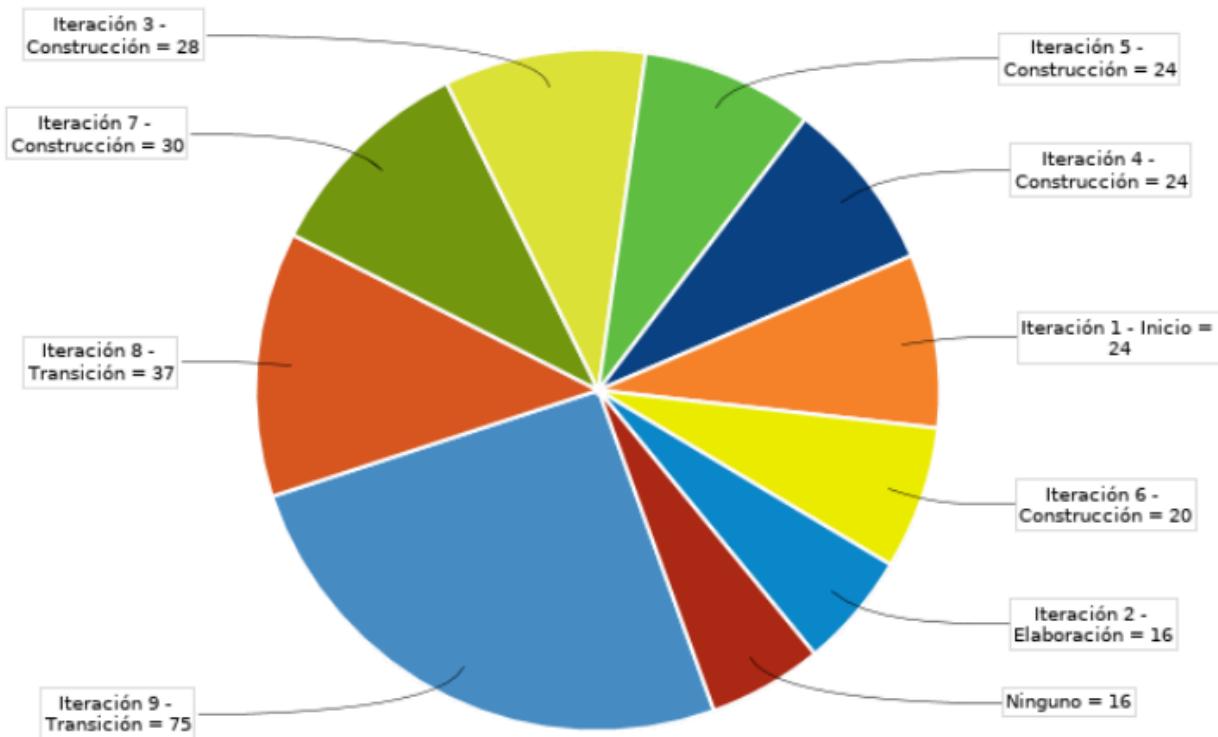
Iteración	Disciplina	Tarea	Rol	Tiempo
IT3	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT3	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	DS	3h 3m
		Diseñar la base de datos y los componentes del sistema	AS	3h 19m
IT6	Imp	Implementar los componentes	DS	1d 2h 38m
		Testear esos componentes (comprobar su funcionamiento)	DS	1d 2h 38m
IT9	I&T	Integrar cada subsistema al completo	DJ	2h 21m
		Integrar todos los subsistemas al sistema		2h 22m
		Testear y evaluar el resultado	DJ	2h 21m
		Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos		2h 22m
IT9	D	Producir la Unidad de despliegue	DS	47m
		Gestionar las pruebas de instalación del sistema en el sitio deseado		47m
		Desarrollar material de soporte		47m

## Distribución de incidencias por iteración

En el siguiente gráfico de tipo tarta, se puede ver la distribución de incidencias a lo largo de cada una de las iteraciones.

Se puede apreciar que las iteraciones que más incidencias contienen son las iteraciones 8 y 9 puesto que en ellas se realizan las tareas de despliegue de los casos de uso.

Las tareas que no están asignadas a ninguna iteración corresponden a las 16 tareas Épicas que son los casos de uso.



A continuación se muestra una tabla con los porcentajes de cada iteración:

## Tabla de datos

	Incidencias	%
Iteración 9 - Transición	75	25%
Iteración 8 - Transición	37	12%
Iteración 7 - Construcción	30	10%
Iteración 3 - Construcción	28	9%
Iteración 5 - Construcción	24	8%
Iteración 4 - Construcción	24	8%
Iteración 1 - Inicio	24	8%
Iteración 6 - Construcción	20	6%
Iteración 2 - Elaboración	16	5%
Ninguno	16	5%

## Contribución de los líderes de equipo

Por último, sobre la planificación anterior establecida en Jira se crean dos gráficos: en el primero, se puede observar la contribución de cada usuario a las tareas asignadas a cada caso de uso.

## Señor Azul (Equipo 1)

### Proceso seguido

El señor azul ha liderado al equipo 4, donde cada rol es interpretado por los siguientes miembros:

- Jefe del proyecto (JP): Sr. Azul
- Arquitecto Software (AS): Sr. Blanco
- Desarrollador Senior (DS): Sr. Marrón
- Desarrollador Junior (DJ): Sr. Naranja

El **Sr. Azul** (JP) se encargará de las tareas de modelado de negocio (BM) y gestión del proyecto software, en otras palabras, se ocupará de asignar las tareas a realizar a cada rol.

El **Sr. Blanco** (AS) se ocupará de las disciplinas de Requisitos, análisis y Diseño e Implementación.

El **Sr. Marrón** (DS) se centra en todas las disciplinas que abarcan el desarrollo completo de cada caso de uso; desde la Definición de Requisitos, Análisis y Diseño, Implementación, Integración y Test y Despliegue. Por último,

El **Sr. Naranja** (DJ) ejecuta las disciplinas de Implementación, Test e Integración y Despliegue.

Respecto los casos de uso, el equipo 4 se encarga de la totalidad de disciplinas de los siguientes casos de uso:

- **CU-13:** Iniciar sesión.
- **CU-14:** Cerrar sesión.
- **CU-15:** Modificar usuario.
- **CU-16:** Eliminar usuario.

### Análisis del progreso

En este apartado se detalla la distribución de las tareas realizadas por cada rol a lo largo de las 9 iteraciones, según las disciplinas que puedan realizar cada uno de ellos.

Antes de empezar, es imprescindible aclarar que el tiempo inicial disponible del arquitecto software es de 37.20 horas semanales. En el caso de que realice sus tareas y quede tiempo sobrante, este se lo dedicará a tareas de gestión.

#### Planificación tareas CU13 - Iniciar sesión

En este caso de uso el usuario podrá iniciar sesión dentro de la aplicación, para lo cual deberá haber creado previamente su usuario.

Todos los roles participarán de manera conjunta para el desarrollo de este caso de uso.

El desarrollo de este caso de uso comienza con su Definición de Requisitos (RE) en la **iteración 3** y finaliza en la **iteración 9**, con el despliegue del paquete de requisitos CU04-16.

El encargado de la **Definición de Requisitos** (RE) en la **iteración 3** es el Arquitecto software (Sr. Blanco).

En cuanto al **Análisis y Diseño** (A&D), esta disciplina comienza en la **iteración 3** y es completada por el Arquitecto software (Sr. Blanco).

La **Implementación** (Imp) se realizará en la **iteración 6**. Los roles encargados de completar las tareas de esta disciplina son el Desarrollador Senior (Sr. Marrón) y el Arquitecto software (Sr. Blanco).

El **Test e Integración** (T&I) se realiza en la **iteración 9** y es completada por el Desarrollador Senior (Sr. Marrón) y el Desarrollador junior (Sr. Naranja).

Por último, el **Despliegue** (D) se completa en la **iteración 9** por el Desarrollador Senior (Sr. Marrón) en el paquete de casos de uso CU04-16.

En la siguiente tabla se puede ver la distribución de manera resumida:

Iteración	Disciplina	Tarea	Rol	Tiempo
IT3	RE	Análisis del problema tratando de comprender a los Stakeholders	AS	2h 21m
		Definición y redefinición de los requisitos del sistema		2h 22m
IT3	A&D	Analizar el comportamiento y definir una Arquitectura software candidata	AS	4h 43m
		Diseñar la base de datos y los componentes del sistema		4h 43m
IT6	Imp	Implementar los componentes	DS	7h 57m
			AS	2h 41m
		Testear esos componentes (comprobar su funcionamiento)	DS	7h 57m
			AS	2h 41m
IT9	INT	Integrar cada subsistema al completo	DS	4m
			DJ	2h 46m
		Integrar todos los subsistemas al sistema	DS	4m
			DJ	2h 17m

IT9	TEST	<b>Testear y evaluar el resultado</b>	DS	4m
		<b>Asegurarse de que el sistema proporcionado cumple con los objetivos establecidos y comprobar que los test son correctos</b>	DJ	2h 16m
IT9	D	<b>Producir la Unidad de despliegue</b>	DS	4m
		<b>Gestionar las pruebas de instalación del sistema en el sitio deseado</b>	DJ	2h 16m
		<b>Desarrollar material de soporte</b>		47m

El resto lo hago en la wiki que me da pereza trabajar dos veces. :)

#### Contribución de los líderes de equipo

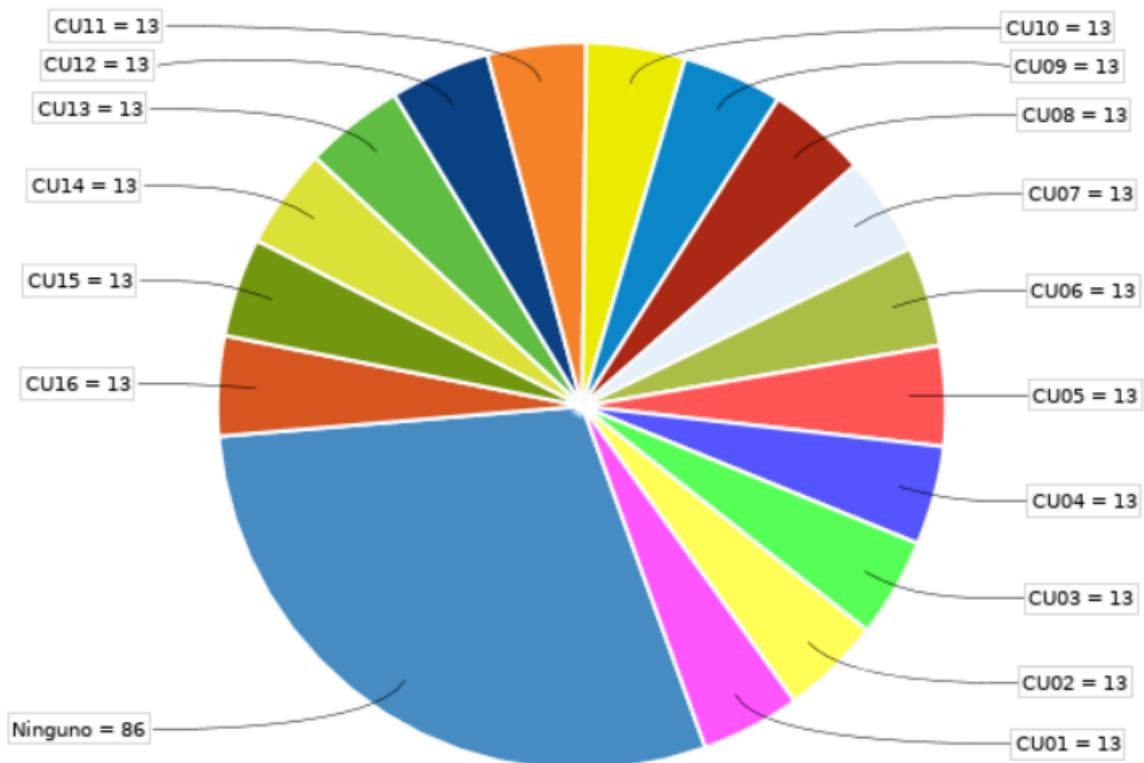
Por último, sobre la planificación anterior establecida en Jira se crean dos gráficos: en el primero, se puede observar la contribución de cada usuario a las tareas asignadas a cada caso de uso.

Grafico de Tarta

#### Proporción de tareas por caso de uso

En el siguiente informe de tipo tarta se puede observar la distribución por casos de uso de las tareas. Cada caso de uso cuenta con el mismo número de tareas pues todos tienen las mismas fases.

Se puede apreciar un gran número de tareas no asignadas a ningún caso de uso. Corresponden a las subtareas, las cuales heredan el caso de uso al que pertenecen de la tarea padre



El siguiente informe de carga de trabajo del usuario muestra el número de incidencias asignadas al **Señor Azul**, además del tiempo estimado que se espera que tarden dichas tareas en realizarse.

Informe de carga de trabajo para el usuario GE01Marron (elige otro)

Proyectos	Incidencias asignadas	Carga de trabajo
ProyectoGE01	63	2 semanas, 4 días, 1 hora, 32 minutos
Total	63	2 semanas, 4 días, 1 hora, 32 minutos

Hay que concretar que dentro de este informe no se tienen en cuenta las subtareas, ni el tiempo que estas ocupan en cada usuario, por lo que puede variar entre distintos usuarios.

## Desarrollo del progreso del proyecto

### Síntesis de la planificación por roles

En este apartado se documentará la distribución de los casos de uso, realizados teniendo en cuenta **el punto de vista de los roles**, que participan en cada uno de los equipos, a modo de síntesis del apartado de configuración del seguimiento de planificación.

En concreto, estos roles siguen la siguiente planificación:

Equipo	JP	AS	DS	DJ	Casos de Uso:
1	Sr Blanco	Sr Marrón	Sr. Naranja	Sr Azul	CU01, CU02, CU03, CU04
2	Sr Marrón	Sr. Naranja	Sr Azul	Sr Blanco	CU05, CU06, CU07, CU08
3	Sr. Naranja	Sr Azul	Sr Blanco	Sr Marrón	CU09, CU10, CU11, CU12
4	Sr Azul	Sr Blanco	Sr Marrón	Sr. Naranja	CU13, CU14, CU15, CU16

Siendo cada uno de los roles el líder de uno de los 4 equipos.

1 - Implementar clase Evento + AppDatabase + EventoDAO + API (solo con municipios si se puede) + EventoActivity con CrearEventoMunicipioFragment

2- Implementar API (con montañas si se puede) + CrearEventoMontanaFragment

3 - Añadir campos de usuario necesarios en AppDatabase, así como crear la clase Usuario junto con su UsuarioDAO y el fragmento PerfilFragment. Finalmente, se crea la actividad Registrarse

.....  
4 - Implementar barra de búsqueda?????????????????

.....  
6 - Crear detalles de ubicación?????????????????

7 - Añadir el fragmento ModificarEventoFragment dentro de la actividad DetallesEventoActivity, así como el método en EventoDAO

8 - Añadir el panel DeleteDialogEvent, así como el método de borrar deleteUser en la interfaz UsuarioDAO.

.....  
9 - Consultar el tiempo en la ubicación actual?????????????????

10 - Modificar el idioma y tema a modo oscuro

11 - Añadir los fragmentos ConsultarEventosFragment y ListaEventosFragment

.....  
12 - Añadir la actividad DetallesEventoActivity, junto con su fragmento DetallesEvento y su layout

13 - Añadir la actividad IniciarSesionActivity y su layout.

5 - Añadir el fragmento AjustesFragment y su layout, así como los cambios necesarios para aplicar el modo oscuro y el método onResume() en las activities.

.....

14 - Cerrar sesión

15 - Añadir el PerfilFragment

16 - Añadir el panel DeleteDiaogFragment

Así mismo, la planificación de los roles respecto de las disciplinas de **Implementación e Integración y Testeo** es:

Encargado	CU01	CU02	CU03	CU04	CU05	CU06	CU07	CU08	CU09	CU10	CU11	CU12	CU13	CU14	CU15	CU16
Sr Blanco					Imp	Imp					Imp	Imp	Imp		Imp	Imp
						I&T			I&T	I&T			I&T	I&T	I&T	
Sr Marrón			Imp						Imp				Imp	Imp	Imp	
										I&T	I&T	I&T		I&T		
Sr Naranja	Imp															
	I&T	I&T	I&T										I&T			
Sr Azul	Imp			Imp				Imp	Imp	Imp	Imp					
				I&T	I&T	I&T	I&T	I&T								

Encargado	CU01	CU02	CU03	CU04	CU05	CU06	CU07	CU08	CU09	CU10	CU11	CU12	CU13	CU14	CU15	CU16
Sr Blanco					Imp	Imp					Imp	Imp	Imp		Imp	Imp
					Test C.				Test C.			Test C.				Test C.
Sr Marrón			Imp						Imp				Imp	Imp	Imp	
													Test C.	Test C.	Test C.	
Sr Naranja	Imp															
	Test C.	Test C.	Test C.			Test C.	Test C.	Test C.								
Sr Azul	Imp			Imp				Imp	Imp	Imp	Imp					
				Test C.						Test C.	Test C.					

Para comprobar que este reparto de tareas es equitativo entre los distintos roles, a continuación se muestra el **número de tareas asignadas a cada rol**, en la que cada disciplina compartida por 2 roles es ponderada con 0,5 en cada rol:

**Distribución de implementaciones:**

Encargado	Casos de uso	Núm
Sr Blanco	CU05, CU06, CU11, CU12, CU13, CU15, CU16	4,5
Sr Marron	CU03, CU09, CU13, CU14, CU15	3
Sr Naranja	CU01, CU02, CU03, CU04, CU05, CU06, CU07, CU08	5
Sr Azul	CU01, CU04, CU08, CU09, CU10, CU11	3,5

**Distribución de integraciones:**

Encargado	Casos de uso	Núm
Sr Blanco	CU06, CU09, CU10, CU14, CU15, CU16	4,5
Sr Marron	CU10, CU11, CU12, CU14	3
Sr Naranja	CU01, CU02, CU03, CU13	4
Sr Azul	CU04, CU05, CU06, CU07, CU08	4,5

## Desarrollo del Equipo 1 (Sr Blanco)

En este apartado se documentará todo el proceso seguido por el Equipo 1 para realizar los casos de uso, desde la implementación hasta la integración final. Se detalla **qué archivos se han modificado** concretamente y **cómo se ha realizado la integración** de estos casos de uso.

El equipo 1 será el encargado de llevar a cabo los casos de uso CU01, CU02, CU03 y CU04. A continuación se explicarán las tareas de implementación e integración que se han realizado en cada uno de ellos.

### CU01 - Añadir Evento de Municipio

Este caso de uso estructural permite al usuario añadir un nuevo evento de municipio en una fecha determinada, estando asignado a una condición meteorológica en tiempo real de la ubicación del municipio.

## Implementación de CU01

Para realizar este caso de uso ha sido necesaria la creación de la clase Evento perteneciente a la lógica de negocio. Esta clase evento, además requirió emplear el servicio de Room para almacenar los eventos en la base de datos, por lo que también ha sido necesaria la realización de una interfaz EventoDao. Así mismo, en este caso de uso también se ha creado la clase **AppDatabase** para conectarse a la base de datos.

Así mismo, se han creado las clases necesarias para obtener todos los municipios existentes de España, que se encuentran en un documento **municipios.json**. Dichas clases son JsonSingleton, que permite obtener los datos del documento, y Municipio, que representa cada municipio obtenido.

Esta disciplina ha sido realizada en conjunto por el **Sr. Naranja** y el **Sr. Azul**.

Por tanto, en la realización de este caso de uso se han creado las siguientes clases:

- La Activity correspondientes a la pantalla de crear un evento de Municipio (CrearEventoActivity) y sus fragmentos (CrearEvento y CrearEventoMunicipio), junto con los layout correspondiente a dichos componentes software
- La clase Evento
- Una Interfaz EventoDao
- La clase AppDatabase
- Las clases correspondientes para obtener los municipios de la API (JsonSingleton y Municipio)

## Integración de CU01

La integración de este caso de uso únicamente ha necesitado la **publicación del código terminado** lanzando una primitiva **Push** en la rama del repositorio correspondiente, pues no necesita la implementación de ningún otro caso de uso.

Esta disciplina ha sido realizada por el **Sr. Naranja**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU02 - Añadir Evento de Ruta Montaña

Este caso de uso estructural permite al usuario añadir un nuevo evento de montaña en una fecha determinada, estando asignado a una condición meteorológica en tiempo real de la ubicación de la montaña..

## Implementación de CU02

Para realizar este caso de uso ha sido necesaria la implementación del CU01 - Crear Evento de Municipio, que contiene las clases Evento y EventoDAO, utilizando también la clase AppDatabase.

Esto ha implicado la creación de un nuevo fragmento CrearEventoMontana en la actividad de creación de un evento (CrearEventoActivity), que permite crear un evento de montaña concreto a partir de las montañas existentes.

Esta disciplina ha sido realizada por el **Sr. Naranja**.

Por tanto, la implementación de este caso de uso ha supuesto la creación de la siguiente clase:

- El fragmento CrearEventoMontana y su layout

## Integración de CU02

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU01, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**. Esta tarea la ha realizado **el Sr. Naranja**.

Una vez finalizada esta fase, **se integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU03 - Añadir Usuario

Este caso de uso estructural permite al usuario registrarse con unas credenciales.

## Implementación de CU03

Para realizar este caso de uso ha sido necesaria la implementación del CU02 - Añadir Ruta de Montaña, que crea un evento de Montaña en la base de datos.

Este caso de uso contiene las clases Usuario y UsuarioDAO, utilizando también la clase AppDatabase.

Esta disciplina ha sido realizada en conjunto por el **Sr. Naranja** y el **Sr. Marrón**.

Por tanto, la implementación de este caso de uso ha supuesto la modificación de la siguiente clase:

- La Activity Main

Así mismo, ha supuesto la creación de las siguientes clases:

- Las Activity correspondiente a la pantalla de registro (Registrarse) y su layout
- La clase del modelo de datos Usuario
- Una Interfaz UsuarioDao

## Integración de CU03

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU02 - Añadir Evento de Ruta Montaña.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**. Esta tarea la ha realizado **el Sr. Naranja**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU04 - Añadir una barra de búsqueda y filtrado de ubicaciones

Este caso de uso estructural consiste en la incorporación de una barra de búsqueda para buscar una ubicación con su tiempo meteorológico asignado.

### Implementación de CU04

Para realizar este caso de uso ha sido necesaria la implementación del CU03 - Añadir Usuario, que contiene las clases Usuario y UsuarioDAO, utilizando también la clase AppDatabase.

Esta disciplina ha sido realizada en conjunto por el **Sr. Naranja** y el **Sr. Azul**.

La implementación de este caso de uso se ha completado, mediante la modificación de las siguientes clases:

- MainActivity. Incluye la incorporación del icono que redirige al listado de ubicaciones
- AndroidManifest
- Layout del menú main.xml

Además, se han añadido las siguientes clases:

- La actividad LocalizacionesActivity
- Clase APIManager y APIManagerDelegate

## Integración de CU04

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU03, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**. Esta tarea la ha realizado **el Sr Azul**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## Desarrollo del Equipo 2 (Sr Marrón)

En este apartado se documentará todo el proceso seguido por el Equipo 2 para realizar los casos de uso, desde la implementación hasta la integración final. Se detalla **qué archivos se han modificado** concretamente y **cómo se ha realizado la integración** de estos casos de uso.

El equipo 2 será el encargado de llevar a cabo los casos de uso CU05, CU06, CU07 y CU08. A continuación se explicarán las tareas de implementación e integración que se han realizado en cada uno de ellos.

### CU05 - Añadir preferencias desde el menú de AppBar (ajustes ...)

Este caso de uso estructural consiste en incorporar un apartado de configuración que permite al usuario personalizar su experiencia con la aplicación, así como el modo oscuro, preferencias, etc.

#### Implementación de CU05

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU13 - Iniciar sesión, que contiene la clase IniciarSesion (Activity).

Esta disciplina ha sido realizada en conjunto por el **Sr. Naranja** y el **Sr. Blanco**.

La implementación de este caso de uso se ha completado mediante la modificación de las siguientes clases:

- Se ha modificado todos los archivos relativos al panel lateral de navegación, drawer, etc.

Además, se han añadido las siguientes clases:

- AjustesFragment y su correspondiente layout

#### Integración de CU05

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU13, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**. Esta tarea la ha realizado **el Sr Azul**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU06 - Consultar el tiempo detallado de una ubicación

Este caso de uso no estructural permite al usuario visualizar el tiempo meteorológico de una ubicación específica en tiempo real.

### Implementación de CU06

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU04 - Añadir una barra de búsqueda y filtrado de ubicaciones, el cual incluye las clases java necesarias para realizar peticiones a la API. Se ha necesitado crear una nueva actividad que gestione el detalle del tiempo de una ubicación, de la lista de ubicaciones (municipios).

Esta disciplina ha sido realizada en conjunto por el **Sr. Naranja** y el **Sr. Blanco**.

La implementación de este caso de uso se ha completado mediante la modificación de las siguientes clases:

- Clase ListadoLocalizacionesActivity

Además, se han incluido la siguiente clase:

- DetalleLocalizaciónActivity y su layout

### Integración de CU06

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU04, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina ha sido realizada en conjunto por el **Sr. Azul** y el **Sr. Blanco**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU07 - Modificar un evento

Este caso de uso no estructural permite al usuario modificar los datos de un evento, así como su nombre, descripción, fecha y ubicación.

## Implementación de CU07

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU06 - Consultar el tiempo detallado de una ubicación, el cual incluye los componentes software de Android necesarios para elegir y ver las condiciones meteorológicas de una ubicación específica.

Esta disciplina ha sido realizada por el **Sr. Naranja**.

La implementación de este caso de uso se ha completado mediante la modificación de las siguientes clases:

- La clase DAO de los eventos llamada EventoDAO

Además, se han creado las siguientes clases:

- Los fragmentos ModificarEventoMontanaFragment y ModificarEventoMunicipioFragment y sus respectivos layouts.

## Integración de CU07

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU06, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina ha sido realizada por el **Sr. Azul**.

Una vez finalizada esta fase, **se integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU08 - Eliminar un evento

Este caso de uso no estructural permite al usuario borrar un evento creado y configurado previamente en la lista de eventos.

## Implementación de CU08

Para realizar este caso de uso ha sido necesaria la implementación del CU01 - Crear Evento de Municipio y el caso CU02 - Crear Evento de Montaña, que contiene las clases Evento y EventoDAO, utilizando también la clase AppDatabase.

De esta forma se ha trabajado sobre el componente de detalles de un evento sobre el cual se ha añadido la funcionalidad de borrar el mismo.

Por tanto, en la realización de este caso de uso se han creado la siguiente clase:

- La clase DeleteEventDialog

Esta disciplina ha sido realizada en conjunto por el **Sr. Naranja y el Sr. Azul**.

### Integración de CU08

La integración de este caso de uso ha necesitado la integración previamente de los casos de uso CU06, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina ha sido realizada por el **Sr. Azul**.

Una vez finalizada esta fase, **se integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

### Desarrollo del Equipo 3 (Sr Naranja)

En este apartado se documentará todo el proceso seguido por el Equipo 3 para realizar los casos de uso, desde la implementación hasta la integración final. Se detalla **qué archivos se han modificado** concretamente y **cómo se ha realizado la integración** de estos casos de uso.

El equipo 3 será el encargado de llevar a cabo los casos de uso CU09, CU10, CU11 y CU12. A continuación se explicarán las tareas de implementación e integración que se han realizado en cada uno de ellos.

### CU09 - Consultar el tiempo meteorológico en la ubicación actual

Este caso de uso no estructural permite al usuario consultar los detalles del tiempo asociado a la localización/municipio más cercano.

### Implementación de CU09

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU08 - Eliminar un evento.

La implementación de este caso de uso se ha llevado a cabo mediante la modificación de los siguientes archivos:

- El fragmento InicioFragment

A su vez se han creado los siguientes nuevos archivos:

- Clase Weather
- Clase APIManager
- Clase APIManagerDelegate

Esta disciplina ha sido realizada en conjunto por el **Sr. Blanco y el Sr. Azul**.

### **Integración de CU09**

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU08, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina ha sido realizada por el **Sr. Blanco**.

Una vez finalizada esta fase, **se integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

### **CU10 - Modificar idioma y tema a modo oscuro**

Este caso de uso no estructural permite al usuario realizar una serie de modificaciones en la aplicación para configurar esta a su gusto como cambiar el idioma, el tema a modo oscuro, etc.

### **Implementación de CU10**

Para la implementación de este caso de uso se han modificado la mayoría de actividades y fragmentos del proyecto.

Cabe mencionar la modificación del fragmento AjustesFragment en el cual se ha añadido la opción de activar el modo oscuro en la aplicación.

Finalmente todo el código referido a las características de ambos modos (oscuro y claro) se encuentra en los dos archivos themes del layout.

Esta disciplina ha sido realizada por el **Sr. Azul**.

### **Integración de CU10**

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU09, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina ha sido realizada en conjunto por el **Sr. Blanco y el Sr. Marrón**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU11 - Consultar lista de eventos

Este caso de uso no estructural permite al usuario visualizar el listado completo de eventos de municipios y montañas creados por su parte.

### Implementación de CU11

Para realizar este caso de uso ha sido necesaria la implementación del CU01 - Crear Evento de Municipio y el caso CU02 - Crear Evento de Montaña, que contiene las clases Evento y EventoDAO, utilizando también la clase AppDatabase.

La implementación de este caso de uso se ha completado mediante la creación de las siguientes clases:

- Clase java PlaceholderItem
- Fragmento ConsultaEventosFragment
- Fragmento ListaEventosFragment

Esta disciplina ha sido realizada en conjunto por el **Sr. Blanco y el Sr. Azul**.

### Integración de CU11

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU10, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina ha sido realizada por el **Sr. Marrón**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU12 - Consultar un evento

Este caso de uso no estructural permite al usuario visualizar en detalle los datos pertinentes de un evento de municipio o montaña.

## Implementación de CU12

Para realizar este caso de uso ha sido necesaria la implementación de los casos de uso CU01, CU02 y CU09 correspondientes con la creación de eventos y obtención de los datos metereológicos de una ubicación.

La implementación de este caso de uso se ha completado mediante la creación de las siguientes clases:

- Interfaz fragment\_detalles\_evento
- Interfaz activity\_detalles\_localizaciones
- Fragmento DetallesEventoFragment
- Actividad DetallesEventoActivity

Esta disciplina ha sido realizada por el **Sr. Blanco**.

## Integración de CU12

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU11, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina ha sido realizada por el **Sr. Marrón**.

Una vez finalizada esta fase, **se integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## Desarrollo del Equipo 4 (Sr Azul)

En este apartado se documentará todo el proceso seguido por el Equipo 4 para realizar los casos de uso, desde la implementación hasta la integración final. Se detalla **qué archivos se han modificado** concretamente y **cómo se ha realizado la integración** de estos casos de uso.

El equipo 4 será el encargado de llevar a cabo los casos de uso CU13, CU14, CU15 y CU16. A continuación se explicarán las tareas de implementación e integración que se han realizado en cada uno de ellos.

## CU13 - Iniciar sesión

Este caso de uso no estructural permite al usuario iniciar sesión con las credenciales asociadas a su cuenta de usuario.

## Implementación de CU13

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU12 - Consultar un evento, el cual incluye las actividades y fragmentos DetallesEvento.

Esta disciplina ha sido realizada por el **Sr. Naranja**.

La implementación de este caso de uso se ha completado mediante la modificación de la siguiente clase:

- La clase MainActivity

Además, se han creado la siguiente clase:

- La actividad InicioSesion junto con su layout

### Integración de CU13

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU12, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Según la planificación de JIRA, esta disciplina la debe realizar tanto el Sr. Marrón como el Sr. Naranja. Sin embargo, en la realidad sólo la ha realizado el **Sr. Marrón**, y la tarea de integración del Sr. Naranja se ha puesto como estado a “Hecho”.

Una vez finalizada esta fase, **se integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

### CU14 - Cerrar sesión

Este caso de uso no estructural permite al usuario cerrar sesión.

### Implementación de CU14

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU05 - Añadir preferencias desde el menú de AppBar (ajustes ...), el cual incluye el código necesario para la implementación de un apartado de Ajustes mediante la clase AjustesFragment.

Esta disciplina ha sido realizada por el **Sr. Marrón**.

La implementación de este caso de uso se ha completado mediante la modificación de la siguiente clase:

- La clase MainActivity
- El layout main.xml correspondiente al menú

## Integración de CU14

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU05, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Según la planificación de JIRA, esta disciplina la debe realizar tanto el Sr. Marrón como el Sr. Blanco. Sin embargo, en la realidad sólo la ha realizado el **Sr. Marrón**, y la tarea de integración del Sr. Blanco se ha puesto como estado a “Hecho”.

Una vez finalizada esta fase, **se integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU15 - Modificar usuario

Este caso de uso no estructural permite al usuario modificar su perfil de usuario, así como su nombre de usuario y contraseña.

### Implementación de CU15

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU14 - Cerrar sesión, el cual incluye el código necesario para cerrar la sesión de usuario.

Esta disciplina ha sido realizada por el **Sr. Marrón** y el **Sr. Blanco**.

Se ha modificado la clase EventoDAO, implementando la operación de la base de datos de eliminación de cuenta de usuario.

Además, se han creado la siguiente clase:

- El fragmento PerfilFragment y su layout

## Integración de CU15

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU14, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina la debe realizar el **Sr. Blanco**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

## CU16 - Eliminar usuario

Este caso de uso no estructural permite al usuario modificar su perfil de usuario, así como su nombre de usuario y contraseña.

### Implementación de CU16

Para realizar este caso de uso ha sido necesaria la implementación del caso de uso CU15 - Modificar Usuario, el cual incluye la clase PerfilFragment.

Esta disciplina ha sido realizada por el **Sr. Blanco**.

Se ha modificado la clase EventoDAO, implementando la operación de la base de datos de eliminación de cuenta de usuario.

Además, se han creado la siguiente clase:

- El evento de diálogo DeleteEventDialog

### Integración de CU16

La integración de este caso de uso ha necesitado la integración previamente del caso de uso CU15, por lo que se ha realizado una primitiva **Merge** para trabajar sobre el código con este caso de uso realizado.

Posteriormente, una vez que el caso de uso está implementado, **se ha realizado la publicación del código** terminado en la rama del repositorio correspondiente lanzando una primitiva **Push**.

Esta disciplina la debe realizar el **Sr. Blanco**.

Una vez finalizada esta fase, se **integrará este caso de uso en la rama Develop** con todos los cambios de la rama correspondiente mediante un Merge, publicando después los cambios correspondientes en la rama Develop mediante la primitiva Push.

# Diseño de la interfaz de usuario

## Mapa Navegación: pantallas, patrones y diagrama

### Diagrama de casos de uso

A continuación se muestra el diagrama de casos de uso de la aplicación, todos referentes a las acciones que puede realizar el usuario y el sistema.

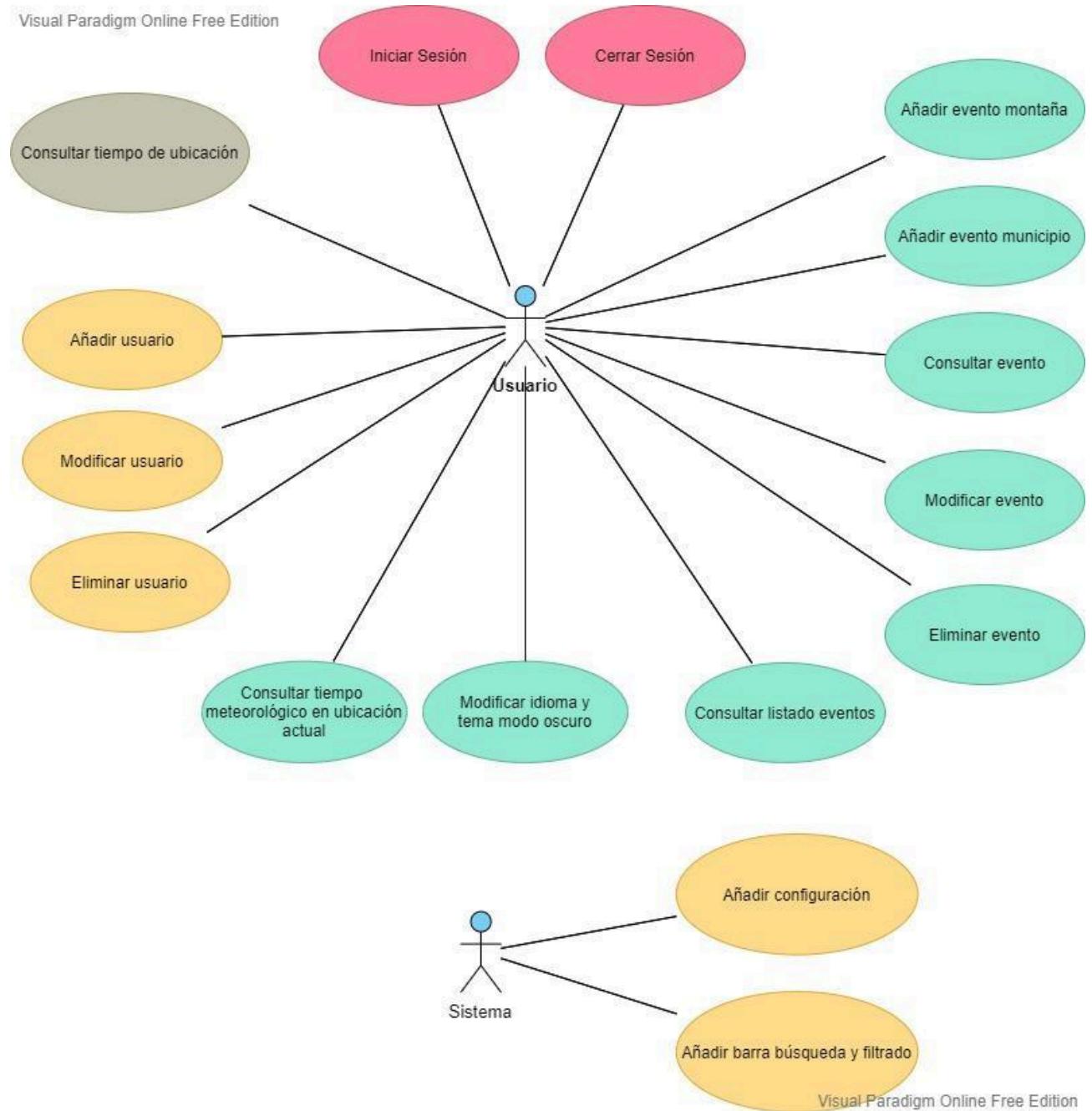


Imagen 3. Diagrama de casos de uso en Visual Paradigm

## Diagrama de navegación

A continuación se muestra un posible diagrama de navegación que podría seguir la aplicación:

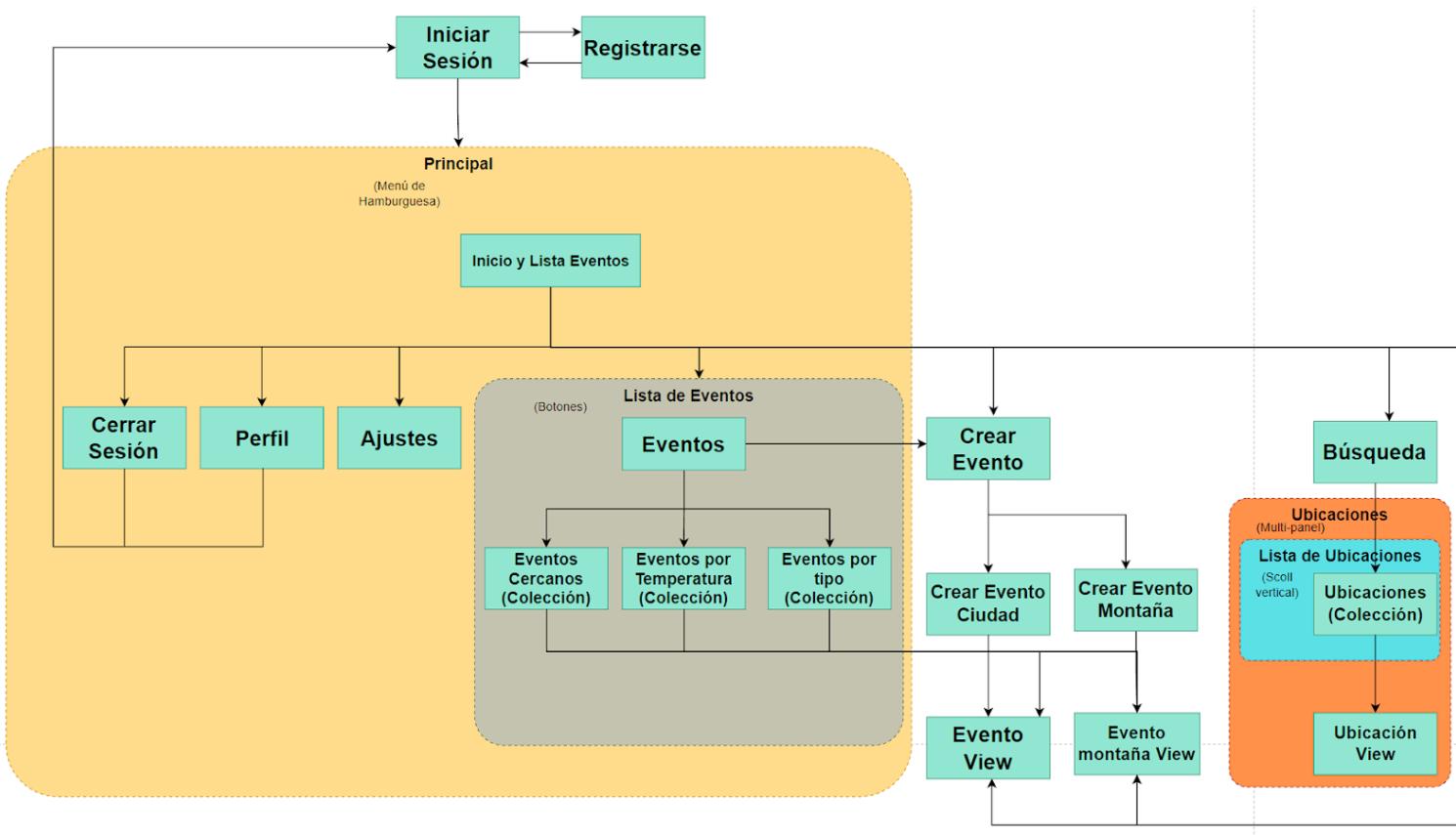
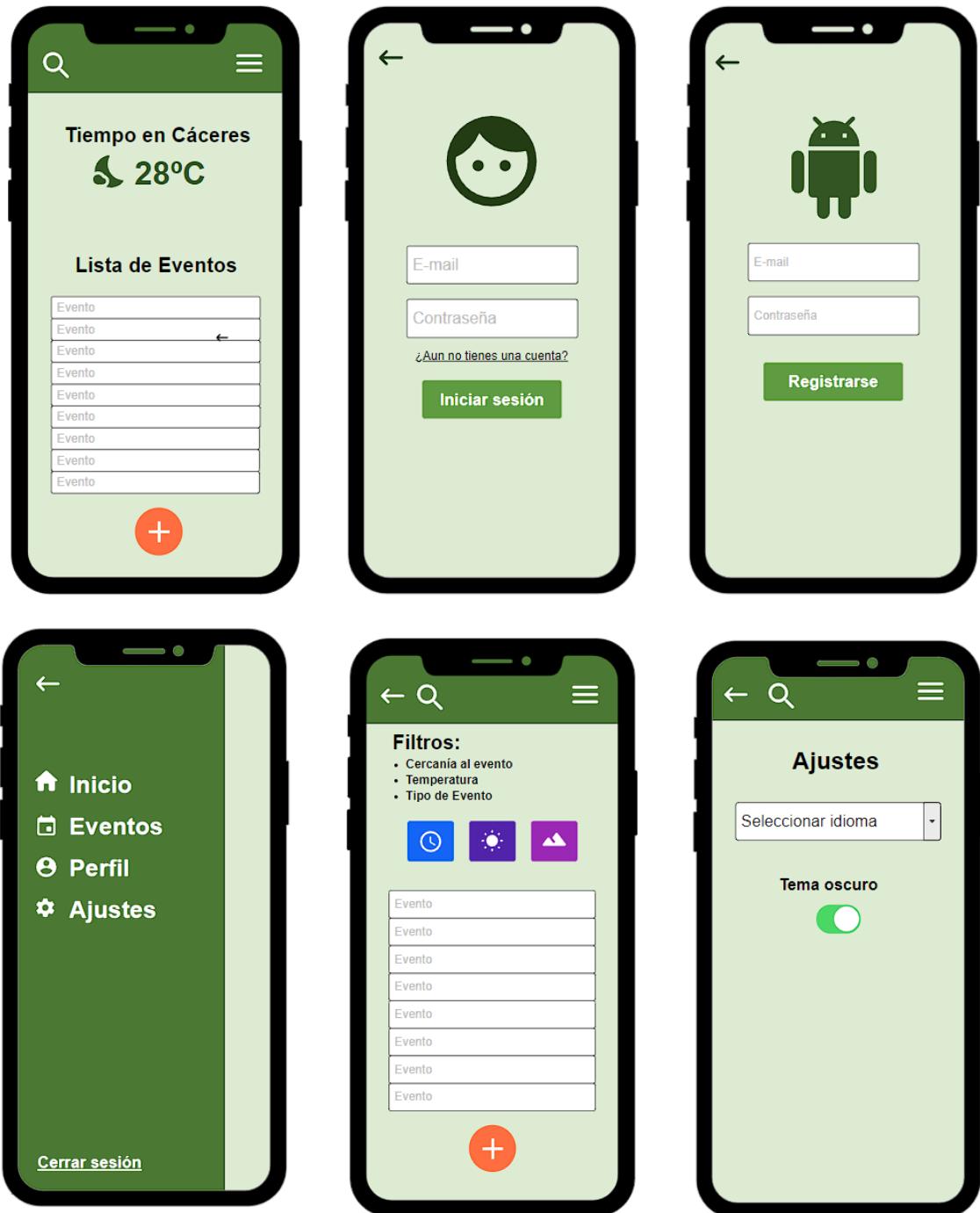


Imagen 4. Diagrama de navegación de las pantallas

## Mockup

A continuación se muestra un Mockup que contiene un conjunto de posibles pantallas del proyecto, a modo de prototipo que muestre las funcionalidades de la aplicación:







Como se puede observar, para poder utilizar la aplicación en primer lugar es necesario iniciar sesión contando con pantallas para iniciar sesión con una cuenta y otra para registrarse, pudiendo crear una cuenta nueva en caso de que esta no exista.

Una vez se ha iniciado sesión, la aplicación consta principalmente de 4 pantallas, siendo estas las siguientes:

- Una **pantalla principal**, que muestra el tiempo que hace (temperatura, clima...) en la ubicación actual del usuario, así como una lista con los eventos que ha creado, a través de la cual se pueden consultar los detalles (tiempo, entre otros) de este.
- Una pantalla destinada al **filtrado de eventos**, pudiendo buscar entre los distintos eventos creados por el usuario a través de una serie de filtros, como el tiempo actual que hace, lugar en el que se desarrolla el evento, etc.
- Una pantalla con el **perfil del usuario**, que muestra todos sus datos, pudiendo consultar o modificar estos, así como incluso borrar el usuario.
- Una pantalla de **ajustes**, que contiene los principales ajustes que se pueden aplicar a la app, como el tema claro/oscuro, notificaciones, etc.

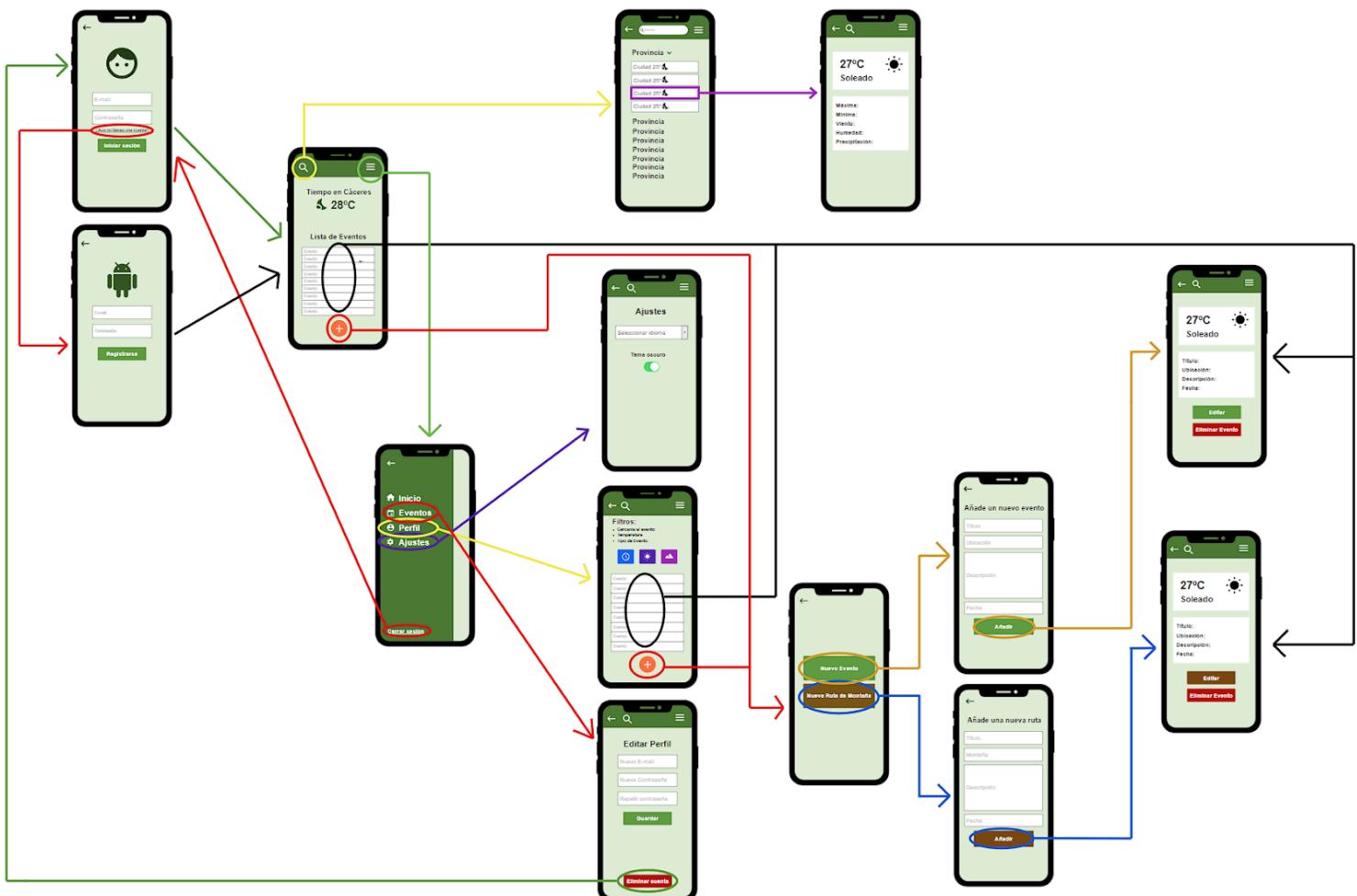
Cabe destacar que tanto en la lista de todos los eventos ubicada en la pantalla principal como en las listas proporcionadas por la pantalla de filtrado de eventos, se pueden acceder a los datos de un evento tocando sobre estos, que mostrará una pantalla desde la que se pueden

consultar sus datos (lugar, temperatura, clima, fecha, etc..) o incluso modificar algunos de ellos, pudiendo también borrar dicho evento de la app.

Así mismo, posee un panel lateral desplegable en la pantalla principal desde el que se pueden acceder a las principales funcionales de la aplicación, siendo estas:

- **Inicio:** Un acceso a la propia pantalla principal.
- **Eventos:** Un acceso a la lista de eventos que ha creado el usuario para poder filtrarlos.
- **Perfil:** Un acceso al perfil del usuario.
- **Ajustes:** Un acceso a la pantalla con los principales ajustes de la aplicación.

## Grafo de navegación



## Patrones de navegación aplicados

En este apartado se muestran todos los patrones de navegación que se han utilizado para la realización de la aplicación, siendo estos los siguientes:

- Patrón de lista y detalle.
- Patrón de cajón de navegación.
- Patrón de botones y objetivos sencillos.

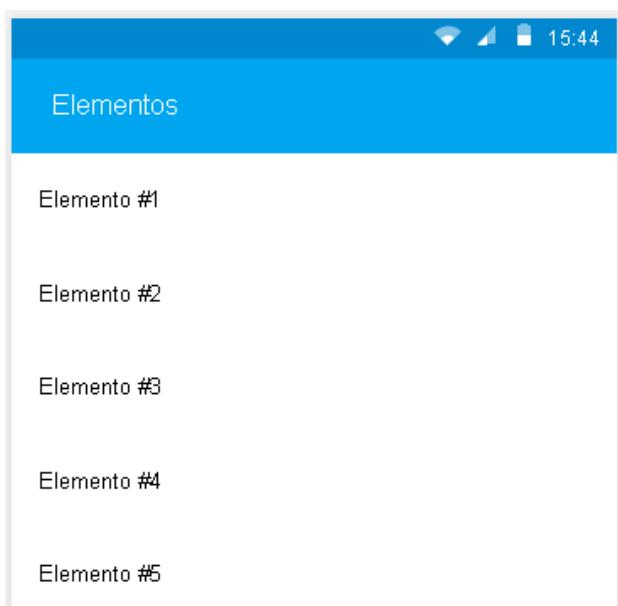
A continuación se detallarán cada uno de los patrones implementados.

### Patrón de lista y detalle

Para desplazarse por la aplicación se ha utilizado **un patrón de lista y detalle** referente a la pantalla de inicio, que mostrará los elementos de más interés para el usuario, en este caso una lista con todos los eventos que ha creado.

De la misma forma, en la pantalla de lista de eventos también se ha aplicado este patrón para buscar una lista de eventos concretos a través de la pantalla de filtrado de eventos.

Así mismo, a través de estas listas, se puede acceder a una pantalla que permite consultar los detalles de un evento concreto tocando sobre dicho evento de la lista, que llevará a una pantalla que muestra sus datos.

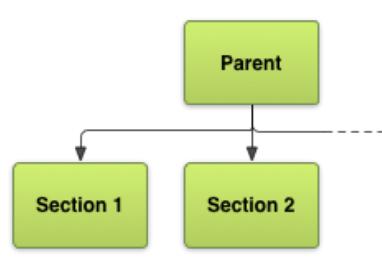


### Patrón de cajón de navegación

En lo referente a cada una de las funcionalidades, se utilizará **un patrón de cajón de navegación**, creando un menú lateral desplegable desde el que se puede navegar entre funcionalidad y funcionalidad, pudiendo cambiar rápidamente de pantalla a través de dicho menú lateral.

### Patrón de botones y objetivos sencillos

Finalmente, se ha aplicado el patrón de botones y objetivos sencillos mostrando mediante botones con iconos intuitivos en ciertas funcionalidades, de manera que permitan a los usuarios acceder más fácilmente a dichas funcionalidades.



EN ROJO, APARTADOS DE GPS. EN AZUL, ASEE.

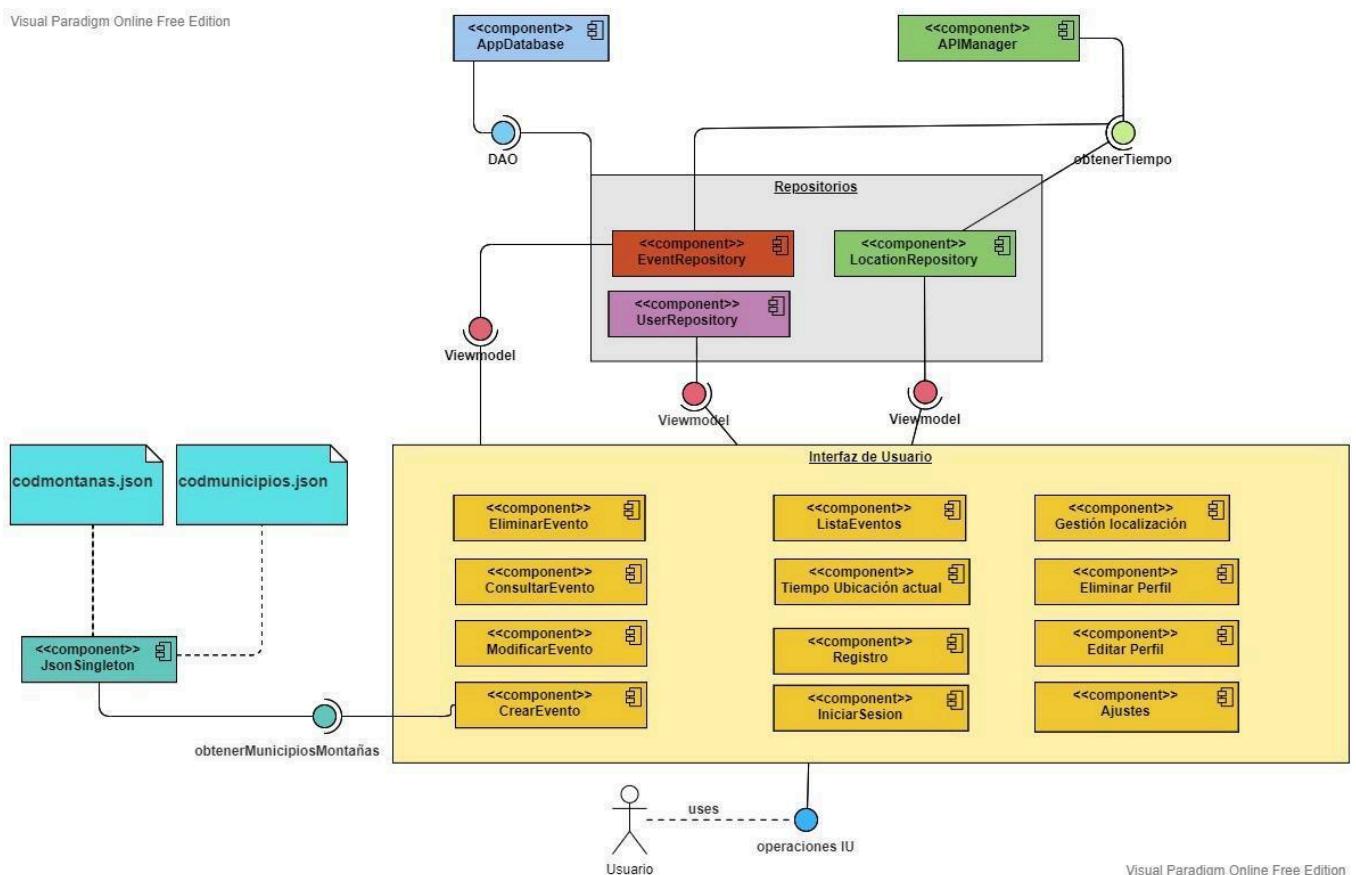
## Diseño arquitectónico

En esta sección se describen los aspectos esenciales del diseño arquitectónico de la aplicación ClimApp, partiendo desde las decisiones tomadas en el diseño a los patrones arquitectónicos y de diseño, comentando el diagrama de componentes de la aplicación.

## Decisiones tomadas

## Diagrama de componentes

A continuación se muestra un diagrama con todos los principales componentes que conforman el sistema:



Estos componentes pueden clasificarse en 3 tipos:

- **Componentes de interfaz de usuario:** Son aquellos que interactúan directamente con el usuario, ofreciéndoles las diversas funcionalidades (crear un evento, consultar la lista de eventos, modificar estos, etc).
- **Repositorios:** Consisten en los componentes que hacen la función de repositorios, utilizados en el patrón Repository para obtener los datos de fuentes externas y son utilizados en otros componentes a través de los viewmodel. Se trata de los componentes EventRepository, UserRepository y LocationRepository.
- **Componentes que enlazan fuentes externas:** Se trata de los componentes que permiten acceder a datos de fuentes externas, como la base de datos de Room (AppDatabase), los ficheros Json con municipios y montañas (JsonSingleton) o la api que obtiene la información del clima (ApiManager).

Así mismo, entre los componentes presentados en el diagrama anterior existen las siguientes interacciones:

- El **usuario interactúa con la aplicación** utilizando la interfaz de usuario (UI) definida por los componentes Android y sus layouts, agrupados en el paquete “Interfaz de usuario”. Esto se representa con la relación entre el actor usuario y la interfaz “operaciones IU”.
- El componente **JsonSingleton** ofrece la interfaz “obtenerMunicipiosMontañas”, requerida por el componente CrearEvento.
- Los **componentes de la interfaz del usuario acceden**, a través de los distintos viewmodels, a **los diferentes repositorios** que existen dentro de AppContainer (UserRepository, LocationRepository y EventRepository) para almacenar y obtener datos de fuentes externas. Por ello, cada componente del patrón repository ofrece interfaces “viewmodel” que son utilizadas por los componentes de la interfaz de usuario.
- La agrupación de componentes de tipo **Repository** (UserRepository, LocationRepository y EventRepository) hacen uso de las operaciones de la Room, a través de las interfaces DAO ofrecidas por el componente AppDataBase, para realizar operaciones en la base de datos.
- El componente **APIManager** proporciona las peticiones pertinentes a la API OpenWeather mediante la interfaz “obtenerTiempo”, requerida por los repositorios EventRepository, donde cada evento tiene asociado el tiempo meteorológico según la ubicación; LocationRepository, donde cada localización (municipio) tiene asignado unas condiciones meteorológicas.

## Patrones arquitectónicos

Son aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software.

## Gestión del entorno

En este apartado se abordarán todos los detalles sobre el entorno en el que se ha llevado a cabo el proyecto, comprendiendo desde las características que poseen hasta los pasos que se han seguido para configurar dichos entornos.

## Gestión de la configuración

### Entorno utilizado

En concreto, para la realización de este proceso se han utilizado 2 herramientas, una para la **codificar la aplicación** (y posteriormente ejecutar el código implementado) y otra para **almacenar el código creado por los integrantes del equipo**, pudiendo compartirse este y permitiendo integrarlo todo en un repositorio Git. A continuación se explicarán cada una de estas herramientas.

#### Herramienta de implementación: Android Studio

Android Studio es una herramienta de codificación y ejecución de programas y aplicaciones para dispositivos Android basada en IntelliJ IDEA.

Este consiste en un software que permite programar en lenguajes como Java o Kotlin aplicaciones para posteriormente poder ser ejecutada en dispositivos con versiones de Android superiores a 3.0.

Además de ofrecer las mismas herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones, entre otras:

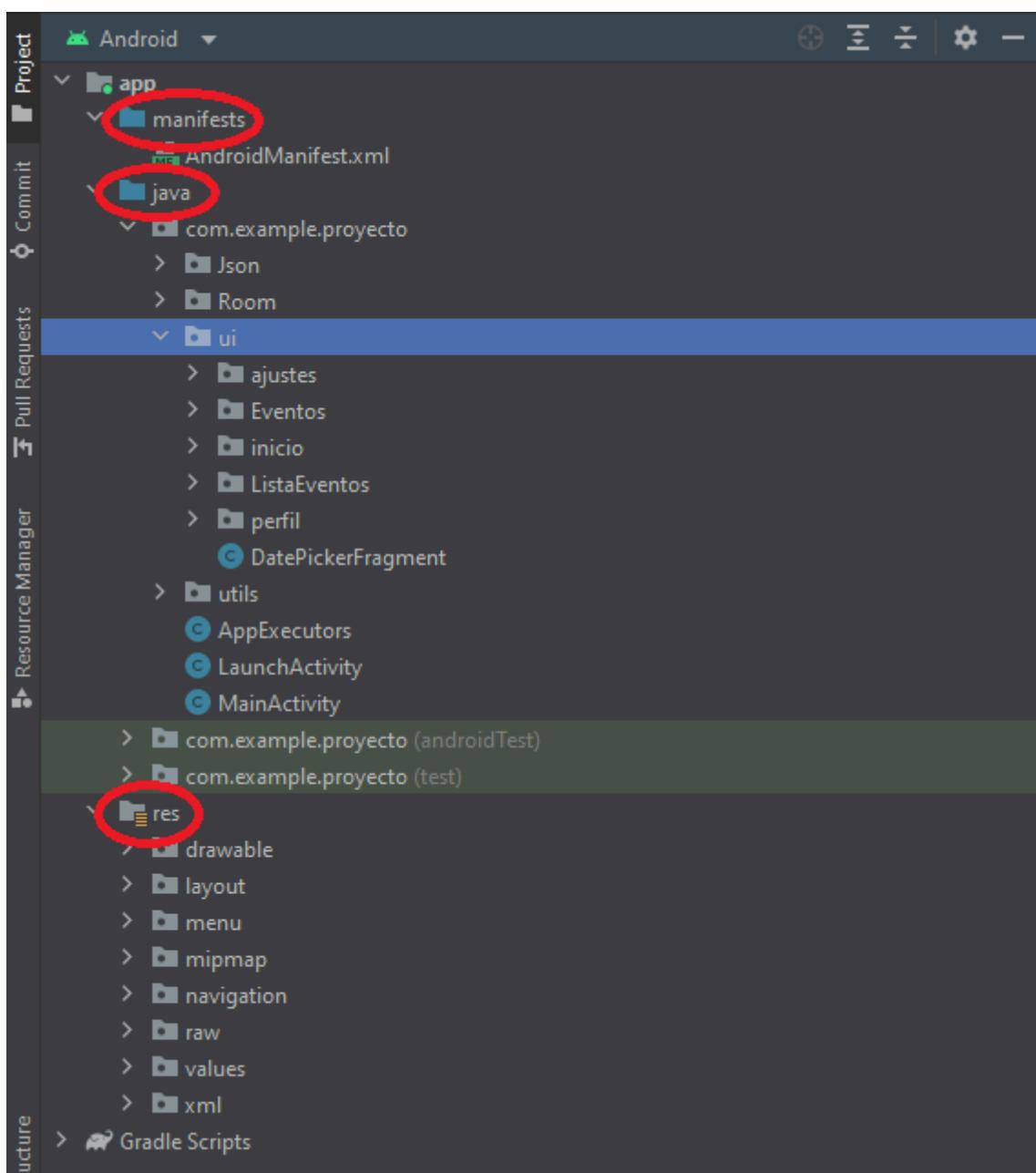
- Un sistema de compilación flexible basado en Gradle
- Un emulador rápido y cargado de funciones
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android
- Aplicación de cambios para insertar cambios de código y recursos a la app en ejecución sin reiniciarla
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de muestra
- Variedad de marcos de trabajo y herramientas de prueba
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones, entre otros
- Compatibilidad con C++ y NDK
- Compatibilidad integrada con [Google Cloud Platform](#), que facilita la integración con Google Cloud Messaging y App Engine

Así mismo, ofrece la funcionalidad de acceso y manejo de bases de datos gracias al soporte con Room, así como la posibilidad de ejecutar un programa en un dispositivo con AVD que va desde teléfonos inteligentes de varios modelos y marcas, hasta tablets, televisiones y ordenadores, entre otros.

## Estructura de un proyecto en Android Studio

En concreto, un proyecto en android studio se compone de las siguientes partes:

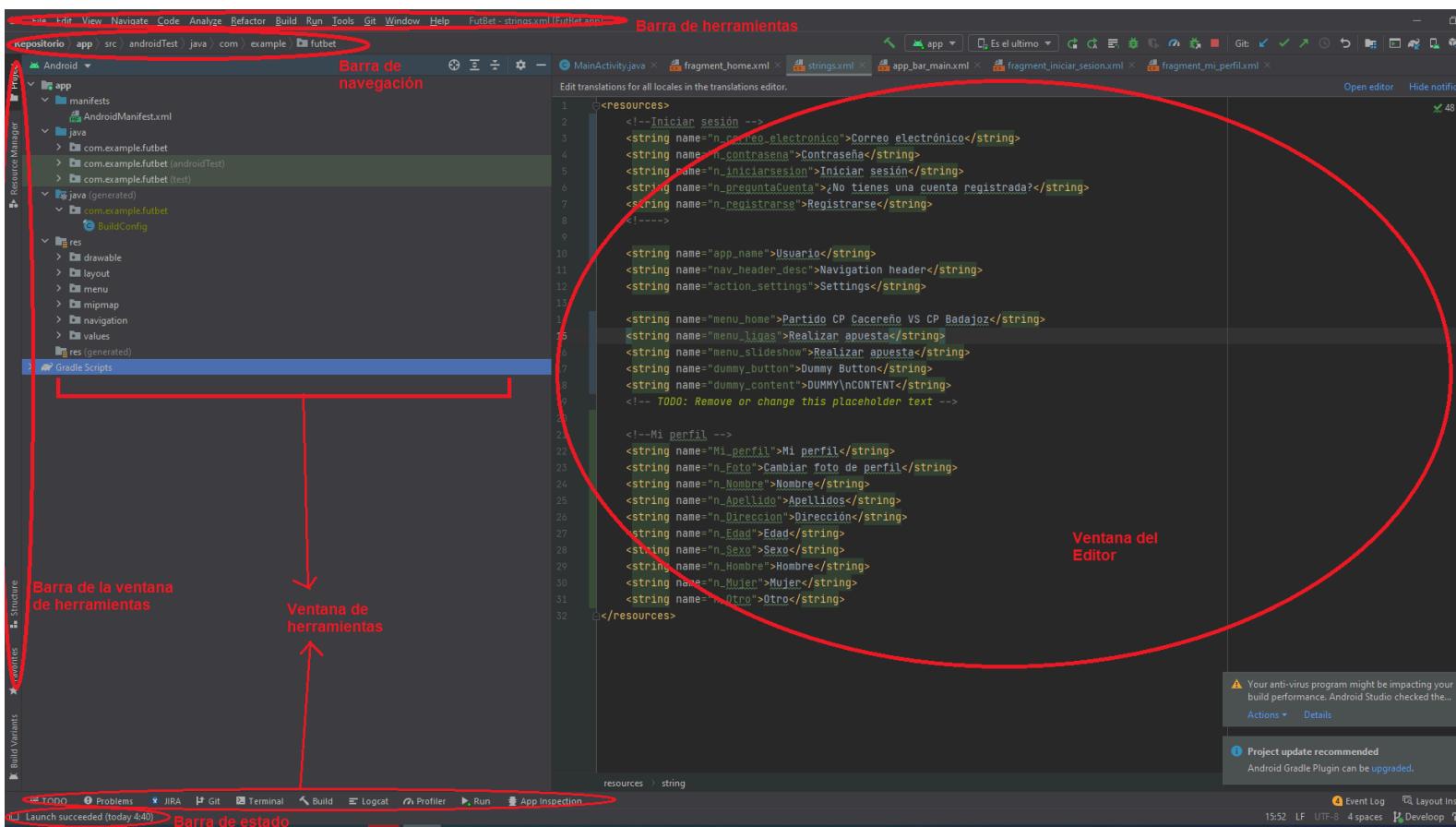
- **manifests**: consiste en un fichero llamado AndroidManifest.xml que contiene información adicional de la aplicación como el nombre de la aplicación, el icono que utilizará el launcher, qué actividad es la principal, los parámetros del intent (intent-filters) por los que se puede filtrar la aplicación, entre otros.
- **java**: contiene los archivos de código fuente Java, incluido el código de prueba de JUnit, este suele constar en clases y enumeraciones en Java, así como las diferentes actividades y fragmentos de los que se componen las interfaces de usuario.
- **res**: contiene todos los recursos sin código, como diseños XML, strings de IU e imágenes de mapa de bits, suele ofrecer todas las imágenes y prediseños que utilizarán los proyectos como recursos.



A continuación se muestran las diferentes vistas que componen la interfaz del programa.

## Componentes de la interfaz de Android Studio

Android Studio ofrece una serie de vistas y componentes en su interfaz que ofrecen información sobre un proyecto. Dentro de una ventana, se encuentran los siguientes componentes:



- La **barra de herramientas** te permite realizar una gran variedad de acciones, como ejecutar tu app e iniciar las herramientas de Android.
- La **barra de navegación** te ayuda a explorar tu proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana **Project**.
- La **ventana del editor** es el área en la que puedes crear y modificar código. Segundo el tipo de actividad actual, el editor puede cambiar. Por ejemplo, cuando ves un archivo de diseño, el editor muestra el Editor de diseño.
- La **barra de la ventana de herramientas** se encuentra afuera de la ventana del IDE y contiene los botones que te permiten expandir o contraer ventanas de herramientas individuales.
- Las **ventanas de herramientas** te brindan acceso a tareas específicas, como la administración de proyectos, la búsqueda, el control de versiones, entre otras. Puedes expandirlas y contraerlas.
- En la **barra de estado**, se muestra el estado de tu proyecto y el IDE, además de advertencias o mensajes.

## Funcionamiento de una App en Android Studio

Este software utiliza las llamadas actividades (Activity) para ejecutarlas como pantallas individuales. Estas son archivos en código Java que indican el funcionamiento de todos los eventos que suceden y las acciones que se realizan dentro de una pantalla de la aplicación.

Estas tienen el diseño indicado en el layout de dicha actividad, que consiste en un fichero xml que se encuentra en la carpeta res > layout, y que codifica cómo se mostrará la pantalla y los elementos que compondrán la interfaz.

Adicionalmente existen los fragmentos (Fragment), que consisten en ficheros java que permiten contener el funcionamiento de una actividad en una pantalla, permitiendo existir varios fragmentos en una misma Activity. Estos se pueden intercambiar por otros en cualquier momento, permitiendo a la aplicación un comportamiento más dinámico.

Una actividad se puede comunicar con otra a través de los llamados Intent, que son clases que permiten a una actividad llamar a otra. Cada instancia de la clase “intent contiene la clase de la actividad que llama a la siguiente, la clase de la actividad llamada, y otros datos adicionales como información que se quiere pasar en la llamada.

Una aplicación puede obtener los recursos como imágenes o iconos a través de la carpeta drawable, dentro de res. Esta contiene los recursos que se utilizan referente a los iconos, imágenes predeterminadas, etc.

Finalmente, cabe destacar el fichero strings.xml, que contiene todas las cadenas que se mostrarán en la aplicación de forma dinámica. Pudiendo cambiarse sin tener que acceder al código. Esto dota de independencia al sistema de asignación de cadenas ya que no necesitan modificar directamente el código para cambiarse, teniendo que modificar únicamente el fichero.

Para acceder al archivo strings.xml debe escribirse la cadena @string/ seguido del identificador que se haya puesto en el campo name de la etiqueta <string> que identifica a la cadena que se quiere insertar.

En el siguiente apartado se explicará el funcionamiento del componente Gradle.

## Compilación en Gradle

Android Studio usa **Gradle** como base del sistema de compilación, y el [complemento de Android para Gradle](#) proporciona capacidades específicas de Android. Este sistema de compilación se ejecuta en una herramienta integrada desde el menú de Android Studio, y lo hace independientemente de la línea de comandos. El propósito de las funciones del sistema de compilación es el siguiente:

- Personalizar, configurar y extender el proceso de compilación
- Crear varios APK para tu app; diferentes funciones usan el mismo proyecto y los mismos módulos
- Volver a usar códigos y recursos en conjuntos de archivos fuente

Gracias a la flexibilidad de Gradle, se puede lograr sin modificar los archivos fuente de la aplicación. Los archivos de compilación de Android Studio se denominan build.gradle. Son archivos de texto sin formato que usan la sintaxis [Groovy](#) a fin de configurar la compilación con elementos que proporciona el complemento de Android para Gradle.

Cada proyecto tiene un archivo de compilación de nivel superior para todo el proyecto y archivos de compilación de nivel de módulo independientes para cada módulo. Cuando se importa un proyecto existente, Android Studio genera automáticamente los archivos de compilación necesarios.

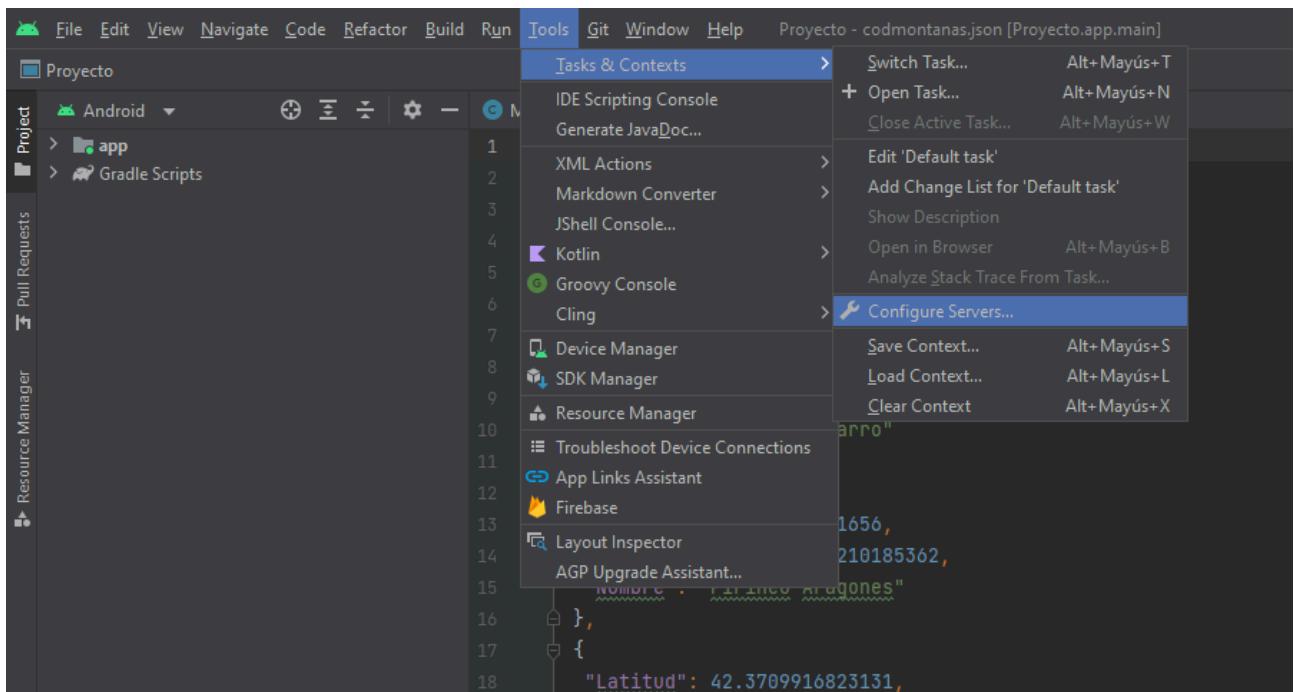
## Integración de Jira en Android Studio

Con el objetivo de poder informar sobre las tareas de implementación o integración que se realizan en el proyecto se ha optado por la sincronización del programa de **Android Studio** con el software **Jira**, de manera que a lo largo de la realización del proyecto se puedan actualizar el estado de las tareas a través del propio Android Studio.

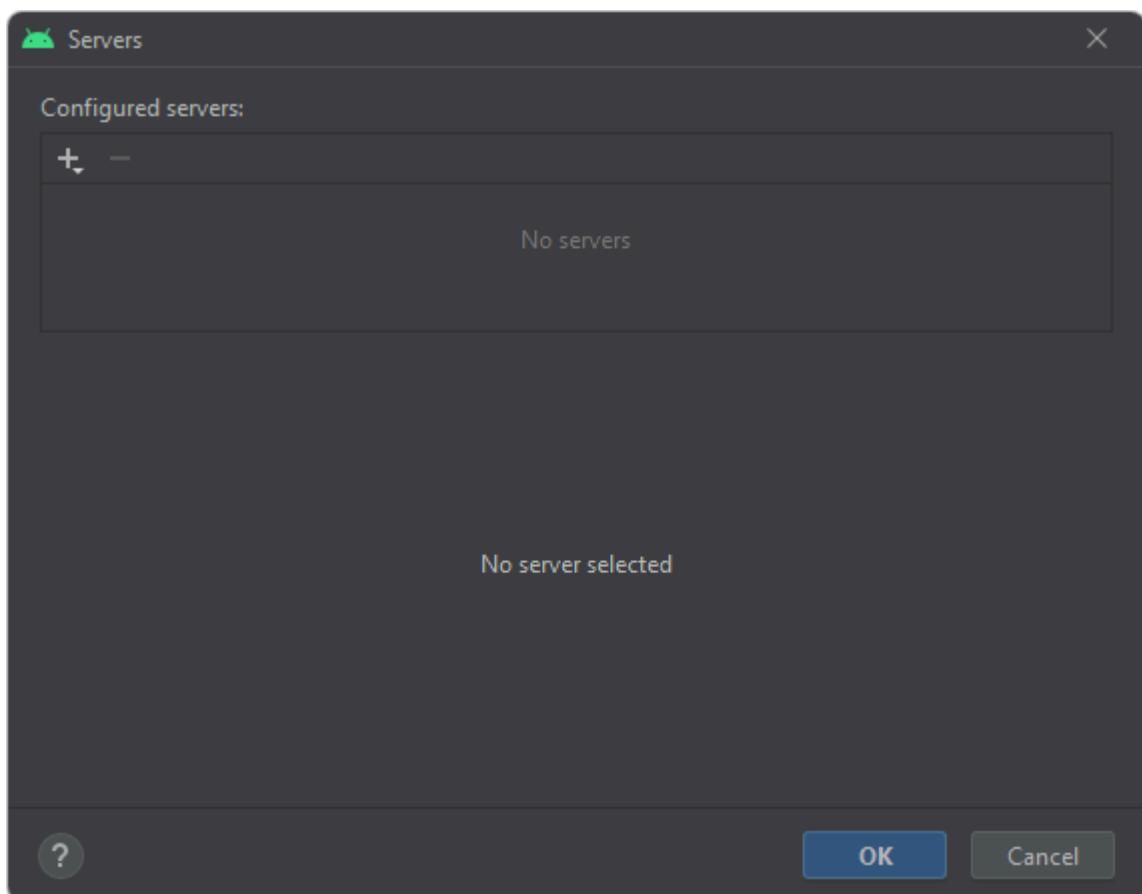
Para llevar a cabo la integración de Jira con el software de Android Studio se pueden seguir dos caminos:

1. Utilizando la herramienta de contexto que viene predefinida dentro de Android Studio.

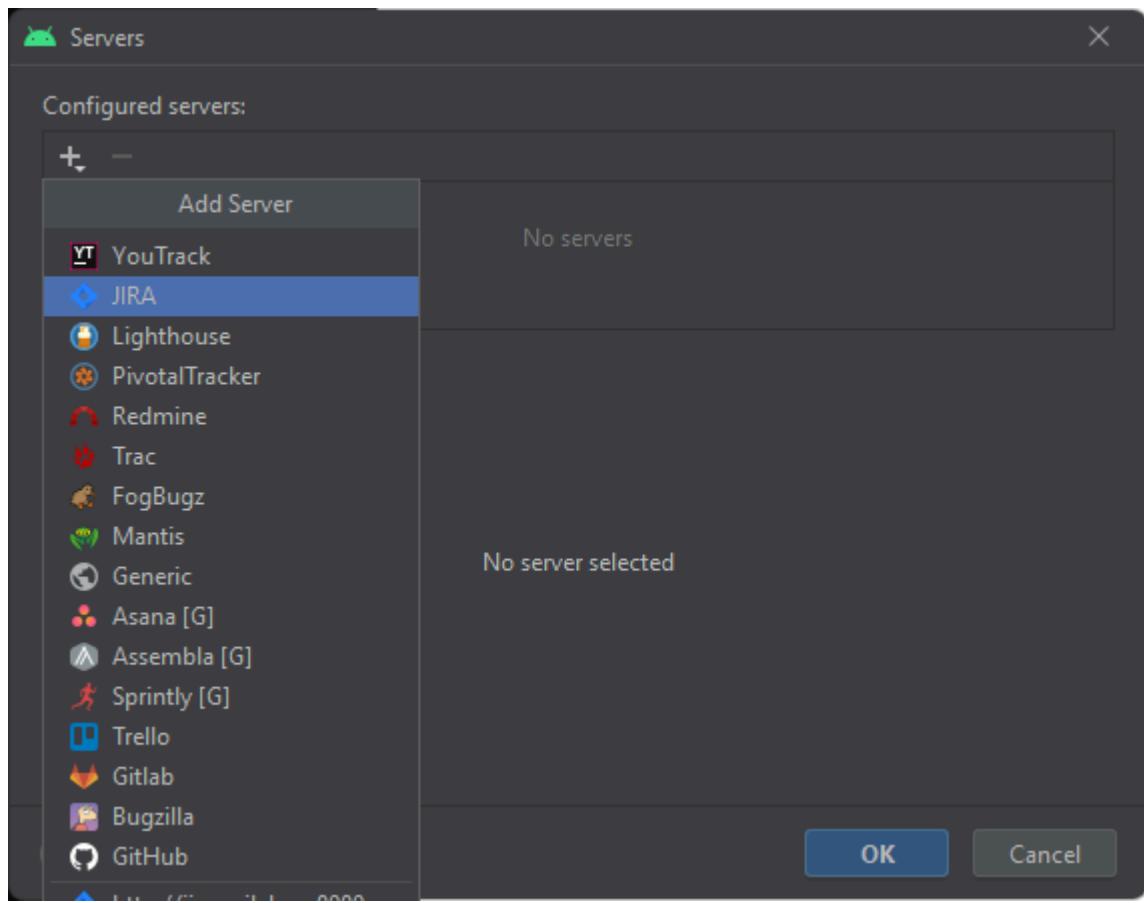
Lo primero que debemos hacer es configurar el servidor de Jira, para ello deberemos irnos a Tools > Tasks & Contexts > Configure Servers..., que se encuentra en la parte superior izquierda del programa.



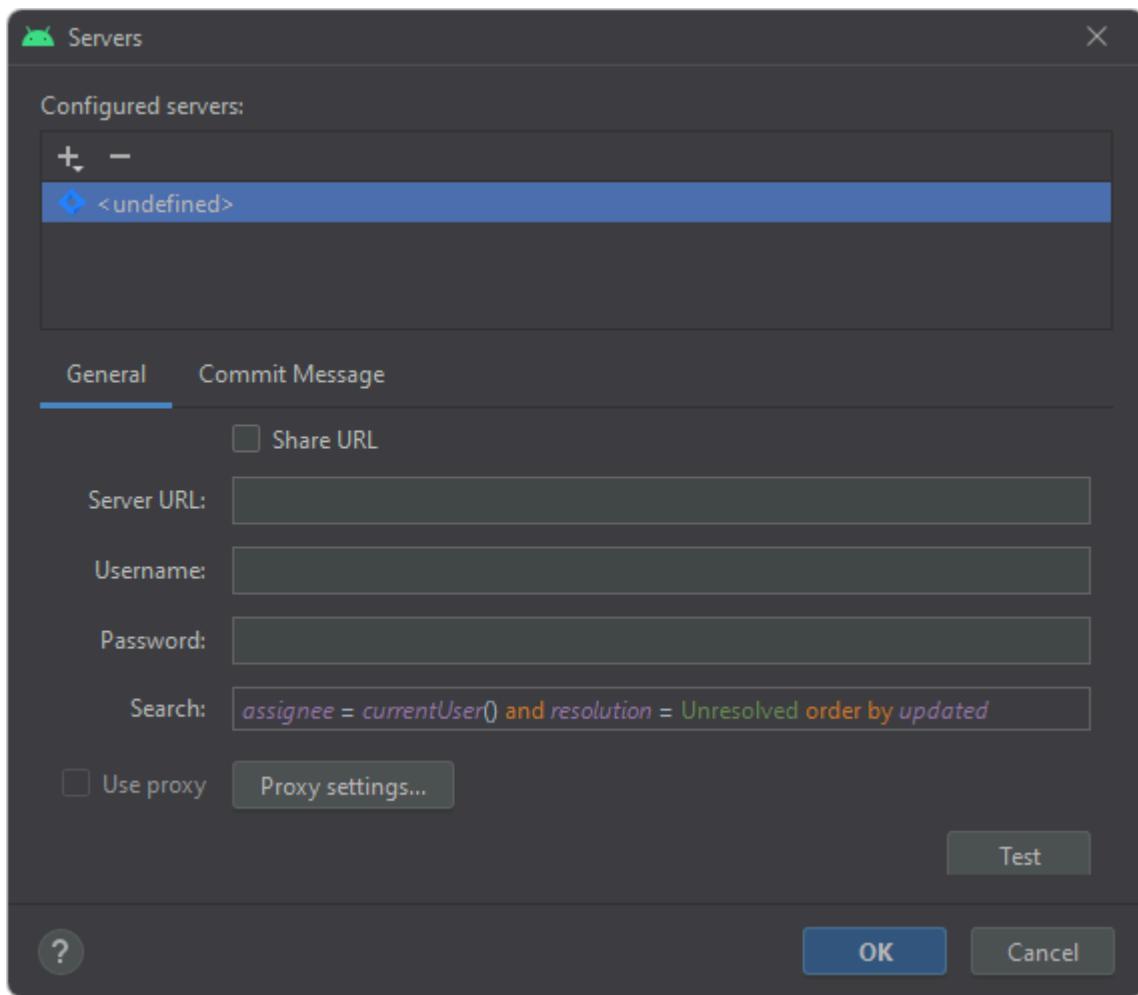
Una vez accedido a este apartado debería de aparecer una ventana de diálogo como la siguiente.



Dentro de esta ventana deberemos seleccionar el botón de "+" y seleccionar el tipo de Server al cual queremos acceder, en nuestro caso será el Server de Jira.



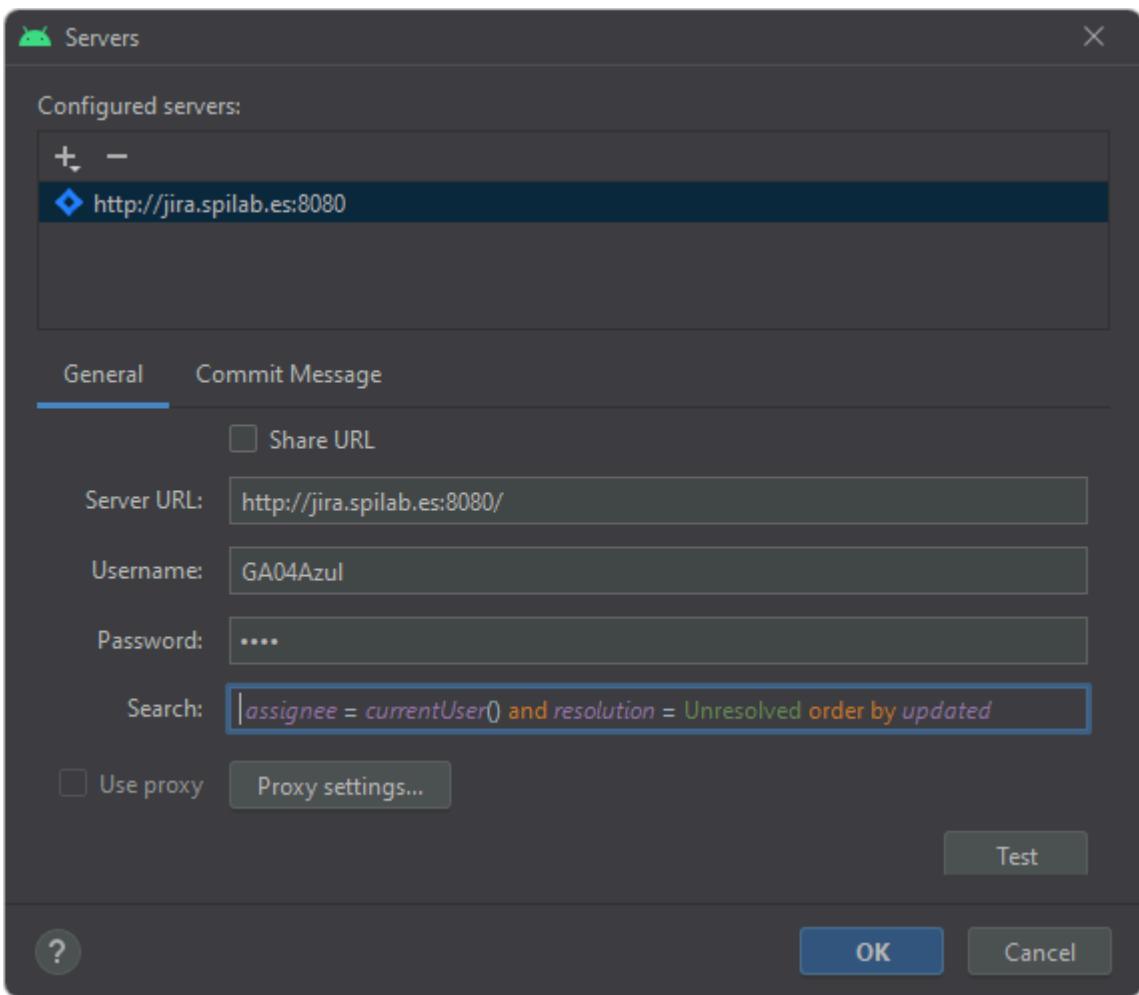
Una vez seleccionado el servidor aparecerán los registros que deberemos de llenar para acceder a este.



Estos atributos son los siguientes:

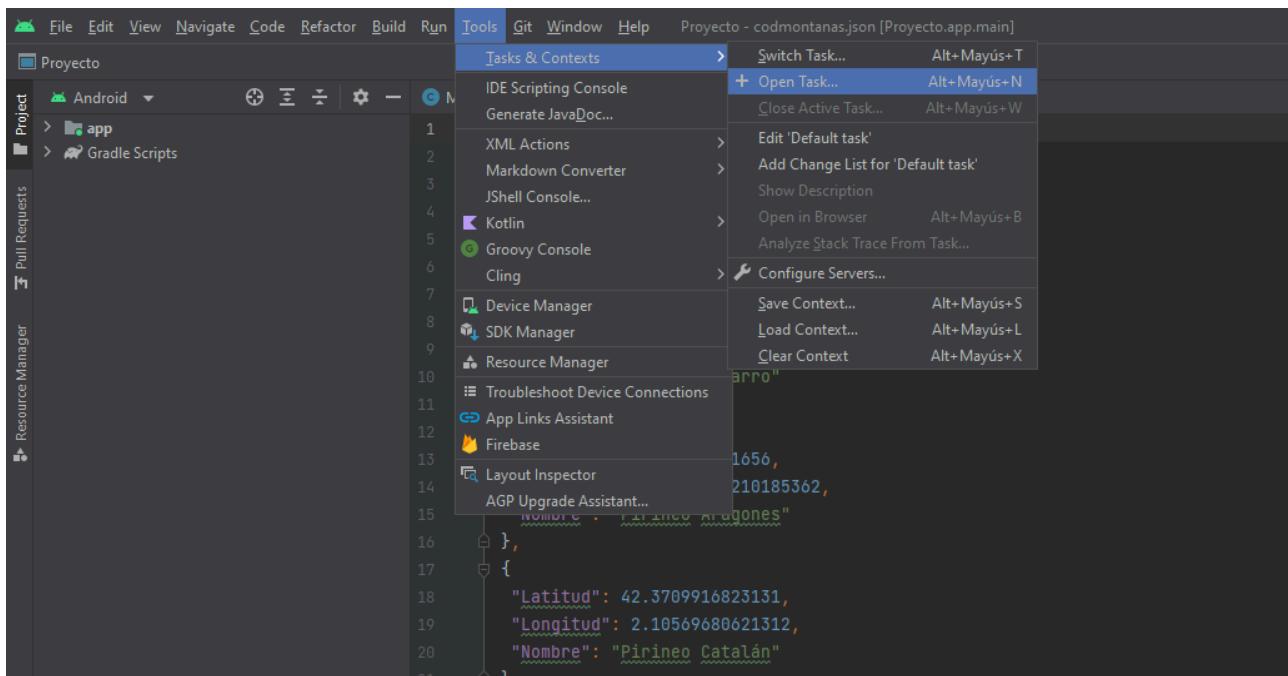
- **Server URL:** La **url** en la que se encuentra el servidor.
- **Username:** El **nombre de usuario** de la conexión que se va a realizar.
- **Password:** La **contraseña** asociada a dicho nombre de usuario.
- **Search:**

Esta debe rellenarse con los siguientes datos

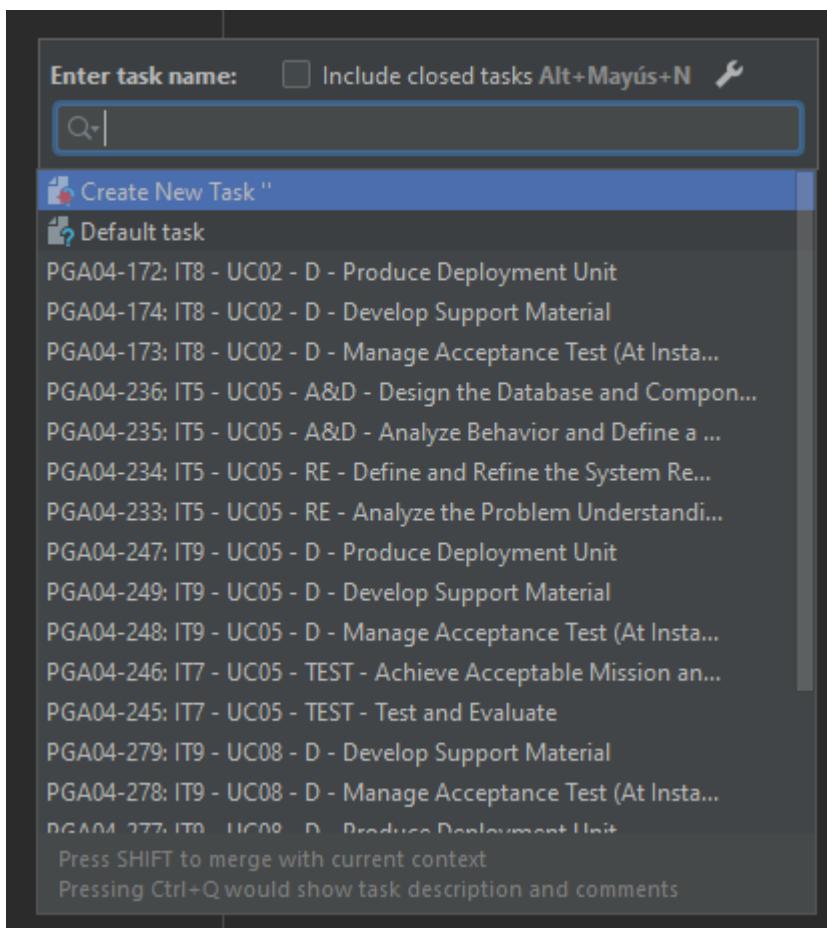


Una vez dado al botón 'ok', y si los datos se han puesto de forma correcta se accederá a servidor de Jira.

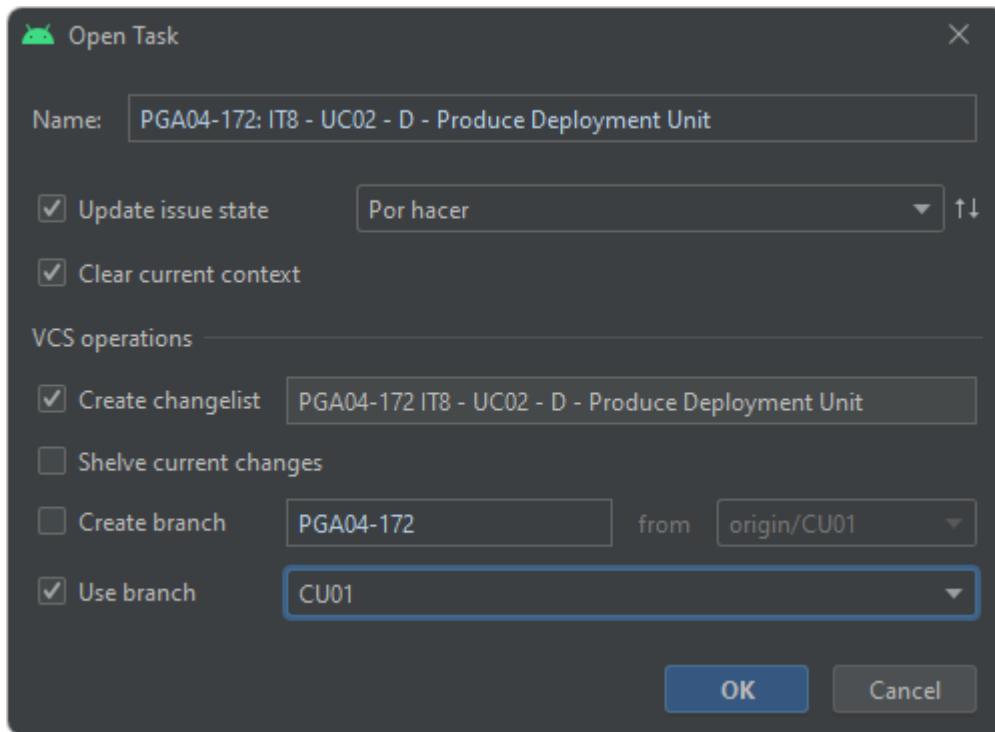
Ahora podremos empezar a realizar tareas, para ello deberemos de acceder a la lista de tareas a través de Tools > Tasks & Contexts > Open Task...



Una vez hecho esto podremos seleccionar la tarea de gira que queremos empezar a desarrollar buscándola dentro de la lista que nos aparece.



Al seleccionarla nos aparecerá una ventana similar a la siguiente, en la cual deberemos de elegir en qué rama vamos a realizar dicha tarea.



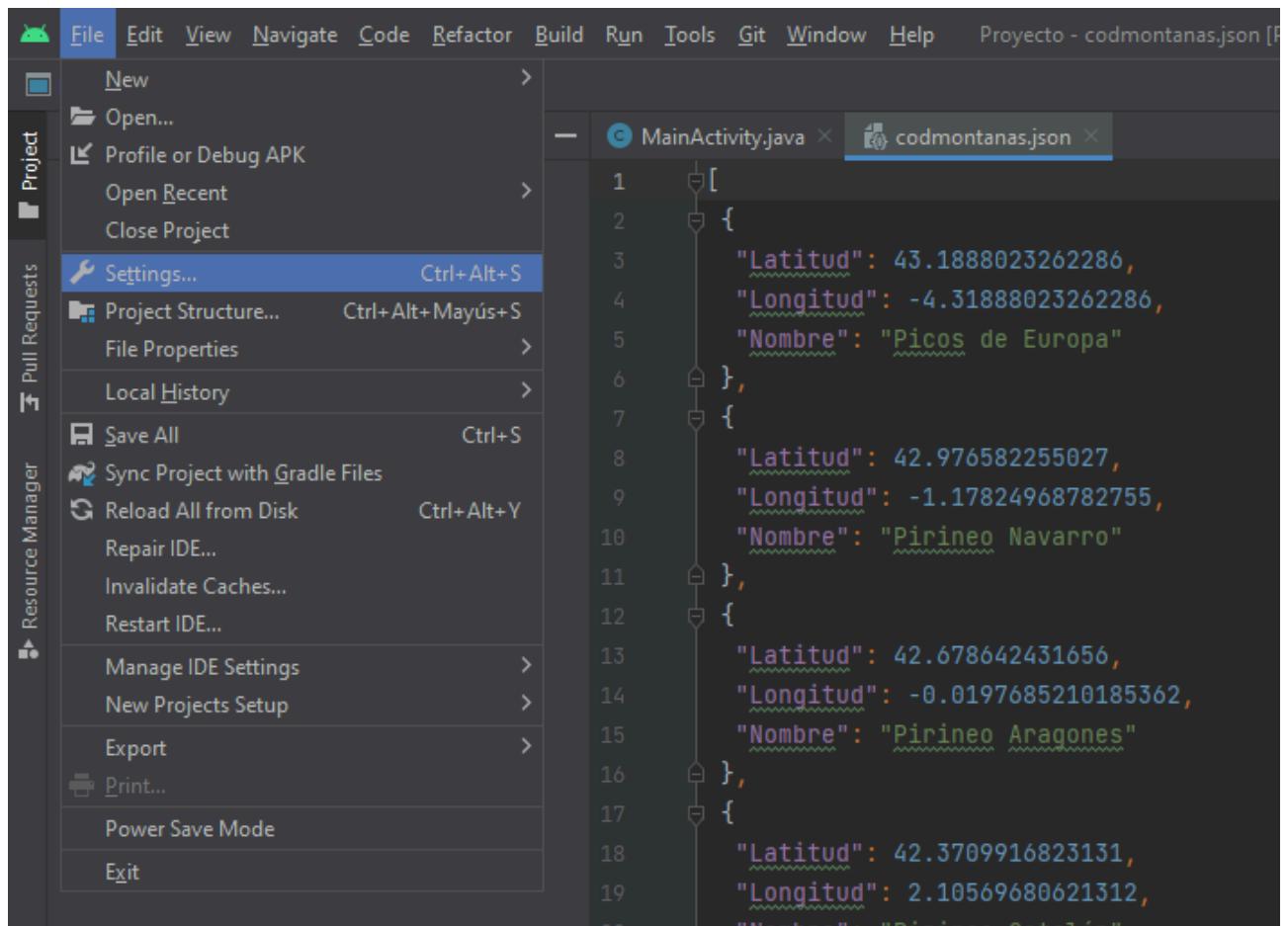
Una vez acabemos de trabajar, tan solo tendremos que realizar un commit en el cual indicamos el id de la tarea, su título y el autor que la realiza. Además de esto se pueden añadir más cosas como un comentario o el tiempo dedicado a dicha tarea utilizando Smart commits, pero no es necesario.

## 2. Utilizando un plugin que nos permite interaccionar con Jira a través de Android Studio.

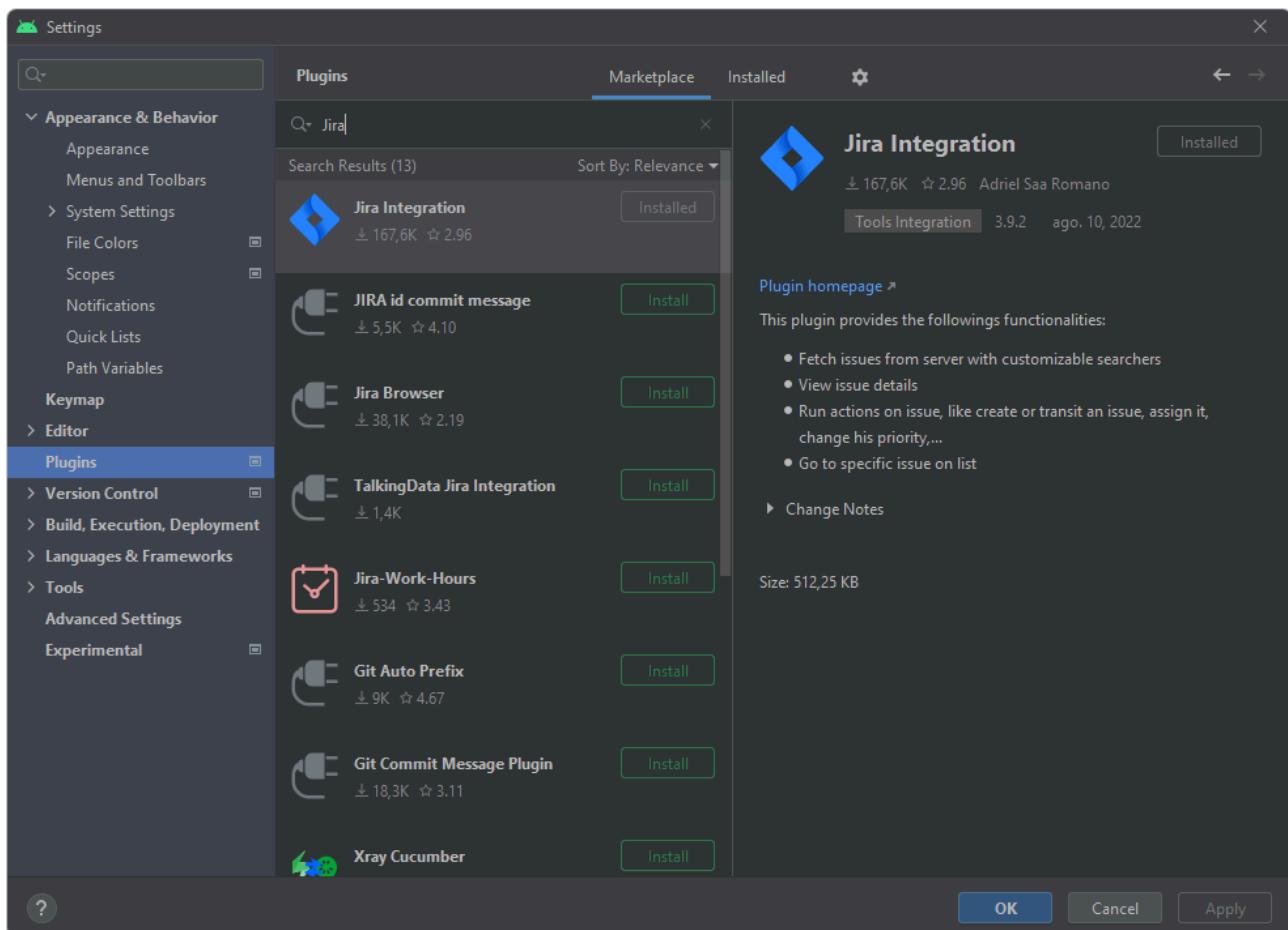
Esta opción contempla el uso de un plugin de Android studio que permite gestionar las tareas de un proyecto de Jira, de manera que se podrá actualizar el estado de estas y asociarlas a ramas de git concretas.

Por ello, se determinará que cada tarea de implementación estará asociada a una rama con el código de un caso de uso.

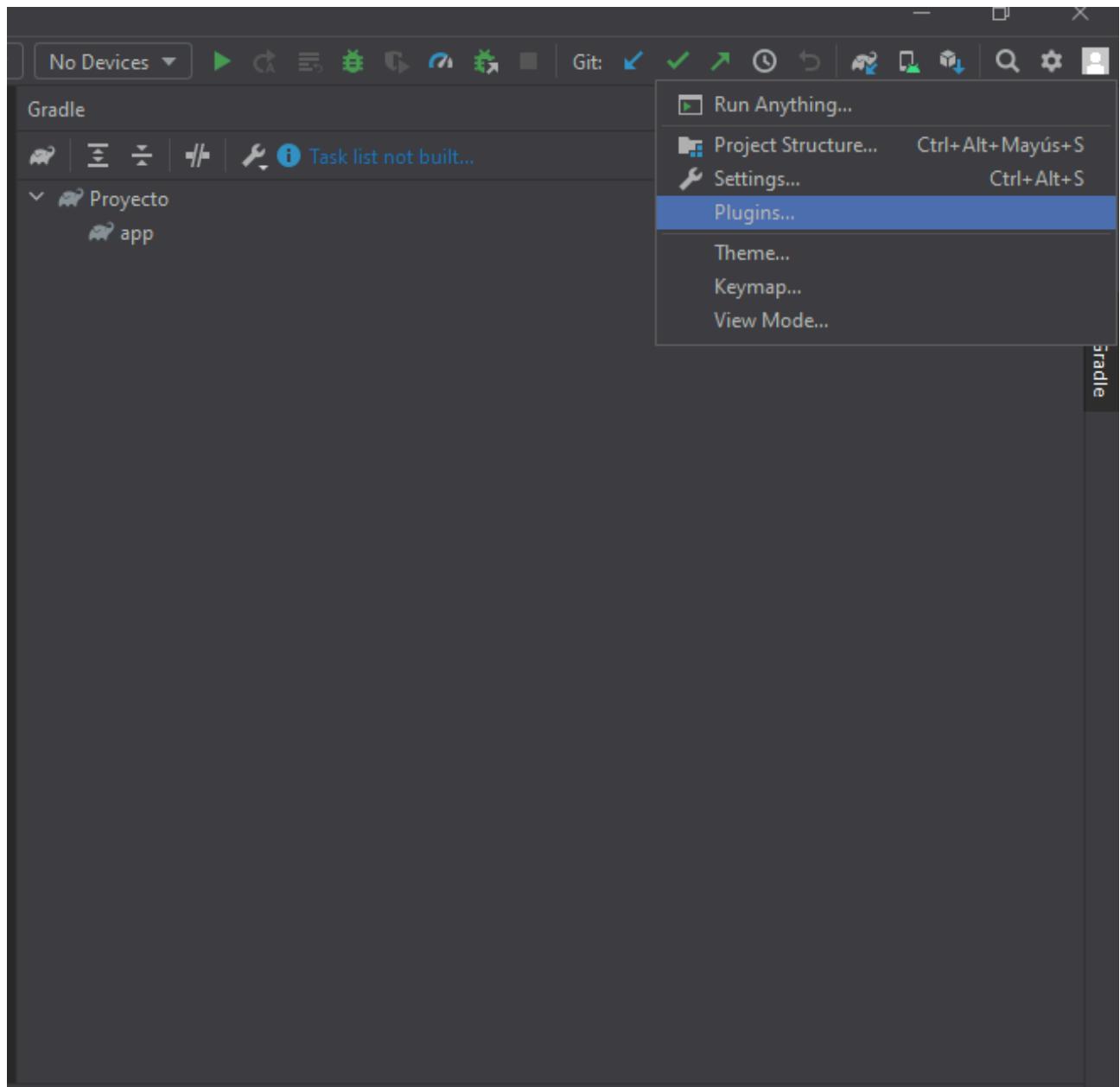
Para descargar el plugin, se debe acceder a la sección de plugins. Esto se puede realizar haciendo clic en la opción **Settings** de la pestaña **Files** y buscando en este dicha sección plugins.



Esto abrirá una ventana con las opciones de configuración de Android Studio.



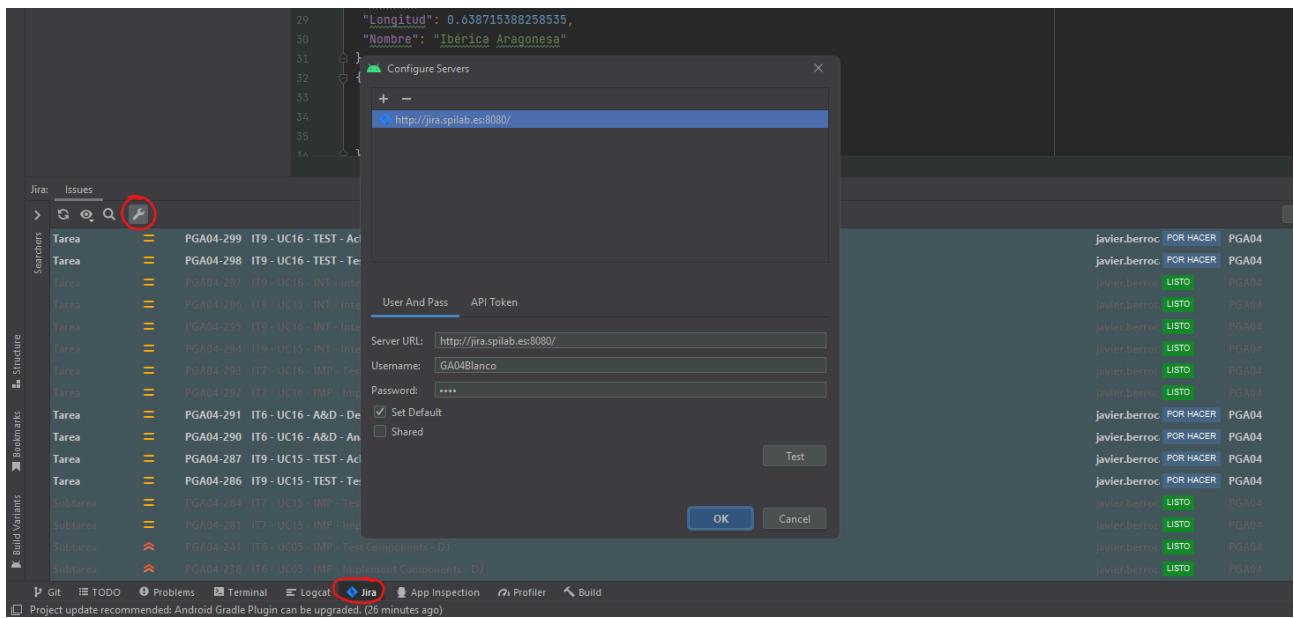
También es posible acceder a esta ventana a través del ícono con forma de rueda situado en la parte superior derecha de Android Studio, pulsando posteriormente en la opción **plugins**.



La ventana de plugins posee 2 pestañas, una para buscar un plugin dentro de todos aquellos que están instalados y otra para buscar todos los plugins y poder instalar nuevos.

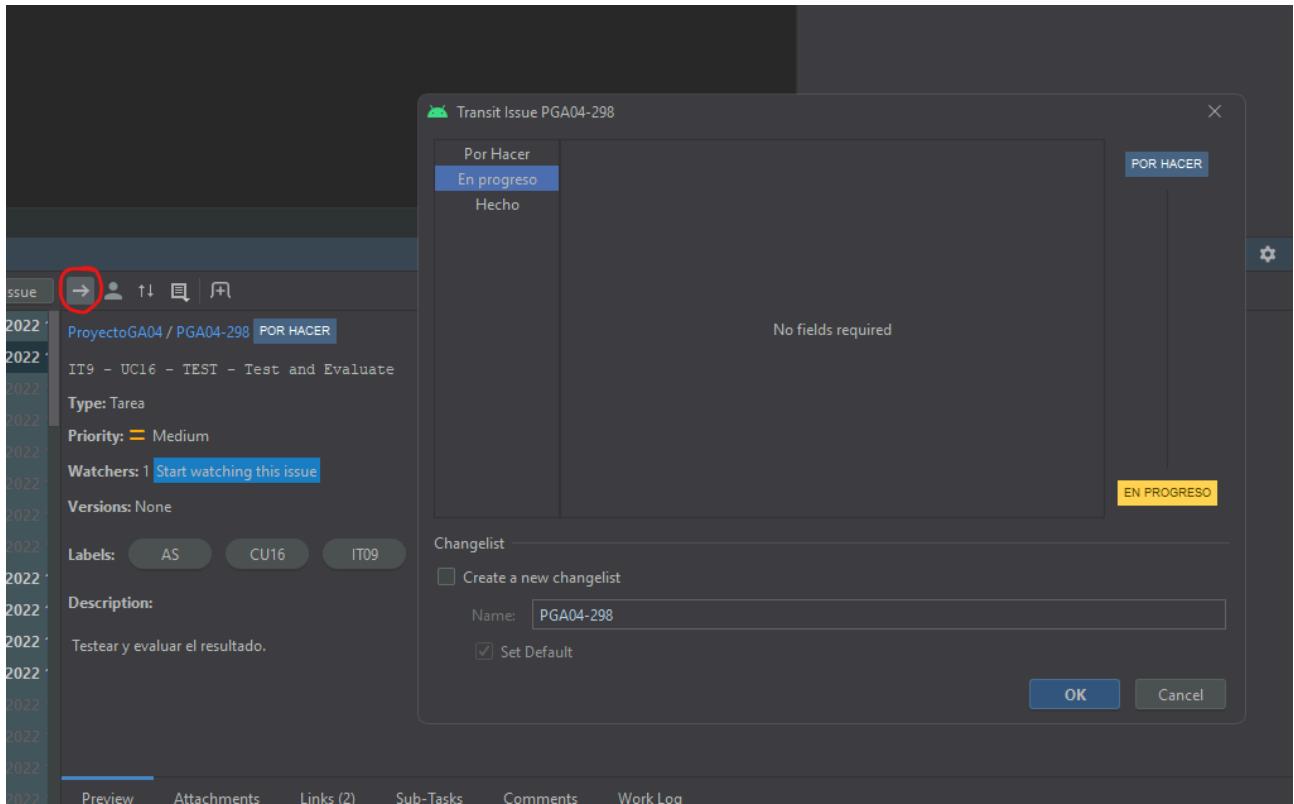
En la pestaña de instalación de nuevos plugins (**Marketplace**) se debe buscar el plugin llamado **Jira Software**, haciendo clic en el botón **Install** de este y comenzando su instalación.

Una vez instalado, nos aparecerá en la parte inferior, en la cual si presionamos el botón de configuración nos aparecerá una ventana para conectarnos al servidor de Jira, la cual deberemos llenar con los siguientes datos:



- **Server URL:** La url en la que se encuentra el servidor.
- **Username:** El nombre de usuario de la conexión que se va a realizar.
- **Password:** La contraseña asociada a dicho nombre de usuario.

Con esto ya podemos trabajar con Jira seleccionando la tarea que queramos realizar y mediante el botón de **transit**, seleccionando la tarea como en progreso:



Este plugin además nos da todas las opciones que podemos realizar a través de la web dentro de Android Studio, lo que nos permite prescindir del uso del navegador.

## Ramas utilizadas en el proyecto

En concreto, en este proyecto se utilizará una rama master que contendrá el código final de la aplicación, además de una rama develop, que contendrá una rama por cada caso de uso con el código de la implementación de estos.

Inicialmente, la estructura de estas ramas era la siguiente:

- master
  - develop
    - cu01
    - cu02
    - cu03
    - cu04
    - cu05
    - cu06
    - cu07
    - cu08
    - cu09
    - cu10
    - cu11
    - cu12
    - cu13
    - cu14
    - cu15
    - cu16

## Integración continua

La integración continua en el desarrollo del proyecto es el conjunto de prácticas que consisten en hacer integraciones automáticas de compilación y ejecución de test, para detectar fallos lo antes posible. De acuerdo a las tareas de integración definidas en JIRA, se usa la tecnología GIT sobre la plataforma GitHub para completar la integración continua entre las ramas de los casos de uso CU01-CU16 con la rama develop, y entre la rama develop con la rama master.

La integración continua es una práctica moderna que facilita el trabajo a los desarrolladores sobre el código de un mismo proyecto. Algunas ventajas que presenta son las siguientes:

- Los desarrolladores pueden detectar y solucionar problemas de integración de forma continua, evitando el caos de última hora cuando se acercan las fechas de entrega.

- Disponibilidad constante de una versión para pruebas, demos o lanzamientos anticipados.
- Ejecución inmediata de las pruebas unitarias.
- Monitorización continua de las métricas de calidad del proyecto.

Esta disciplina consistirá en 2 tareas:

- Integración de cada subsistema o “Integrate each subsystem”
- Integración del sistema “Integrate the system”

## Definición de disciplinas

En este apartado se explicará el proceso que seguirá en cada disciplina relativa al desarrollo de la aplicación, siendo estas **Implementación e Integración y Testeo**.

### Implementación

Esta disciplina se divide en 2 tareas:

- “Implement components”
- “Test components”

Este proceso consistirá únicamente en la disciplina de **“Implement components”**

Esta corresponde a la implementación del caso de uso correspondiente utilizando Android Studio a quien tenga asignada dicha implementación según la planificación del Jira.

Para ello, el rol asignado debe marcar en Jira la tarea correspondiente del estado **TO-DO** al estado **In Progress**. El código debe contener tanto las pantallas (Activitys y Fragments) que muestran el código, como los componentes de la lógica de negocio (clases), así como las posibles interfaces DAO que deban intervenir.

Todo el código deberá funcionar correctamente y contener toda la funcionalidad completa de ese caso de uso.

Una vez finalizada la implementación del código en Android Studio, se deberá marcar el estado de la tarea correspondiente a **DONE**. Confirman los cambios realizados en este desde el inicio de la implementación mediante el lanzamiento de un commit, que guardará los cambios realizados en el repositorio local.

Este commit se deberá enlazar con la tarea correspondiente en el jira, guardando los cambios realizados y relacionándolos con esta.

### Integración y Testeo

La tarea **Integrate each subsystem** consistirá en recuperar el código depositado en el repositorio que está ubicado en la rama Develop a la rama correspondiente del caso

de uso, permitiendo mediante la realización de un **Pull** trabajar sobre los cambios ya realizados en la rama Develop.

Este proceso también debe actualizar la tarea del Jira correspondiente marcando su estado según si es **TO-DO** (por realizar), **In Progress** (realizándose) o **DONE** (realizada).

Por otra parte, la tarea “**Integrate the system**” consistirá subir en el repositorio remoto compartido por todos los integrantes el código del caso de uso ya completamente implementado mediante la realización de un **Push**, que lo publicará en la rama del repositorio correspondiente.

Una vez hecho esto en todas las ramas, se deben integrar todos los casos de uso de las ramas correspondientes en la rama Develop mediante la realización de un **merge**, que unificará todos los casos de uso realizados en la rama Develop, que posteriormente subirá todo el código integrado a la rama master.

Cabe destacar que también se debe relacionar este proceso con la tarea correspondiente en el Jira, modificando su estado como **TO-DO** (por realizar), **In Progress** (realizándose) o **DONE** (realizada).

## Integración de cada subsistema

Una vez finalizada la última tarea de implementación de un caso de uso y subido con un push a la rama remota “origin/CUxx”, siendo ‘xx’ el número del caso de uso, se procede a integrar dicho caso de uso con el resto del proyecto que se encuentra en la rama remota “origin/develop”.

En primer lugar, se debe realizar un merge del estilo <origin/develop into “CUxx”> y resolver los conflictos que aparezcan. De esta forma, se pretende unificar el caso de uso implementado con el nuevo código subido por otro equipo a la rama remota “origin/develop”.

En este instante, se dispone del código de los subsistemas integrados en el último caso de uso implementado junto con el commit resultante del merge. Estas tareas de integración no contienen un commit adicional, por tanto, desde Android Studio se modifica el estado de la tarea de integración de subsistemas a ‘Hecho’ y el autor de la tarea coincide con el usuario logueado en el servidor de JIRA en Android Studio.

## Integración del sistema

Una vez finalizada la integración de cada subsistema, se procede a cometer la siguiente tarea de integración desbloqueada llamada integración del sistema. Primero, se hace un checkout de la rama “develop” para tenerla en local. En ella, se procede a hacer un nuevo merge del estilo <“CUxx” into “develop”> para mantener en la rama local “develop” la integración del sistema, exenta de errores. Por último, queda hacer un push de la rama local “develop” en la remota “origin/develop” para finalizar la tarea de integración de sistema.

En este instante, si se comete una nueva tarea de integración de cada subsistema de otro caso de uso, el miembro del equipo responsable de dicho caso de uso tendrá que integrar su subsistema al resto de subsistemas, incluido el nuestro.

Por último, cuando todas las ramas de los 16 casos de uso se encuentren implementadas e integradas en la rama remota “develop”, sólo queda integrar la rama “develop” con la rama remota “master”, quedando así finalizado el desarrollo de la aplicación Android, a espera de futuras mejoras y pruebas.

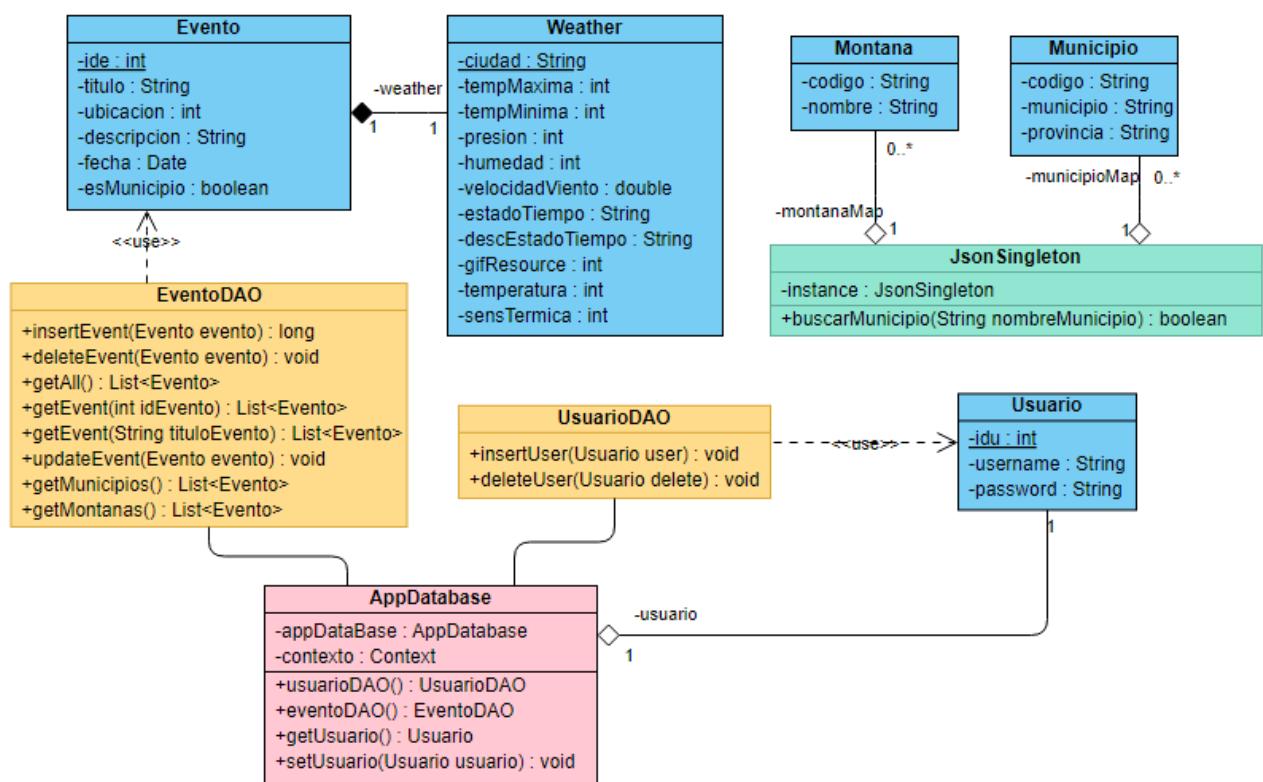
## Implementación

En este apartado se documentará todo el proceso de **implementación** que se ha seguido en el proyecto, desde su planificación hasta las clases implementadas que componen la aplicación, pasando por decisiones de implementación como su sintaxis, patrones de diseño, etc.

## Modelo de datos

En este apartado se detalla todo lo referente al modelo de datos de la aplicación Android ClimApp, datos provenientes tanto de la API como de JSON.

Este se puede apreciar en el siguiente diagrama de clases, que contiene los principales componentes del modelo de datos:



En primera instancia, existe una entidad en la Room llamada **Usuario** que se encarga de almacenar las credenciales procedentes de un usuario durante el registro. Contiene los siguientes campos:

- ‘idu’ (int). Contiene el identificador de una tupla de Usuario en la base de datos.
- ‘username’ (String). Contiene el nombre de usuario introducido durante el registro.
- ‘password’ (String). Contiene la contraseña introducida durante el registro.
- ‘conectado’ (booleano). Almacena el valor ‘true’ si el usuario se encuentra conectado en la aplicación, ‘false’ en caso de no estar logueado.

El usuario puede crear eventos en la aplicación. Existen dos tipos de eventos: aquellos cuya ubicación se encuentra en un municipio y aquellos cuya ubicación se encuentra en una montaña. Sin embargo, se ha tomado la decisión de no almacenar las condiciones meteorológicas de la ubicación de un evento (atributo de tipo Weather que se detalla posteriormente) en la base de datos, puesto que el tiempo meteorológico es dinámico y cambia respecto pasan las horas o días. Como consecuencia de ello, en la base de datos no se hace diferencia respecto a los dos tipos de eventos, sino que sendos eventos se representan con una **única entidad Evento** en el modelo de la Room.

Esta clase contiene los siguientes atributos:

- ‘ide’ (int). Contiene el identificador de una de las muchas tuplas de Evento en la base de datos.
- ‘titulo’ (String). Contiene el título del evento introducido durante su creación.
- ‘ubicacion’ (String). Contiene una cadena relativa a la ubicación del evento. Si es un evento de municipio, contiene el nombre del municipio. En caso de ser un evento de montaña, contiene el nombre de la montaña.
- ‘descripcion’ (String). Contiene la descripción del evento introducida por el usuario durante la creación del evento.
- ‘esMunicipio’ (boolean). Valor booleano que sirve para diferenciar en la misma entidad si es un evento de municipio o de montaña.
- ‘fecha’ (Date). Contiene la fecha en la que se da el evento. Tiene un formato del estilo ‘dd/MM/yy’. No se opta por contener las horas, minutos ni segundos. Este atributo de la entidad es sumamente importante, a partir de él se realizará una llamada a la API del tiempo para obtener las condiciones meteorológicas del evento en ese día.

Se requiere de una clase que almacene las condiciones meteorológicas al hacer una llamada a la API del tiempo, independientemente del tipo de evento (Municipio o Montaña) y de la petición del tiempo de la ubicación actual. Esta clase es la llamada **Weather** y contiene los siguientes atributos:

- ‘ciudad’ (String). Contiene el nombre del municipio o montaña de la ubicación sobre la que se obtiene el tiempo.

- ‘gifResource’ (int). Contiene un código numérico referente al estado del tiempo (“Lluvia”, “Soleado”, etc). Este código se mapeará dinámicamente en un GIF para personalizar la interfaz relativa al tiempo meteorológico.
- Los atributos relativos a las condiciones meteorológicas:
  - ‘temperatura’ (int)
  - ‘sensTermica’ (int)
  - ‘tempMinima’ (int)
  - ‘tempMaxima’ (int)
  - ‘presion’ (int)
  - ‘humedad’ (int)
  - ‘velocidadViento’ (double)
  - ‘estadoTiempo’ (String)
  - ‘descEstadoTiempo’ (String)

Se ha mencionado que los eventos tienen asignado una ubicación de montaña o municipio según lo considere el usuario durante su creación, que junto con la búsqueda del tiempo en la ubicación actual del dispositivo, es necesario incluir en la aplicación un archivo JSON que contengan todos los **municipios** de España y otro archivo destinado a las **montañas** de España.

- El JSON de municipios se ha obtenido de un archivo excel de la página oficial de AEMET. Como el excel incorporaba columnas innecesarias, se ha transformado para incluir el código de municipio: concatenación del código de provincia (3 dígitos) + código de municipio (2 dígitos), el nombre del municipio y el nombre de la provincia a la que pertenece. Seguidamente, se manejó un conversor online XLS to JSON para obtener el JSON final para poseer todos los municipios listos para usarse en la app.

<https://beautifytools.com/excel-to-json-converter.php>

- El JSON de montañas se ha creado manualmente (sólo existen 9 montañas en España) , como consecuencia de la API OpenWeather, la cual solo permite realizar una petición del tiempo de una montaña en base a sus coordenadas (longitud, latitud). Por lo tanto, siguiendo el mismo proceso que en el JSON de municipios, un objeto de montaña del JSON contiene la latitud, longitud y el nombre de la montaña.

Para convertir los objetos del JSON en objetos java y cargarlos en nuestra aplicación, se han diseñado dos clases java con la herramienta jsonSchema2Pojo, obteniendo las clases **Montaña** y **Municipio**.

<https://www.jsonschema2pojo.org>

## Detalles de implementación

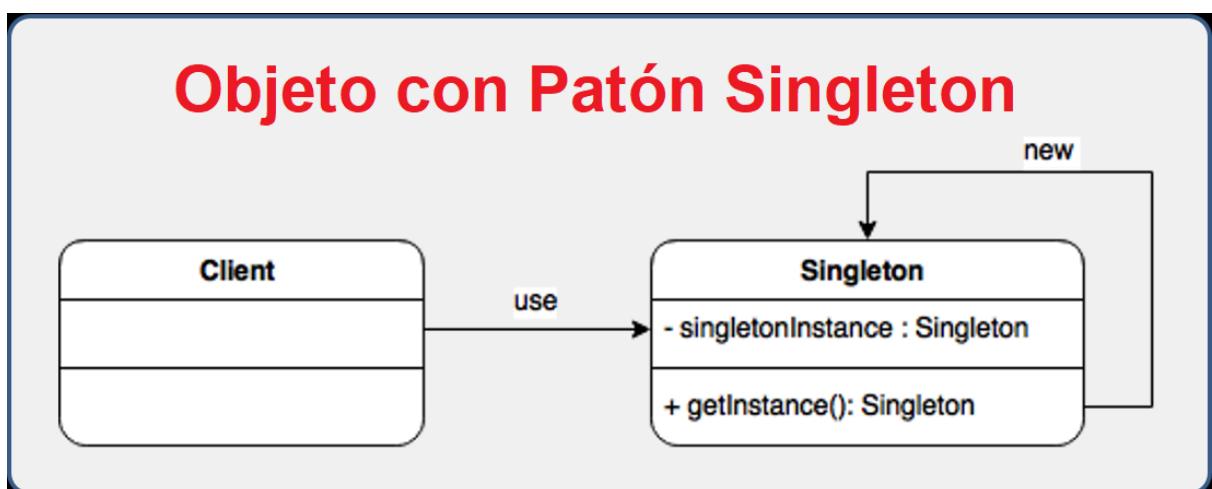
## Patrones de Diseño

A lo largo del proceso de desarrollo de este proyecto se han incluido una serie de Patrones de Diseño en su implementación, que expresan esquemas para definir estructuras de **diseño** (o sus relaciones) con las que construir sistemas de software. Facilitan la codificación, seguridad y consistencia a la aplicación durante su ejecución.

Los patrones de diseño utilizados se indican a continuación.

### Patrón Singleton

El patrón de diseño Singleton es un patrón de diseño creacional que gestiona la creación de objetos de una clase concreta, de manera que solo pueda existir una única instancia en tiempo de ejecución.



Esto se consigue gracias a que la clase que implementa este patrón sigue las siguientes condiciones:

- Posee un atributo privado y estático del tipo de la clase, que será la única instancia en tiempo de ejecución.
- El constructor de la clase es de tipo privado.
- Posee un método público y estático `getInstance` que devolverá el atributo de la clase declarado como privado. De forma que si el atributo aún no está inicializado (`atributo == null`) previamente se creará mediante el constructor privado. Y si ya está inicializado, lo devolverá directamente.

De esta forma, cada vez que se tenga que utilizar un objeto de esta clase, se invocará utilizando llamando al método `getInstance()` de forma estática desde el nombre de la clase (`Clase.getInstance()`), que devolverá siempre la misma instancia del objeto que utiliza el patrón.

Se ha utilizado el Patrón singleton en la creación de la clase `AppDatabase`, que crea y gestiona la base de datos, ya que con este la clase sólo podrá tener una única instancia en tiempo

de ejecución para no tener que inicializarla varias veces, y acceder siempre a la misma para modificar la base de datos.

Así mismo, también se ha utilizado en la creación de la clase **JsonSingleton**, que permite obtener todos los datos relativos al clima de las montañas y municipios cargándose de la API. De, esta forma, se pueda acceder a todos los municipios y montañas existentes junto con la información sobre su clima en cualquier parte del programa, permitiendo más facilmente la consulta de los datos de estos por el usuario o su asignación a eventos.

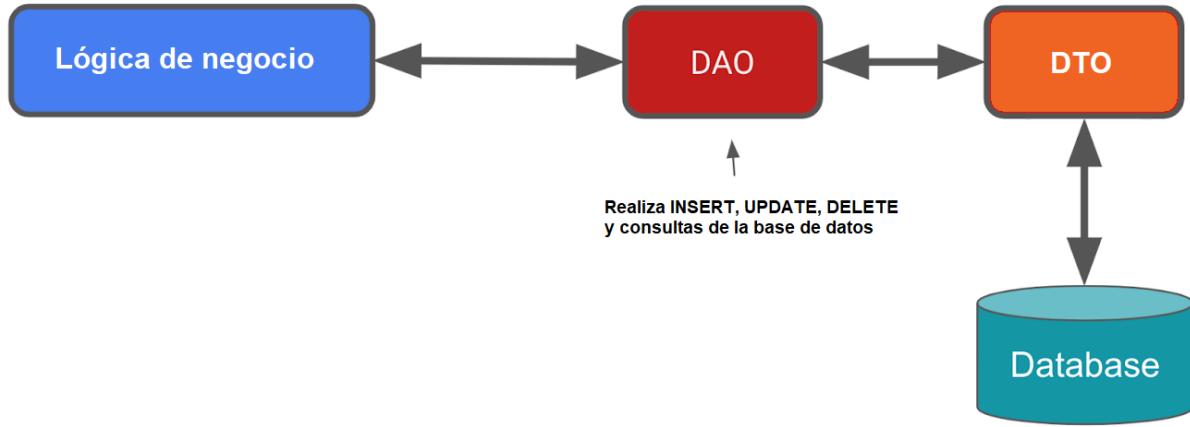
#### Ventajas del patrón Singleton

- La propia clase es responsable de crear la única instancia. Por medio de su método constructor.
- Permite el acceso global a dicha instancia mediante un método de clase.
- Declara el constructor de clase como privado para que no sea instanciable directamente.
- Al estar internamente **autoreferenciada**, en lenguajes como Java, el recolector de basura no actúa.
- Se puede ejercer un **control preciso** sobre cuándo y cómo se accede a él.

## Patrón DAO

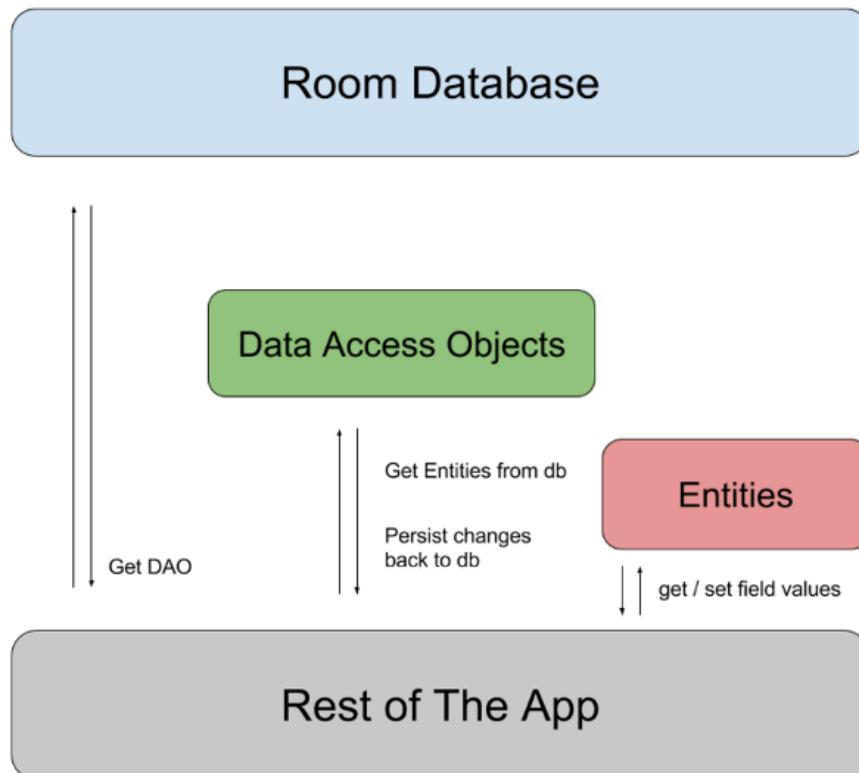
El patrón de diseño Data Access Object (DAO) es un patrón de diseño Arquitectónico que permite gestionar el desarrollo de una aplicación que utiliza bases de datos (persistencias de datos) al separar todos los componentes del sistema en 3 tipos bien definidos:

- Componentes relacionados con el **modelo de datos** y la lógica de negocio (clases del diagrama)
- Componentes destinados a la **transferencia de datos**, encargados de conectarse a la base de datos o modificarla. Implementan el patrón de diseño **Data Transference Object (DTO)**.
- Componentes destinados al **acceso a datos**, conectados con los de transferencia de datos para enviar la información a la lógica de negocio, aislando los detalles de implementación. Se tratan de interfaces que son implementadas por los componentes de transferencia, permitiendo ocultar detalles concretos sobre la implementación de la conexión a la base de datos.



Este patrón permite **separar la lógica de datos** de la forma de acceder a estos con conexiones a la base de datos, pues los componentes de acceso a datos son capaces de implementar varios tipos de componente de transferencia de datos, lo que permite **modificar el tipo de acceso** a la base de datos sin afectar la aplicación, pudiendo incluso **tener varios tipos de acceso a datos** al mismo tiempo.

Este patrón se ha utilizado durante la inclusión del servicio Room en Android Studio, ya que este utiliza las interfaces DAO creadas para implementar los componentes de transferencia de datos, que almacenarán la información en las clases de la lógica de negocio (declaradas como *Entity*), conectándose a la base de datos a través de la clase AppDatabase.



Esta **clase AppDatabase**, que deberá ser abstracta, se utilizará para recuperar información y modificar la base de datos, pues devolverá las interfaces DAO que contienen la información y pueden manipular la base de datos.

A su vez, **las interfaces DAO** modificarán y accederán los valores de los componentes del modelo de datos a través de los métodos getters y setters definidos de estos. Así mismo, estos componentes DAO contarán con métodos que según como se marquen realizan distintas operaciones en la base de datos, siendo la inserción (*Insert*), modificación (*Update*), borrado (*Delete*) y consulta (*Query*).

Por otra parte, **los componentes de la lógica de negocio** serán las clases marcadas como *Entity*, que compondrán la estructura de la base de datos y almacenarán la información recuperada de esta.

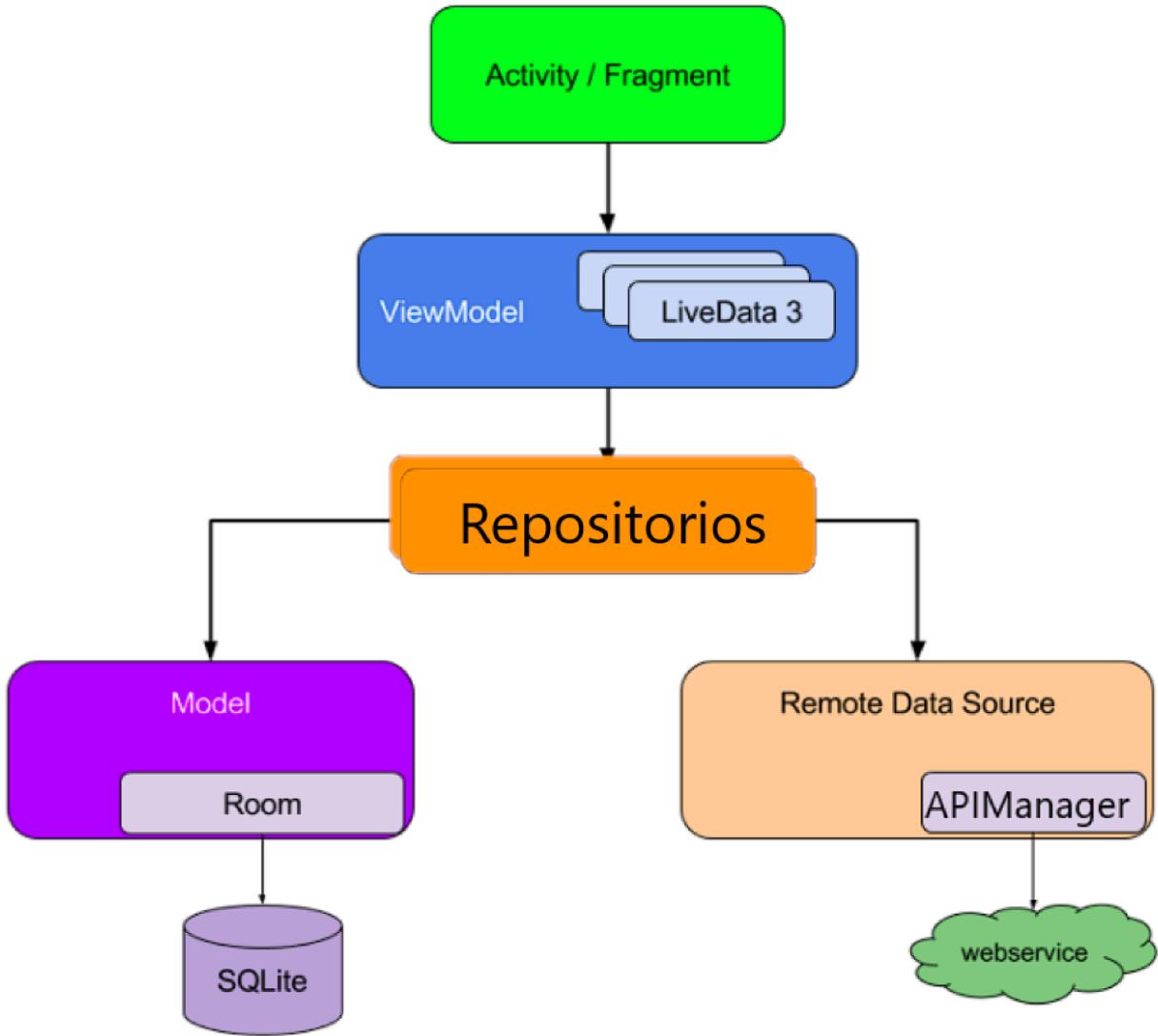
Ventajas de utilizar el patrón DAO:

- Es fácil de implementar
- Permite separar por completo la **lógica de acceso** a datos en una capa separada y así solo trabajar con la **lógica de negocio** sin preocuparnos de donde vienen los datos o los detalles técnicos para consultarlos o actualizarlos.
- Puede funcionar en conjunto con el **patrón Repository**.

## Refactorización

### Patrón Repository

Se ha optado por unificar todas las fuentes de datos creando una sola instancia que permita acceder a todos los datos.



Para ello, se han creado una serie de Repositorios asociados a los objetos del modelo de datos que son almacenados en la base de datos y accedidos desde la app.

La clase **EventRepository** permitirá acceder tanto a los datos procedentes de la api (APIManager) como a la información que se encuentra en la base de datos de Room (AppDatabase). La clase de repositorio aísla las fuentes de datos del resto de la app, una capa intermedia entre la capa de dominio y la capa de acceso de datos. Usar una clase de repositorio garantiza que este código sea independiente de la clase ViewModel y es una práctica recomendada para la separación del código y su arquitectura.

Por demás, un evento tiene asociado datos relacionados con el tiempo meteorológico proporcionados por la API. Si el usuario desea consultar los detalles de un evento consultado recientemente, **EventRepository** se encarga de cargar los datos de la Room (o caché). En el caso de consultar los detalles de un evento en un periodo de tiempo superior al umbral establecido, entonces se realiza una petición a la API OpenWeather y se actualiza la caché con los datos retornados de la API.

Así mismo, también se ha creado una clase **LocationRepository** la cual permitirá acceder a los datos de la API (APIManager) como fuente de datos externa y al modelo de Room (LocationDAO), en el que se manejan las localizaciones alojadas (municipios). De esta manera, el repositorio hará de capa intermedia entre la app (los ViewModels) y estas fuentes de datos.

Del mismo modo, se ha creado una clase **UserRepository** la cual permitirá acceder a los usuarios de Room.

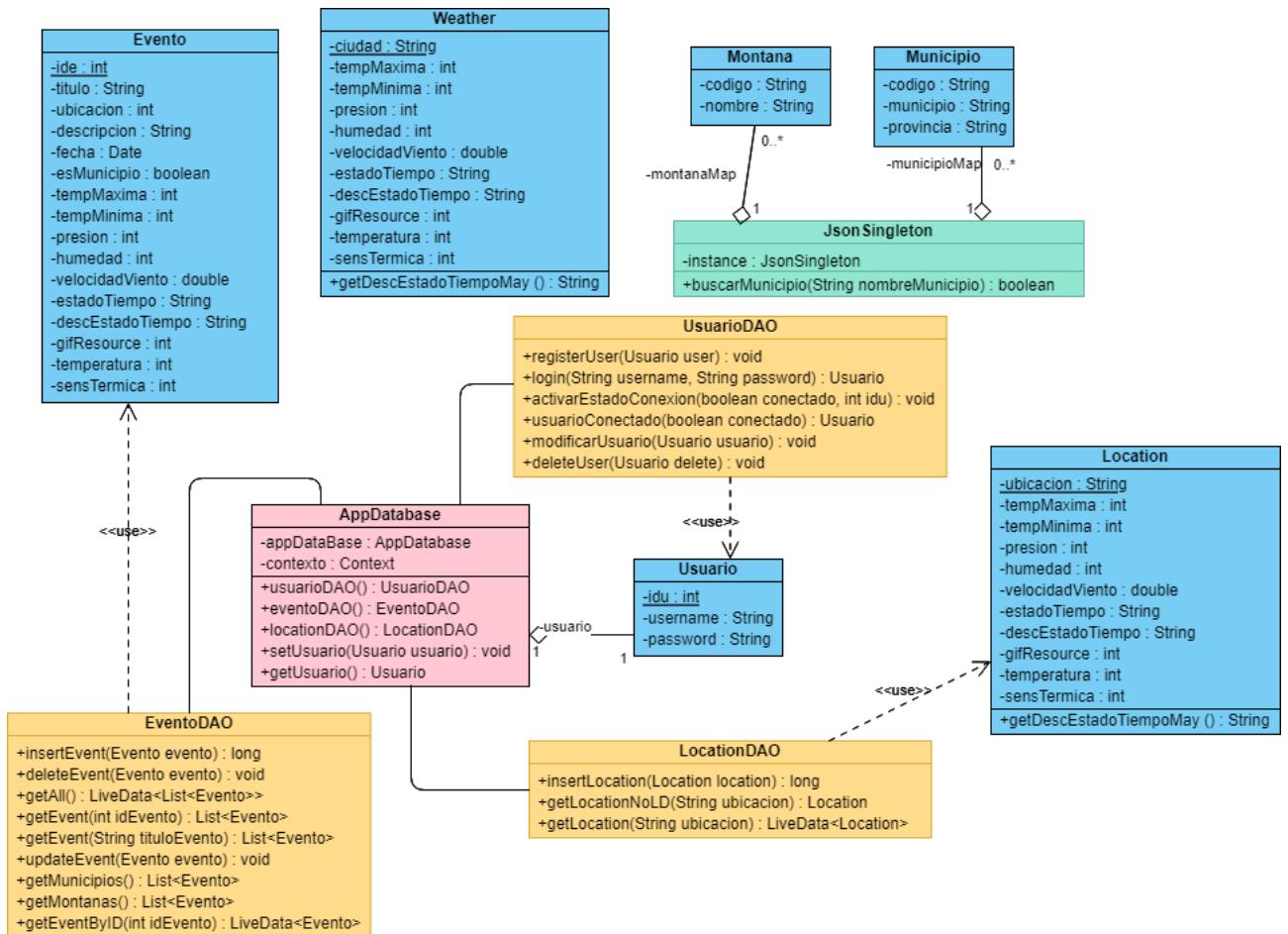
En el proceso de implementación de este patrón, se consideraron una serie de cambios relativos al modelo de datos de la aplicación propuesto en una versión temprana, con el objetivo de implementar la memoria caché.

- Para comprobar si se requiere una actualización del tiempo meteorológico de un evento, es conveniente almacenar la información del tiempo dentro del propio objeto **Evento**. Por ello, se ha modificado la entidad **Evento** incluyendo los atributos relativos al tiempo meteorológico, que anteriormente se encontraba en la clase **Weather**.

Por ende, al consultar los detalles de un evento, se comprueba si es necesario recuperar los datos asociados al tiempo de dicho evento desde la caché o actualizar la base de datos con los datos devueltos de la API.

- Creación de una clase **Location**, que almacena toda la información (tiempo) relativa a una ubicación concreta. Esta clase sustituye a la anterior **Weather**, que se introducirá dentro de la clase Evento.
- Creación de una interfaz dao **LocationDAO** relativa a la gestión de objetos Location en la base de datos.

Tras todas estas modificaciones, el modelo de datos resultante utilizado en la base de datos sería el siguiente:



## Ventajas de usar un repositorio

Un módulo de repositorio controla operaciones de datos y te permite usar varios backends. En una app real típica, el repositorio implementa la lógica para decidir si debe recuperar datos de una red o usar resultados almacenados en caché de una base de datos local.

Con un repositorio, puedes intercambiar los detalles de la implementación, como la migración a una biblioteca de persistencia diferente, sin afectar el código de llamada, como los modelos de vista. Esto también permite que tu código sea modular y se pueda probar. Puedes simular con facilidad el repositorio y probar el resto del código.

Un repositorio debe funcionar como una única fuente de verdad para una parte específica de los datos de tu app. Cuando se trabaja con varias fuentes de datos, como un recurso conectado en red y una caché sin conexión, el repositorio garantiza que los datos de la app sean lo más precisos y actualizados, lo que proporcionará la mejor experiencia posible incluso cuando la app esté sin conexión.

## Patrón Model - View - ViewModel (MVVM)

Con el objetivo de incrementar la seguridad de la aplicación y gestionar los datos de forma más rápida y eficiente se ha implementado el patrón **Model - View - ViewModel (MVVM)**,

Se han implementado los siguientes viewmodels:

- **ListaEventosViewModel:** Viewmodel que permite gestionar los datos de la lista de eventos del fragmento ListaEventosFragment.
- **DetallesEventoViewModel:** Viewmodel que gestiona los datos al mostrar los detalles de un evento en el fragmento DetallesEventoFragment.
- **TiempoActualViewModel:** Viewmodel que gestiona el tiempo actual mostrado en el fragmento de inicio InicioFragment.
- **DetallesLocalizacionViewModel:** Viewmodel que gestiona el tiempo de una ubicación concreta en la actividad DetalleLocalizacionActivity.
- **IniciarSesionViewModel:** Viewmodel que gestiona el inicio de sesión en la actividad InicioSesion.
- **RegistrarseViewModel:** Viewmodel que gestiona la pantalla de registro de usuario en la actividad Registrarse.
- **PerfilViewModel:** Viewmodel que gestiona la consulta y modificación del usuario en el fragmento PerfilFragment.
- **BorrarPerfilViewModel:** Viewmodel que permite eliminar el usuario en el fragmento DeleteDialogFragment.
- **ModificarEventoViewModel:** Viewmodel que gestiona los detalles relativos a la pantalla de modificar un evento del fragmento ModificarEventoFragment.
- **mainUsuarioViewModel:** Viewmodel que permite comprobar y cerrar la sesión del usuario en MainActivity.

Cabe mencionar que los ViewModels tienen una dependencia relacionada con un Repository específico. Para solventar la **escalabilidad** de código de nuestra aplicación respecto la adicción de nuevas dependencias a los ViewModel, se ha implementado el patrón de diseño **Factory** para crear instancias ViewModel de manera sencilla. Además, ante el crecimiento de dependencias de la app como los repositorios, AppDataBase y factories, entre otros; se han agrupado como atributos en una clase Singleton llamada **AppContainer**, la cual es instanciada por primera vez en una nueva clase llamada MyApplication que extiende Application. Para iniciar esta clase, se ha modificado el AndroidManifest incrustando un atributo android:name=".MyApplication".

El recurso de ViewModel destaca como el componente que se encargará de servir como puente entre la interacción de la Vista (View) y el Modelo (Model).

Entre sus ventajas encontramos:

- Su capacidad para **separar de forma limpia la presentación** de una aplicación determinada **y la lógica del negocio** de su interfaz de usuario. Lo que contribuye a abordar múltiples tipos de inconvenientes de desarrollo, prueba, mantenimiento y evolución del sistema.
- Permite que los desarrolladores creen **pruebas unitarias** para el Model View y el modelo, sin que sea necesario el uso de la vista.
- Los encargados del diseño y desarrollo de aplicaciones pueden ser capaces de **trabajar de manera simultánea e independiente**, cada uno en sus componentes durante los procesos de la app.

- ViewModel **permite la conservación** tanto en el **estado** que contiene un ViewModel como en las **operaciones** que esté activa. Este almacenamiento en caché significa que **no necesitas recuperar datos** mediante cambios de configuración comunes, como una rotación de pantalla.
- **SaveStateHandle** te permite conservar datos no solo a través de cambios de configuración, sino también durante la recreación de procesos. Es decir, te permite **mantener el estado de la IU intacto**, incluso cuando el usuario **cierra la app** y la **abre** más adelante.
- **Reducción de la complejidad:** al separar la lógica de presentación de la lógica de negocio en componentes distintos, el código de la aplicación se vuelve más fácil de entender y mantener.
- **Mayor reutilización de código:** debido a que la lógica de presentación y la lógica de negocio están separadas, se pueden reutilizar fácilmente en diferentes vistas y contextos.
- **Mejora del rendimiento:** el patrón MVVM permite que la vista se actualice automáticamente cuando los datos cambien en el modelo, lo que reduce la cantidad de código que se debe escribir y mejora el rendimiento de la aplicación.

## Aspectos novedosos

### Gestión de la API

En primer lugar se ha decidido no hacer uso de **Retrofit** para la gestión y configuración de la API.

En su lugar, esta funcionalidad se encuentra en una sola clase **APIManager** que seguirá un patrón de delegación mediante una interfaz **APIManagerDelegate** la cual se añade como atributo de aquel componente que quiera hacer uso de la API y así poder implementar los métodos de esta interfaz cuando las llamadas a la API devuelven un resultado.

De esta forma todo el networking de la aplicación se encuentra en la misma clase, cuando a los métodos que hacen referencia a las distintas llamadas a la API. La llamada a la API se realiza mediante una petición asíncrona haciendo uso de la **librería AsyncHttpClient** ahorrándonos así la creación y gestión de un hilo.

### Obtención de Localización

Uno de los casos de uso de la aplicación, requiere de obtener las coordenadas (longitud y latitud) de la ubicación del dispositivo. Para ello es necesario comprobar que se tienen los permisos de **GPS**, los cuales se comentan en el siguiente apartado. Una vez se tienen los permisos mediante un **LocationManager** se obtienen del proveedor de internet dichas coordenadas.

## Gestión de Permisos

Este apartado detalla la solicitud de los permisos necesarios para el correcto funcionamiento de la aplicación.

Los permisos (que no se hayan concedido) siempre se piden al iniciar la aplicación mediante la clase **Launch**. En caso de no conceder los de GPS, puesto que forman parte de una funcionalidad básica y esencial de la aplicación, no se podrá acceder a la misma hasta que no se otorguen.

Una vez concedidos los permisos, la aplicación comprueba en todo momento que estos sigan estando concedidos antes de ejecutar alguna operación que los requiera. En caso de no estar otorgados porque el usuario los haya quitado mientras está usando la aplicación, esta gestionaría correctamente la ausencia de permisos sin generar errores.

En el hipotético caso de necesitar nuevos permisos en el desarrollo de la aplicación, el código ha sido modularizado de forma que únicamente habría que añadir el nombre de dichos permisos al vector de permisos de la clase **Launch** (obviamente deberían estar presentes en el archivo manifest).

## Carga de municipios y montañas desde JSON

La lista de montañas necesaria para el spinner del caso de uso de Crear Evento de Montaña junto a sus coordenadas se encuentra almacenada en un archivo json. De igual forma para el caso de uso de filtrado de localizaciones se ha hecho uso de una lista de todos los municipios de España la cual también se encuentra almacenada en un archivo json.

Ambos archivos son cargados en un mapa de Java gestionado como una única instancia accesible desde cualquier lugar de la aplicación (Singleton) en la clase **JsonSingleton** al iniciar la aplicación por primera vez. Haciendo uso de la librería Gson obtenemos la información del json en los modelos **Municipio** y **Evento** que posteriormente se almacenarán en los mapas del singleton.

Esto nos permite que cada instancia instalada de nuestra aplicación **tenga los datos necesarios para la gestión de las APIs** utilizadas.

## Menú de Hamburguesa

Hemos decidido la utilización de un menú de hamburguesa como métodos de navegación entre las pantallas principales de nuestra aplicación.

Este menú nos permite que el usuario pueda navegar entre las cuatro pantallas principales de la aplicación sin que esto ocupe espacio de pantalla, este menú contiene las pantallas inicio, eventos, perfil y ajustes, dejando para la barra de navegación la búsqueda de ubicaciones y el botón de cerrar sesión.

Para su implementación se ha utilizado la main activity como contenedora del menú de hamburguesa y de los diferentes fragmentos de inicio, eventos, perfil y ajustes.

De esta manera hemos conseguido que de una forma compacta todo lo que tiene relación con la navegación de la app.

## Filtro de Eventos

Dentro de la pantalla de la lista de eventos hemos implementado un filtro que nos permite diferenciar los diferentes eventos que haya guardado el usuario, tanto por tipo de evento como teniendo en cuenta o bien el orden de creación del evento o la fecha de dicho evento.

Para los tipos de evento hemos decidido que la filtración se haga con un Tab, intercambiando entre los eventos de municipio y los eventos de montaña.

Para filtrar por orden de creación o de fecha de evento, hemos decidido crear un **spinner** que nos permite seleccionar entre las dos opciones.

Con esto conseguimos que se intercalan los dos tipos de filtrados que hacemos pudiendo obtener así cuatro configuraciones distintas:

1. Municipios ordenados por fecha de creación del evento
2. Municipios ordenados por fecha del evento
3. Montañas ordenadas por fecha de creación del evento
4. Montañas ordenadas por fecha del evento

De esta forma podemos darle todas las opciones que el usuario necesita para filtrar sus eventos de una manera cómoda y sencilla.

## Usuario único

En esta aplicación hemos implementado el usuario de tal forma que solamente existe un único usuario, esto se ha hecho así debido a que el usuario se guarda de manera local y no en remoto, y también a que no hemos concebido la aplicación para qué se guarden diferentes cuentas dentro de nuestra aplicación, sino como una lista local de eventos.

## Implementación de Spinners en diversos campos

Se han implementado campos con el tipo de Spinners en los **campos de Montaña** de las pantallas destinadas a crear un evento de Montaña (layout del fragmento CrearEventoMontana) y modificar un evento de montaña (layout del fragmento ModificarEventoMontana) con el objetivo de seleccionar una montaña existente del conjunto obtenido del JSON para crear o modificar un evento.

De esta forma, se asegura que tras seleccionar el nombre de la montaña en la creación o modificación de eventos, esta no sea errónea, pues se ha tenido que elegir un nombre existente entre las montañas cargadas, ahorrando al usuario el hecho de probar hasta dar con el nombre exacto de la montaña.

Así mismo, también se ha implementado un campo de Spinner para especificar el **tipo de Ordenación** que se aplicará al filtrar todos los eventos, en la sección **Eventos**. Dicho campo con el tipo de ordenación permitirá organizar los eventos por orden de creación o por fecha.

## Filtrado localizaciones

Una vez iniciada sesión, desde la pantalla principal se puede acceder a la funcionalidad de filtrado de localizaciones si se pulsa el ícono de lupa de la ToolBar.

Se inicia una nueva actividad que se encarga de leer el listado de municipios de España desde la colección de tipo mapa instanciada en el SingletonJSON y se utiliza un Adapter como intermediario entre la obtención de los municipios con el layout.

En el layout, se define un nuevo elemento SearchView, parecido a un input que permite actualizar en tiempo real la lista de municipios, de acuerdo a la cadena introducida por el usuario.

En el código de la actividad LocalizacionesActivity, se define un listener para el SearchView, sobreescribiendo un método llamado onQueryTextChange(String text). Dentro, se llama a una función filtradora llamada filter(text) y se le pasa la cadena del SearchView introducida por el usuario. En la función de filtrado, se inserta en una variable auxiliar (ArrayList) aquellos municipios que coinciden con la cadena introducida, mediante la función findWithPrefix(municipios, cadena).

Por último, al disponer de la nueva colección de municipios filtrados, queda actualizar la lista de items del Adapter y notificar que se ha modificado para actualizar la interfaz con los nuevos municipios coincidentes.

Esto **permite** al usuario buscar una localidad sin necesidad de que este sepa su nombre exacto, el cual es necesario tanto para crear un evento como para ver el tiempo actual en una localidad.

## Modo Oscuro

El modo oscuro es un ajuste de configuración que se encuentra en la sección “Ajustes” del menú hamburguesa. Se ha implementado con las Preferencias que nos ofrece la API de Android. Por defecto, el tema de la aplicación es el modo oscuro.

Cuando se activa el checkbox, desde el fragmento “AjustesFragment” se modifica el valor relativo al nuevo tema de la aplicación en las SharedPreferences , gracias a su editor. A continuación, se realiza un callback a la actividad “MainActivity” que soporta el fragmento. La función del callback se encarga de leer el archivo SharedPreferences modificado y delega el cambio de tema de la aplicación al AppCompat, recreando de nuevo la actividad y sus fragmentos involucrados para que la interfaz se actualice en base al nuevo tema.

Por otro lado, se incluyen dos archivos en la carpeta de recursos de la aplicación llamados theme (uno con la configuración de la interfaz en modo claro y el otro en modo oscuro). En estos archivos theme, se definen unos estilos que serán intercambiados según la configuración de la aplicación.

En el resto de componentes de la aplicación, se realiza el mismo proceso. En el caso de las actividades sin fragmentos, en el `onResume()` se llama al método que se encarga de leer SharedPreferences y delegar el cambio de tema.

Entre las **ventajas** que existen a la hora de que nuestra aplicación tenga modo oscuro destacan las siguientes:

- Es saludable para los ojos: Ahorrándole al usuario molestias visuales.
- Se visualiza mejor en la oscuridad.
- Prolonga la vida de la batería: Utilizando menos brillo, o incluso con el apagado de los píxeles de la mayor parte de la pantalla en los paneles OLED.
- Mejora la accesibilidad: Muchas personas con diferentes problemas de vista pueden encontrar en esta opción una forma de visualizar mejor el contenido.

## Gestión de la calidad del software

En este apartado se describen las pruebas realizadas a la aplicación Android, así como el análisis del código para solventar problemas.

### Pruebas

En primer lugar, se implementan los test **unitarios**, los cuales son más simples de implementar y suelen identificar un mayor número de bugs o problemas. Estos presentan una serie de características:

- Deben ser muy pequeños
- Deben estar centralizados en funcionalidades muy específicas.
- Se recomienda la realización de test locales
- Normalmente, no es necesario realizar test instrumentalizados.

Existen multitud de frameworks para implementar test unitarios como JUNIT, Cactus, Mockito, etc. Se ha elegido JUnit, un framework para escribir test unitarios repetibles que pueden ser fácilmente integrados en cualquier proyecto Java. Es uno de los frameworks más importantes para el desarrollo del concepto Test-Driven Development.

Se utiliza la librería JUnit para implementar los test unitarios asociados a las tareas “Test Components” de cada caso de uso. Para ello, en el build.gradle se incluyen las siguientes librerías:

```
testImplementation 'junit:junit:4.12'  
testImplementation 'androidx.test:core:1.4.0'  
androidTestImplementation 'androidx.test:core:1.4.0'  
  
androidTestImplementation 'androidx.test:runner:1.4.0'  
androidTestImplementation 'androidx.test:rules:1.4.0'  
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
```

Los test unitarios se almacenan en un paquete de pruebas llamado “test”. Los test son clases java con anotaciones de JUnit, que le indican al “Runner” por defecto del framework Android cómo se deben ejecutar las clases test.

Respecto a los nombres de los test unitarios, estos deben describir el funcionamiento/objetivo del test, qué caso de uso se está testeando y deben incluir el sufijo <\_UnitTest>.

Los test **funcionales** conllevan un mayor tiempo de implementación y de ejecución, aunque también proporcionan una mayor fiabilidad.

- Normalmente, evalúan las funcionalidades de la aplicación.
- Tests de secciones verticales de la aplicación. Estos test evalúan diferentes interacciones entre las capas.
- No es recomendado el uso de test locales.
- Se recomiendan tests instrumentalizados.

Probar las interacciones de usuario dentro de una sola app ayuda a garantizar que los usuarios no tengan resultados inesperados ni una mala experiencia cuando interactúen con la app. Para realizar los test de interfaz o funcionales, se utiliza la librería **Espresso**.

Espresso está dirigido a desarrolladores, que creen que las pruebas automatizadas son una parte integral del ciclo de vida del desarrollo. Si bien Espresso se puede usar para pruebas de caja negra, quienes están familiarizados con la base de código bajo en modo de prueba pueden aprovecharlo al máximo.

En el build.gradle, se añade las dependencias necesarias de la librería:

```
androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'  
androidTestImplementation 'androidx.test:runner:1.3.0'  
androidTestImplementation 'androidx.test:rules:1.3.0'
```

Además, como Espresso es un test instrumentado, hay que indicar en el build.gradle la clase que nos ayuda con la instrumentalización: AndroidJUnitRunner.

```
testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
```

La grabadora de pruebas **Espresso** te permite crear pruebas de IU para tu app sin la necesidad de escribir código de prueba. Es imprescindible deshabilitar las animaciones de la app, para evitar fallos en la grabadora. Para ello, se accede a *Configuración - Opciones para desarrolladores* y desactiva todas las siguientes opciones:

- Escala de animación de ventana
- Escala de animación de transición
- Escala de duración de animador

## Pruebas implementadas

En primera instancia, cada líder de los equipos de desarrollo han creado las ramas correspondientes a los casos de uso asignados a cada equipo. Estas nuevas ramas tienen como nombre TEST - CU<Número> y han sido creadas a partir de la rama “origin/develop” de la práctica de implementación.

A continuación, se engloba en una tabla la planificación por roles de las tareas de testeo:

Encargado	CU01	CU02	CU03	CU04	CU05	CU06	CU07	CU08	CU09	CU10	CU1 1	CU1 2	CU 13	CU14	CU15	CU16
Sr Blanco						Test Evl.			Test Evl.	Test Evl.				Test Evl.	Test Evl.	Test Evl.
						AAM			AAM	AAM				AAM	AAM	AAM
Sr Marrón											Test Evl.	Test Evl.				
										AAM	AA M	AA M	AA M	AAM		
Sr Naranja	Test Evl.	Test Evl.	Test Evl.										Te st Evl.			
	AAM	AAM	AAM											A A M		
Sr Azul				Test Evl.	Test Evl.		Test Evl.	Test Evl.								
				AAM	AAM	AAM	AAM	AAM								

Cabe destacar que en esta tabla la tarea **Test Evl.** corresponde a la de la disciplina de test “Test and Evaluate”, mientras que la tarea denominada **AAM** corresponde a “Achieve Acceptable Mission” and Improve Test Assets”.

Una vez creadas las ramas de caso de uso para el testeo, cada líder liderará acerca de las tareas que deben realizar cada uno de los roles, así como las clases y componentes que deben ser evaluados para cada caso de uso.

## Equipo 1 (Señor Blanco)

### CU1 - Añadir Evento de Municipio

#### Test unitario

Se crea un paquete en los “test” llamado **AñadirEventoMunicipio**. En él, se crean dos clases test unitarias: **EventoUnitTest** y **MunicipioUnitTest**, correspondientes a las clases del modelo de datos **Municipio** y **Evento**.

Respecto el modelo **Municipio**, se testean los métodos getter/setters. De la clase **Evento**, se testean los métodos getter/setters y el método `compareTo()`.

#### Test funcional

Se crea una clase **CrearEventoMunicipioTest** en el paquete `AndroidTest` para testear el funcionamiento de la Interfaz de usuario al crear un evento de municipio en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra con una credenciales e inicia sesión en la aplicación. En la pantalla principal, se pulsa el floating button para crear un nuevo evento. Aparece una pantalla de elección del tipo de evento a crear, se elige el de tipo de evento llamado Municipio.

Esto mostrará la pantalla de creación, donde se piden todos los datos del evento, en este caso se introducirá como nombre del evento “Futbol” y como localidad “Sevilla”.

Se rellenan todos los campos del formulario asociado al nuevo evento y se confirma la creación, apareciendo una pantalla de detalles. En este instante, se aplican **asserts** para enriquecer el test.

A continuación, se presiona el botón “back” para comprobar que se ha incluido el nuevo evento en la lista de eventos, mediante un nuevo **assert**. Para que el test no dependa de otros, es

necesario devolver la aplicación al estado de ejecución inicial (eliminando el evento creado y la cuenta de usuario).

## CU2 - Añadir Evento de Montaña

### Test unitario

Se crea un paquete en los “test” llamado **AñadirEventoMontana**, el cual requiere dos test unitarios: Montana y Evento. Sin embargo, el segundo se incorpora en el CU01. Por tanto, se crea una clase test unitaria: **MontanaUnitTest**, correspondientes a las clases del modelo de datos **Montana**.

Respecto el modelo **Montana**, se testean los métodos getter/setters.

### Test funcional

Se crea una clase **CrearEventoMontanaTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al crear un evento de montaña en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra con una credenciales e inicia sesión en la aplicación. En la pantalla principal, se pulsa el floating button para crear un nuevo evento. Aparece una pantalla de elección del tipo de evento a crear, se elige el de tipo de evento llamado Montaña.

Esto mostrará la pantalla de creación, donde se piden todos los datos del evento, en este caso se introducirá como nombre del evento “Senderismo” y cómo localidad “Sierra nevada”.

Se rellenan todos los campos del formulario asociado al nuevo evento y se confirma la creación, apareciendo una pantalla de detalles. En este instante, se aplican **asserts** para enriquecer el test.

A continuación, se presiona el botón “back” para comprobar que se ha incluido el nuevo evento en la lista de eventos, mediante un nuevo **assert**. Para que el test no dependa de otros, es necesario devolver la aplicación al estado de ejecución inicial (eliminando el evento creado y la cuenta de usuario).

## CU3 - Añadir Usuario

### Test unitario

Se crea un paquete en los “test” llamado **AñadirUsuario**. En él, se crea una clase test unitaria: **UsuarioUnitTest**, correspondiente a las clases del modelo de datos **Usuario**.

Respecto el modelo **Usuario**, se testean los métodos getter/setters. De la clase **Usuario**, se testean los métodos getter/setters y el método `compareTo()`.

#### Test funcional

Se crea una clase **AnadirUsuarioTest** en el paquete `AndroidTest` para testear el funcionamiento de la Interfaz de usuario al añadir una cuenta de usuario en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra un nuevo usuario introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Jorge” y como contraseña “1234”, se incorporan además **asserts** para comprobar que las credenciales son las esperadas.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla principal, donde es necesario devolver la aplicación al estado de ejecución inicial (eliminando la cuenta de usuario).

## CU4 - Añadir barra de búsqueda y filtrado de ubicaciones

#### Test unitario

Los test unitarios de este caso de uso son **Municipio** (lo tiene implementado el test CU01) y **Montaña** (lo tiene implementado el test CU02).

#### Test funcional

Se crea una clase **AnadirBarraBusquedaTest** en el paquete `AndroidTest` para testear el funcionamiento de la Interfaz de usuario al realizar una búsqueda de una localización en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, primero se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Luis” y como contraseña “123456”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla principal.

Una vez se ha iniciado sesión, se accede a la barra de búsqueda haciendo clic en el icono de la lupa colocado en la parte superior de barra de navegación (appbar) y se coloca una cadena de texto para comprobar que existen ubicaciones con el nombre de la cadena introducida.

En este caso, se escribirá la cadena “Sevilla”, lo que deberá mostrar una lista de ubicaciones en pantalla con al menos una de ellas llamada Sevilla, esto se comprobará con un **assert** que se realizará sobre un item encontrado del recyclerview que deberá contener esta cadena.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## Equipo 2 (Señor Marrón)

### CU5 - Añadir preferencias desde el menú AppBar

#### Test unitario

En este caso de uso no se requiere ningún test unitario.

#### Test funcional

Se crea una clase **MenuTest** en el paquete AndroidTest para testear el funcionamiento del panel lateral de navegación de la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, primero se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “pepe” y como contraseña “pepe”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla principal.

Tras llegar a la pantalla principal, se procede a navegar a través de todas las pantallas que se encuentran en el menú lateral, recorriendo las pantallas de Lista de filtrado de eventos, perfil, ajustes y finalmente de nuevo Inicio.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU6 - Consultar tiempo detallado de una ubicación

### Test unitario

Se crea un paquete en los “test” llamado **ConsultarTiempoDetalladoUbicacion**. En él, se crea una clase test unitaria: **WeatherUnitTest**, en referencia a la clase del modelo **Weather**, sobre la cual se testean los métodos getter/setters.

### Test funcional

Se crea una clase **DetallesLocalizacionTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al consultar los detalles (tiempo) de una localización en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, primero se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Luis” y como contraseña “123456”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla principal.

Tras iniciar sesión, se accede a la barra de búsqueda haciendo clic en el ícono de la lupa colocado en la parte superior de barra de navegación (appbar) y se coloca una cadena de texto para acceder a una ubicación con el nombre de la cadena introducida.

En este caso, se escribirá la cadena “Sevilla”, lo que muestra una lista de ubicaciones que contienen este nombre, y se hace clic en la ubicación que contiene el nombre igual a la cadena “Sevilla”, lo que llevará a la pantalla de detalles de la localización.

Esto deberá mostrar una pantalla con la información de la localidad “Sevilla”, comprobando mediante **assert** que existen cadenas en la pantalla relativas a esta ubicación dado que no son igual a las cadenas determinadas por defecto.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU7 - Modificar un evento

### Test unitario

El test unitario de este caso de uso es el del modelo **Evento**, el cual ha sido implementado en el CU01.

### Test funcional

Se crea una clase **ModificarEventoTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al modificar un evento en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra un nuevo usuario introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “a” y como contraseña “a”, se incorporan además **asserts** para comprobar que las credenciales son las esperadas.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla principal. En ella, se crea un nuevo evento de municipio pulsando en el floating button situado en la parte inferior derecha. Se rellenan los campos del formulario y se confirma la creación. En la siguiente pantalla de detalles, se pulsa en el botón “Modificar” y se modifican los datos del evento. Se cambia el título “padel” por “padelcomida” y se localiza el evento en Madrid. Se incorporan **asserts** para enriquecer el test y comprobar que los campos se han modificado correctamente. Con el objetivo de que otros test no dependan de éste, se elimina el nuevo evento modificado.

Se sigue el mismo proceso explicado anteriormente para un evento de tipo montaña, incorporando **asserts** correspondientes a los valores del evento modificado. Del mismo modo, al modificar el evento, se elimina de la aplicación.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU8 - Eliminar un evento

### Test unitario

El test unitario de este caso de uso es el del modelo **Evento**, el cual ha sido implementado en el CU01.

## Test funcional

Se crea una clase **EliminarEventoTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al eliminar un evento en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra un nuevo usuario introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Jorge” y como contraseña “1234”, se incorporan además **asserts** para comprobar que las credenciales son las esperadas.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla de inicio.

Tras llegar a la pantalla principal, se pulsa el floating button para crear un nuevo evento. Aparece una pantalla de elección del tipo de evento a crear, se elige el de tipo de evento llamado Municipio.

Esto mostrará la pantalla de creación, donde se piden todos los datos del evento, en este caso se introducirá como nombre del evento “Futbol”, como localidad “Sevilla” y como descripción “Con amigos”.

Se rellenan todos los campos del formulario asociado al nuevo evento y se confirma la creación, apareciendo una pantalla de detalles. En este instante, se aplican **asserts** para enriquecer el test.

A continuación, se presiona el botón “back” para comprobar que se ha incluido el nuevo evento en la lista de eventos, mediante un nuevo **assert**.

Posteriormente, se vuelve a acceder a la pantalla de detalles del evento creado haciendo clic en el ítem del evento del recycler View.

Esto llevará a la pantalla para consultar este, donde se pulsará el botón para eliminar el evento, comprobando que tras esto lleva a la lista de eventos y que dicha lista está vacía mediante un **assert**.

Posteriormente, se pulsa el floating button para crear un nuevo evento. Aparece una pantalla de elección del tipo de evento a crear, se elige el de tipo de evento llamado Montaña.

Esto mostrará la pantalla de creación, donde se piden todos los datos del evento, en este caso se introducirá como nombre del evento “Senderismo” y como localidad “Sierra nevada”.

Se rellenan todos los campos del formulario asociado al nuevo evento y se confirma la creación, apareciendo una pantalla de detalles. En este instante, se aplican **asserts** para enriquecer el test.

A continuación, se presiona el botón “back” para comprobar que se ha incluido el nuevo evento en la lista de eventos, mediante un nuevo **assert**.

Más tarde, se vuelve a acceder a la pantalla de detalles del evento creado haciendo clic en el ítem del evento del recycler View.

Esto llevará a la pantalla para consultar este, donde se pulsará el botón para eliminar el evento, comprobando que tras esto lleva a la lista de eventos y que dicha lista está vacía mediante un **assert**.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## Equipo 3 (Señor Naranja)

### CU9 - Consultar tiempo meteorológico en la ubicación actual

#### Test unitario

El test unitario de este caso de uso es el del modelo **Weather**, el cual ha sido implementado en el CU06.

#### Test funcional

Se crea una clase **TiempoActualTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al mostrar el tiempo de la localización actual en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, primero se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Luis” y como contraseña “123456”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla de inicio.

Al llegar a la pantalla principal, se comprueba que el tiempo actual se muestra correctamente en esta (en la parte superior de la pantalla) comprobando mediante **asserts** que los valores que se encuentran en el cuadro del tiempo actual son distintos de los valores por defecto.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU10 - Modificar idioma y tema a modo oscuro

### Test unitario

En este caso de uso no se ha implementado ningún test unitario.

### Test funcional

Se crea una clase **ModoOscuroTest** en el paquete AndroidTest para testear el funcionamiento del modo oscuro o claro en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, primero se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “pepe” y como contraseña “pepe”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla de inicio.

Tras iniciar sesión, se accede a la pestaña de ajustes a través del panel lateral de navegación para cambiar el tema modo claro, navegando posteriormente a la pestaña de Perfil del panel lateral de navegación, regresando después a la pestaña de ajustes para cambiar de nuevo el tema a modo oscuro.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y tras comprobar mediante un **assert** que el color del nombre de usuario es el del valor del azul oscuro (R.color.Azul\_osc), selecciona la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU11 - Consultar lista de eventos

### Test unitario

El test unitario de este caso de uso es el del modelo **Evento**, el cual ha sido implementado en el CU01.

### Test funcional

Se crea una clase **ListarEventosTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al realizar una búsqueda de una localización en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, en primer lugar se han creado 4 eventos accediendo al eventoDAO de la base de datos, siendo los nombres de estos “cena1”, “cena2”, “cena3” y “cena4”, con “Madrid” como localizaciones y con “Cañas” como descripciones, insertando estos eventos en la base de datos.

Posteriormente se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Miguel” y como contraseña “miguel”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla de inicio.

Tras llegar a la pantalla principal, se comprueba mediante **asserts** que todos los eventos creados se muestran en la pantalla de inicio, siendo estos 4 eventos de municipio.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU12 - Consultar un evento

### Test unitario

El test unitario de este caso de uso es el del modelo **Evento**, el cual ha sido implementado en el CU01.

### Test funcional

Se crea una clase **ConsultarEventoTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al consultar los detalles de un evento en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, primero se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Jorge” y como contraseña “1234”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla de inicio.

Se crea un nuevo evento en la aplicación de tipo Municipio, con el título “Futbol” y localizado en Sevilla, siendo la descripción del evento “con amigos”. Al confirmar la creación, se realizan **asserts** para comprobar que los campos que se han rellenado en el test son correctos. En última instancia de este evento, se elimina para limpiar la aplicación.

A continuación, se realiza el mismo proceso para los eventos de tipo Montaña. Se rellena el formulario asociado al nuevo evento y se comenten **asserts**, para enriquecer el test. En última instancia de este evento, se elimina para limpiar la aplicación.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## Equipo 4 (Señor Azul)

CU13 - Iniciar sesión

Test unitario

El test unitario de este caso de uso es el del modelo **Usuario**, el cual ha sido implementado en el CU03.

Test funcional

Se crea una clase **IniciarSesionTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al iniciar sesión en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Para llevar a cabo este test, primero se ha registrado un usuario en la aplicación, introduciendo en la pantalla de registro los datos relativos al nuevo usuario (nombre de usuario y contraseña) y confirmando la operación. En este caso, se ha introducido como nombre de usuario “Jorge” y como contraseña “1234”.

Una vez se ha registrado el usuario, se introducen sus credenciales en la pantalla de inicio sesión para acceder a las funcionalidades de la aplicación confirmando el inicio haciendo clic en el botón de “Iniciar sesión”, lo que llevará a la pantalla de inicio.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU14 - Cerrar sesión

### Test unitario

El test unitario de este caso de uso es el del modelo **Usuario**, el cual ha sido implementado en el CU03.

### Test funcional

Se crea una clase **CerrarSesionTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al cerrar la sesión actual en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra un nuevo usuario con una credenciales, y se inicia sesión en la aplicación. En la pantalla principal, en el AppBar (barra de opciones situada en la parte superior de la pantalla) se pulsa en el ícono “Cerrar Sesión” situado en la esquina superior derecha, cerrándose la sesión y retornando en la pantalla de inicio de sesión.

Para que el test no dependa de otros, finalmente se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU15 - Modificar usuario

### Test unitario

El test unitario de este caso de uso es el del modelo **Usuario**, el cual ha sido implementado en el CU03.

### Test funcional

Se crea una clase **ModificarUsuarioTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al modificar el usuario en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra un nuevo usuario con una credenciales, y se inicia sesión en la aplicación.

En la pantalla principal, se navega por el menú de hamburguesa a la opción “Perfil”. Dentro de ella, aparece un formulario con los campos asociados a las credenciales de la cuenta de usuario con la sesión iniciada. Por defecto, el campo asociado al “username” queda cargado por el valor actual que tenga la cuenta y se modifica por el usuario “juan”. A continuación, se añade la contraseña actual “pepe” y se incorpora una nueva contraseña llamada “juan”. Se pulsa en el botón “Modificar” y aparece la pantalla principal. Para comprobar que la modificación se ha realizado correctamente, se vuelve a los ajustes de perfil en el menú hamburguesa y se incorpora un **assert** para comprobar que el usuario se ha modificado correctamente.

Finalmente, tras comprobar esto, se procede a borrar el usuario accediendo a la pantalla de “Perfil” del propio usuario y seleccionar la funcionalidad de eliminar el usuario registrado haciendo clic en el botón, lo que borrará el usuario de la base de datos y cerrará sesión.

## CU16 - Eliminar usuario

### Test unitario

El test unitario de este caso de uso es el del modelo **Usuario**, el cual ha sido implementado en el CU03.

### Test funcional

Se crea una clase **EliminarUsuarioTest** en el paquete AndroidTest para testear el funcionamiento de la Interfaz de usuario al eliminar el usuario en la aplicación. Este método se ha creado a partir de la grabadora de Espresso, incorporada en Android Studio.

Este test consiste en lo siguiente: primero se registra un nuevo usuario con una credenciales, y se inicia sesión en la aplicación. En la pantalla principal, se navega por el menú de hamburguesa a la opción “Perfil”. Dentro de ella, se pulsa el botón “Eliminar” para eliminar la cuenta de usuario.

Para comprobar que la eliminación se ha realizado correctamente, se busca iniciar sesión con las mismas credenciales del usuario borrado, sin tener ningún efecto, comprobando así que el caso de uso funciona correctamente.

## Arreglo de errores

A partir de la implementación y observación de los test, se han cometido los siguientes arreglos:

- Se ha incluido un constructor por defecto, no parametrizado en la clase Evento, Municipio, Usuario y Montana.

- Al eliminar una cuenta de usuario, se ha incorporado el borrado de todos sus eventos asociados.
- Se ha modificado en el layout fragment\_crear\_evento\_montana.xml un atributo del EditText de la fecha llamado focusable. Incorporaba un fallo puesto que al querer elegir la fecha de un nuevo evento, se tenía que pulsar dos veces en el EditText. Por tanto, se le ha asociado el valor “false” al focusable.
- Se ha arreglado un error del CU Modificar Evento, específicamente en la clase ModificarEventoMunicipioFragment. Se actualiza la vista en un Thread secundario, cuando sólo se puede modificar en el hilo principal. Por tanto, se ha incorporado un requireActivity().runOnUiThread() para solventar el problema.
- Se ha modificado el acceso público a los atributos de la clase Weather, restringiendo el acceso a privado. Este arreglo ha supuesto unas pequeñas modificaciones en 4 componentes software (fragmentos y actividades, entre ellos), debido a la incorporación de los getter/setters en la clase Weather.

## Análisis de la calidad del código

### LUIS

En este apartado se ha realizado un proceso de análisis y gestión de calidad del proyecto que se ha realizado a lo largo de todas estas entregas, de manera que se compruebe si la calidad de este proyecto es aceptable, así como corregir el proyecto para mejorarla en caso contrario.

Para ello se utilizará la herramienta **SonarQube**, que permite detectar e identificar posibles fallos para mejorar la calidad de un proyecto, así como si esta es aceptable o no, o en su defecto **SonarCloud**, que consiste en su versión online.

En concreto, se debe utilizar esa herramienta para realizar un análisis de la calidad de la versión actual del proyecto e identificar los posibles fallos que pueda presentar esta, tratando de mejorar la calidad corrigiendo 2 de los fallos indicados por cada uno de los integrantes del grupo.

Posteriormente, se publicarán las versiones con fallos corregidos en nuevas ramas (una por cada incidencia) y se integrarán estas, de manera que se compila de nuevo el repositorio y comprobar que los los fallos se han solucionado.

De esta forma, se demuestra que la herramienta permite realizar correctamente un análisis de calidad del proyecto para identificar correctamente los fallos de calidad y corregirlos.

En este caso, se emplea la herramienta de SonarCloud, utilizándose de forma online a través de las Github Actions.

## Características de SonarCloud

SonarCloud es una plataforma en línea creada por SonarSource que ofrece análisis de código y herramientas de colaboración para desarrolladores de software.

Ayuda a los equipos de desarrollo a mejorar la calidad del código y a identificar problemas de seguridad y de cumplimiento normativo, entre otras cosas. También ofrece integraciones con diferentes herramientas de desarrollo y sistemas de control de versiones para facilitar su uso en el flujo de trabajo de un equipo.

## Repositorios utilizados

Para llevar a cabo el análisis de calidad, se ha utilizado el repositorio con el código del proyecto (ubicado en Github) cuyo enlace es el siguiente:

<https://github.com/UniExtremadura/proyecto-gps-asee-2022-23-ga04>.

Este se ha añadido al repositorio de SonarCloud con el nombre de “app”, que se puede encontrar en el siguiente enlace:

[https://sonarcloud.io/project/overview?id=UniExtremadura\\_proyecto-gps-asee-2022-23-ga04](https://sonarcloud.io/project/overview?id=UniExtremadura_proyecto-gps-asee-2022-23-ga04)

## Pasos a seguir

Para realizar esta práctica, se seguirán los siguientes pasos:

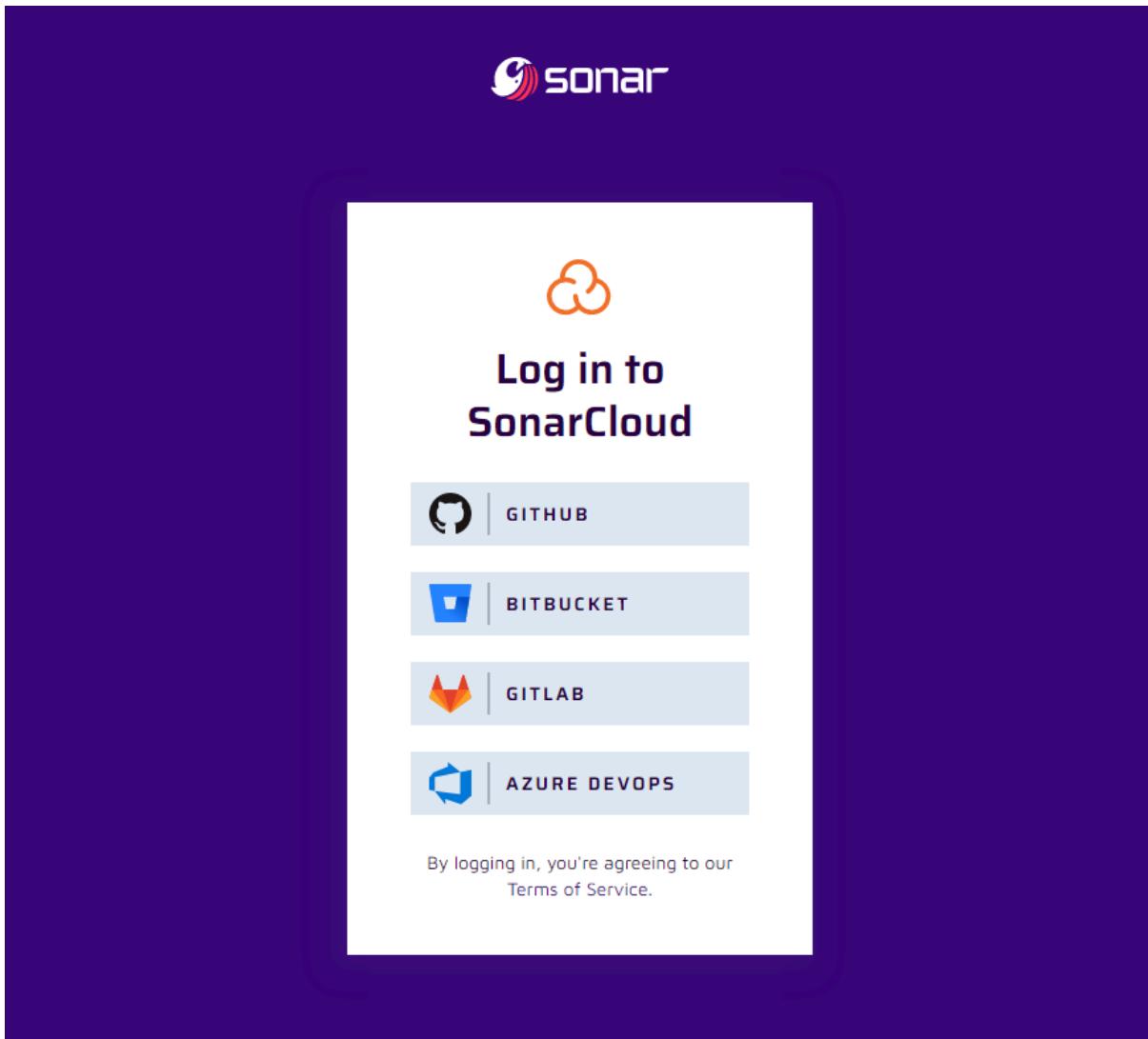
1. Compilar la rama actual sobre la que se encuentra la última versión del proyecto desde github (utilizando github Actions), para analizar la calidad del proyecto y comprobar si tiene errores.
2. Tras el análisis, seleccionar 2 puntos de mejora de calidad indicados por SonarCloud para cada integrante del grupo y tratar de mejorarlos, implementando dichas mejoras en una rama por cada corrección.
3. Integrar las ramas correspondientes a develop, y a su vez develop en la rama main.
4. Tras aplicar las correcciones implementadas al proyecto, comprobar si estas han solucionado los fallos de calidad que presentaba este compilado de nuevo la rama del proyecto utilizando SonarCloud.

<https://imgur.com/LViIRvI.png>

De esta forma, se espera que las correcciones implementadas solucionen los fallos detectados por SonarCloud y que el nuevo análisis realizado refleja esto.

## Detección de fallos

En esta primera fase se llevará a cabo un análisis inicial de la calidad del proyecto, realizando un proceso de compilación del proyecto en la página oficial de SonarCloud (<https://sonarcloud.io>).



Para ello, en primer lugar es necesario importar el proyecto dentro de SonarCloud, sincronizando este con la cuenta correspondiente de GitHub. Cabe destacar que para que se pueda importar el proyecto es necesario que el repositorio sea público y ser el administrador del mismo.

## Problemas solventados

Cada miembro del equipo se ha encargado de resolver dos incidencias en base al análisis de calidad de código completado mediante las Github Actions, las cuales utilizan gradle y java para enviar los resultados del análisis a SonarCloud.

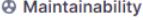
## Análisis del Sr. Blanco

El análisis base sobre el que se parte para resolver las dos incidencias del **Sr. Blanco** es el siguiente:

5.7k Lines of Code ? Version unspecified Last analysis 4 minutes ago 3ed9f218 Merge remote-tracking branch 'origin/develop' into main

Quality Gate ? The Quality Gate helps you see if your New Code is deployable or not. Ask your admin to [set a New Code definition](#) to get one.

**Not computed**

 Reliability 2 Bugs ? C	 Maintainability 580 Code Smells ? A
 Security 0 Vulnerabilities ? A	 Security Review 21 Security Hotspots ? 0.0% Reviewed E
Coverage 0.0% Coverage ? O	Duplications 1.3% Duplications ? O

Se puede observar cómo existen 580 Code Smells o incidencias de mantenibilidad. Las dos incidencias resueltas por el Sr. Blanco son las siguientes:

 Remove this unused import 'com.example.proyecto.Room.Modelo.Evento'. <a href="#">Why is this an issue?</a> 25 days ago L3 No tags
 Remove this unused import 'android.Manifest'. <a href="#">Why is this an issue?</a> 25 days ago L6 No tags

Una vez incorporados los dos commits de las dos incidencias a la rama develop y hecha la integración a la rama main, se lanzan las github actions para enviar el análisis a SonarCloud con las dos incidencias resueltas. El análisis resultante al resolver las dos incidencias son las siguientes:

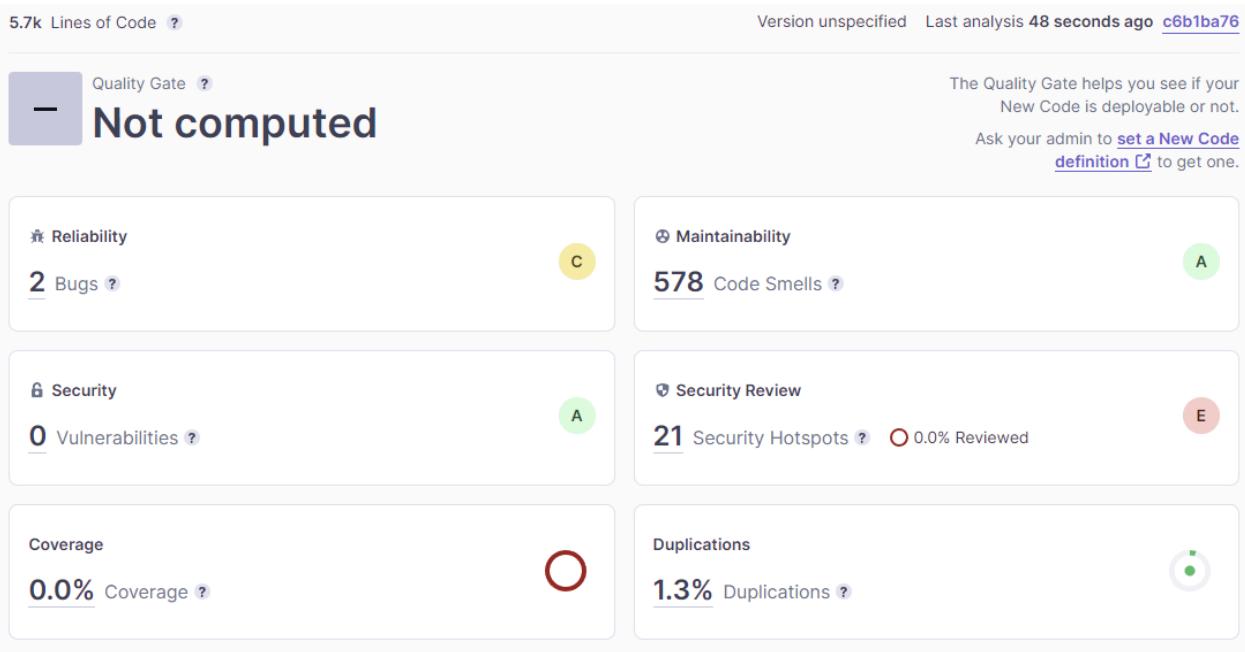


Imagen 1: <https://i.imgur.com/oXWYfx4.png>

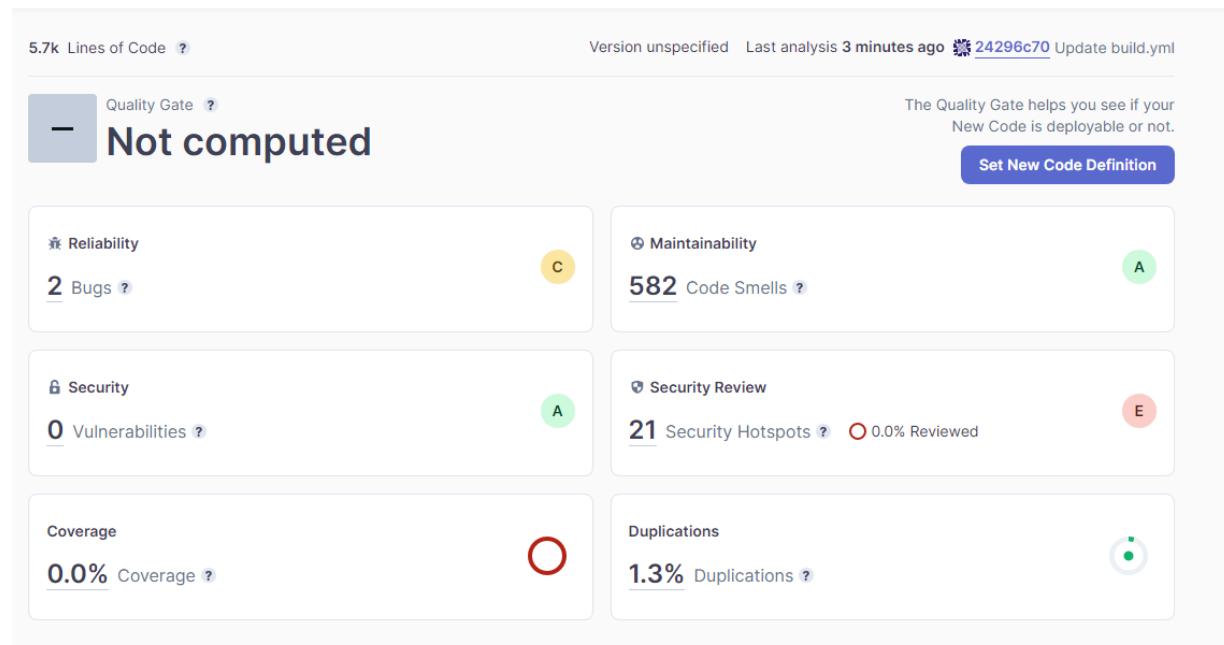
Imagen 2: <https://i.imgur.com/l02kTea.png>

Imagen 3: <https://i.imgur.com/JlzhG8T.png>

## Análisis del Sr. Marrón

### Análisis Previo a resolver las incidencias:

<https://i.imgur.com/GCHi0Az.png>



Mi integración de las dos incidencias resueltas se realizó en dos fases (**push**) a main, pues fui el primero del equipo en aprender a utilizar Sonar. Por eso aparecen dos análisis, en los cuales se observa que se ha resuelto una incidencia en cada uno.

<https://i.imgur.com/p7OIZM0.png>

Latest Activity

**NEW ANALYSIS** Main Branch

December 13 at 7:40 PM 3ed9f218 Merge remote-tracking branch 'origin/develop' into main

● 1 Fixed Issues   ● 0 New Issues   ● 0.0% Coverage   ● 0.0% Duplications   ● +1 Lines of Code

Older Activity

● Main Branch

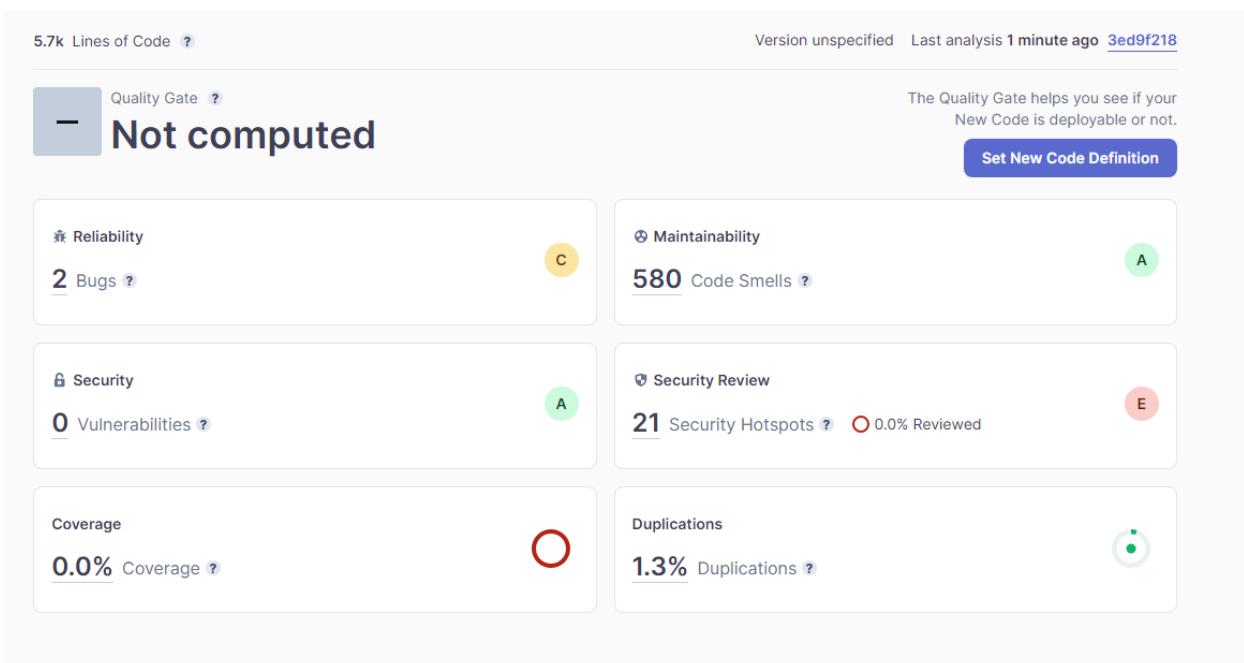
December 13 at 7:29 PM 77e47fd Merge remote-tracking branch 'origin/develop' into main

● 1 Fixed Issues   ● 0 New Issues   ● 0.0% Coverage   ● 0.0% Duplications   ● +1 Lines of Code

## Análisis Posterior a la resolución de las incidencias:

Se observa que hay dos incidencias menos de Mantenibilidad.

<https://i.imgur.com/NkK83MN.png>



## Análisis del Sr. Azul

El análisis base sobre el que se parte para resolver las dos incidencias del **Sr. Azul** es el siguiente:

<https://imgur.com/KHZnYpV.png>

Se puede observar cómo existen 576 Code Smells o incidencias de mantenibilidad. Las dos incidencias resueltas por el **Sr. Azul** son las siguientes:

<https://imgur.com/h7Fc8CL.png>

Una vez incorporados los dos commits de las dos incidencias a la rama develop y hecha la integración a la rama main, se lanzan las github actions para enviar el análisis a SonarCloud con las dos incidencias resueltas.

<https://imgur.com/vkSGeFZ.png>

<https://imgur.com/LViRvl.png>

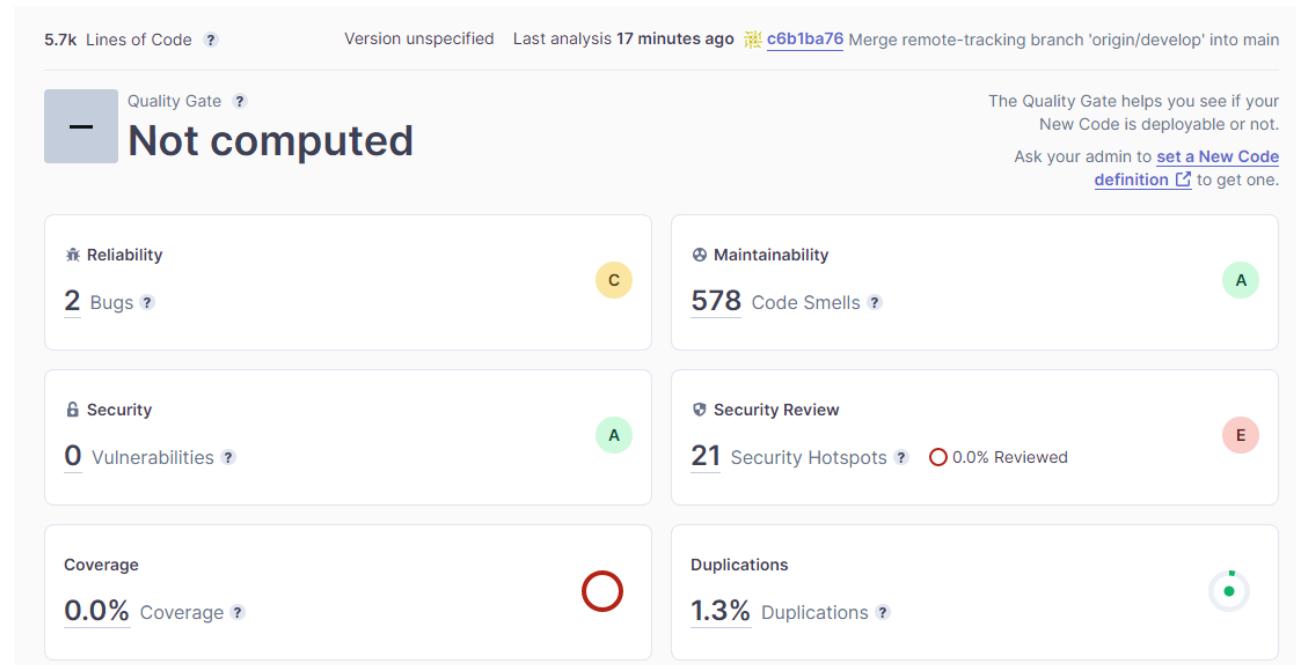
El análisis resultante al resolver las dos incidencias son las siguientes:

<https://imgur.com/qdZptfO.png>

## Análisis del Sr. Naranja

El análisis base sobre el que se parte para resolver las dos incidencias del **Sr. Naranja** es el siguiente:

<https://i.imgur.com/62naat1.png>



Se puede observar cómo existen 578 Code Smells o incidencias de mantenibilidad. Las dos incidencias resueltas por el Sr. Blanco son las siguientes:

<https://i.imgur.com/ekn2bDg.png>

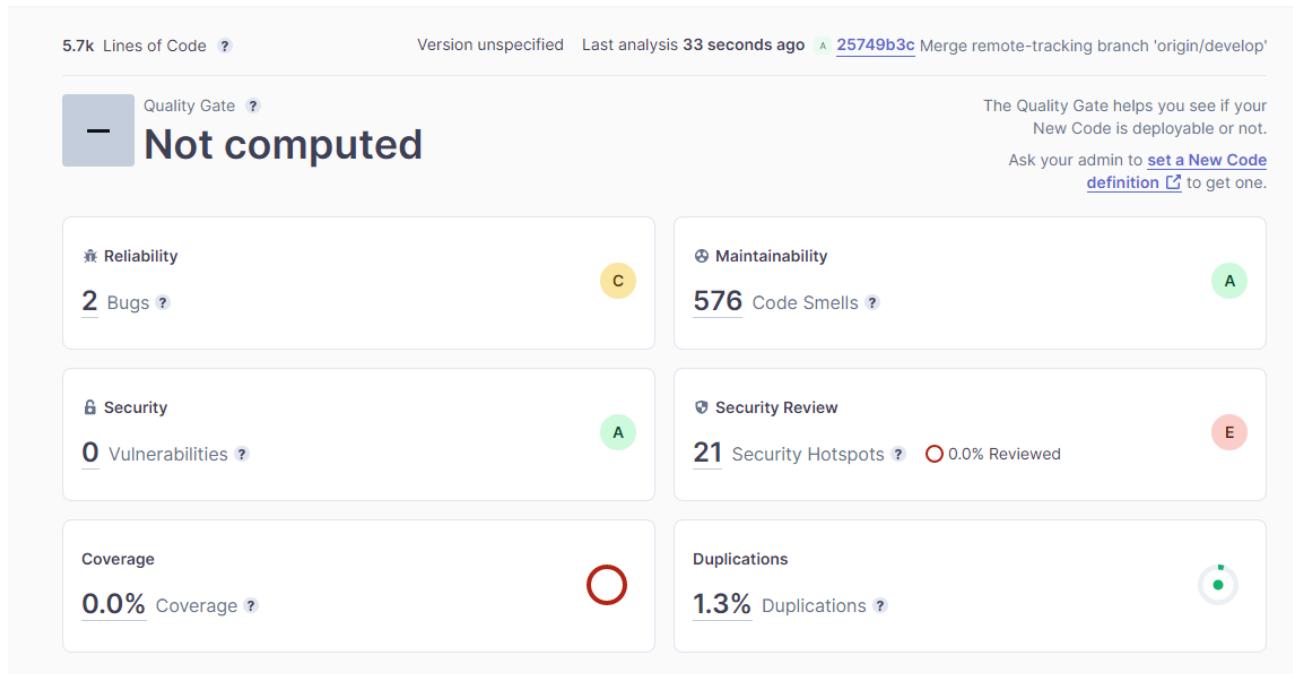
	src/main/java/com/example/proyecto/MainActivity.java	
<input type="checkbox"/>	⌚ Remove this unused import 'android.annotation.SuppressLint'. Why is this an issue?	25 days ago L6 <a href="#">Comment</a> <a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	⌚ Code Smell <span style="color: green;">Minor</span> <span style="color: blue;">Resolved (Fixed)</span> Not assigned 2min effort	No tags

<input type="checkbox"/>	⌚ Remove this unused import 'android.content.Context'. Why is this an issue?	25 days ago L7 <a href="#">Comment</a> <a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	⌚ Code Smell <span style="color: green;">Minor</span> <span style="color: blue;">Resolved (Fixed)</span> Not assigned 2min effort	No tags

Tras esto, una vez incorporados los dos commits de las dos incidencias a la rama develop y hecha la integración a la rama main, se lanzan las github actions para enviar el análisis a SonarCloud con las dos incidencias resueltas. El análisis resultante al resolver las dos incidencias son las siguientes:

<https://i.imgur.com/LZMvmB7.png>



Se puede observar que el análisis de calidad muestra 576 Code Smells o incidencias de mantenibilidad, indicando que se han podido resolver correctamente las anteriores.

## Reflexión

Desarrollar una aplicación Android es un proyecto muy interesante y desafiante que puede proporcionar gran satisfacción al ver tu trabajo en uso en dispositivos móviles. Consideramos que tanto la asignatura de ASEE como GPS son las primeras de la carrera que nos han proporcionado un método realista para nuestro futuro laboral en el desarrollo de aplicaciones Android y gestión de proyectos.

ASEE nos ha brindado una nueva perspectiva de programación diferente a la que veníamos acostumbrados. Android Studio es un framework con infinidad de configuraciones y es muy flexible a las necesidades del programador. Además, nos hemos visto empapados de la tecnología Git durante el desarrollo de la app. A pesar de conocer el controlador de versiones de cursos anteriores, es recomendable recordar y afianzar conceptos de Git asiduamente.

No nos podemos olvidar de GPS, un conjunto de conocimientos que hemos adquirido sobre la gestión y metodología de desarrollo de un proyecto. Junto con Android Studio, se suman herramientas populares como JIRA, Slack, Github Actions, SonarCloud, JUnit, Espresso, Mosquito, etc. Aun cuando las asignaturas han significado una de las más intensas y extensas cargas de trabajo en todo el grado de ingeniería informática, estamos en lo cierto que todo el esfuerzo dedicado ha merecido la pena, aunque siempre se puede dedicar un poco más de tiempo a perfeccionar las ideas de la aplicación (en referencia a la interfaz).

En resumen, desarrollar una aplicación Android es un proyecto que requiere de mucho esfuerzo, tiempo y dedicación, pero puede ser muy gratificante echar la vista atrás y ver el proceso seguido para crear la versión final.