Universidad de Valladolid

E.T.S Ingeniería Informática

Trabajo de Fin de Grado

Grado en Ingeniería Informática
Mención Ingeniería de Software

Ampliación del sistema de comunicación interno en una empresa

Autor:
Héctor Rogel Delgado
Tutor UVA:
Benjamín Sahelices Fernández
Tutor empresa:
Pablo Carrascal Muñoz

Resumen

Este Trabajo de Fin de Grado consiste en la rearquitectura completa de una aplicación iOS cuyo uso es la gestión de las vacaciones de los empleados de una empresa. Esta aplicación ya existente en la empresa requiere de un cambio completo de la misma: cambio de la arquitectura, interfaces, patrón de acceso a datos y la inclusión de tests automáticos para mejorar la calidad del "producto". También incluye un incremento de la funcionalidad de la aplicación.

Abstract

This Final Degree Project consists in a complete rearchitecture of an iOS application whose use is the holidays management of the company employees. This application already exists and requires a complete change of the whole application: architecture change, data access change and add automatic testing to improve the product quality. It also includes the implementation of new functionality to the application.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Contexto

En toda empresa, ya sea pública o privada, uno de los aspectos más importantes de la vida laboral de un empleado son las vacaciones. Cada empleado debe de solicitar sus vacaciones pero, dependiendo de la empresa esto se hace de una u otra manera. Del mismo modo la aceptación de las solicitudes.

Partiendo de una solución, en parte ya existente para este problema, surge la necesidad de actualizar y mejorar la aplicación existente. Dicha aplicación no ha sido actualizada desde 2017, pero cuenta con un backend y un frontend que pueden ser utilizados, ya que este Trabajo de Fin de Grado no se centra en estas partes.

Esta aplicación tiene un directorio de compañeros de trabajo, un tablón de anuncios donde se publican asuntos internos a la empresa, integración con las redes sociales de la empresa y la posibilidad de solicitar vacaciones.

La arquitectura de software de esta aplicación es el Modelo Vista-Controlador (MVC), el patrón de acceso a datos es el patrón observador y no tiene ningún tipo de test.

1.2 Objetivos

Los objetivos principales de este Trabajo de Fin de Grado son los siguientes:

- Cambiar la arquitectura de la aplicación al Modelo - Vista - Vista de Modelo (MVVM).
- Cambiar el patrón de acceso a datos. De patrón Observador a patrón Repositorio.
- Añadir tests unitarios, de integración y de Interfaz de Usuario.
- Añadir nuevas funcionalidades de administración:
  - Posiblilidad de aceptar o rechazar vacaciones como Administrador.

- Rediseño de alguna interfaz y consulta de vacaciones como Administrador.

En estos objetivos principales están incluidos otros como aprender el leguaje de programación Swift en su última versión, familiarizarse con la aplicación ya existente, dominar la arquitectura MVVM, así como los tests.

1.3 Contenido de la memoria

En esta sección se van a presentar de forma breve los diferentes capítulos que componen este Trabajo de Fin de Grado.

- **Capítulo 1. Introducción:** incluye en el contexto y objetivos de este Trabajo de Fin de Grado.
- **Capítulo 2. Entorno tecnológico:** presenta las herramientas, tecnologías y recursos utilizados, así como algunas definiciones teóricas.
- **Capítulo 3. Gestión y planificación del proyecto:** se exponen las razones de elección de la metodología, así como la gestión de riesgos del proyecto y una estimación de tiempo y costes.
- **Capítulo 4. Análisis:** constituye un análisis completo de la aplicación existente y se exponen los requisitos del proyecto.
- **Capítulo 5. Diseño:** consiste en el diseño de la rearquitectura completa y el diseño del incremento de la funcionalidad, además de una descripción de cada iteración del proceso de desarrollo de software utilizado.
- **Capítulo 6. Implementación y tests:** descripción de cómo se ha realizado la implementación y el testing.
- **Capítulo 7. Conclusiones:** de este Trabajo de Fin de Grado.
- **Bibliografía:** fuentes que se han consultado durante la realización de este Trabajo de Fin de Grado.
- **Anexos:** manual de usuario y de instalación.

Capítulo 2

Entorno tecnológico

2.1 Herramientas utilizadas

Las herramientas utilizadas para la realización de este Trabajo de Fin de Grado son las siguientes:

- XCode (versión 10.1): entorno o herramienta de desarrollo integrado (IDE) creado por Apple para macOS y que permite desarrollar software para macOS, iOS, watchOS y tvOS.
- Simulator (versón 10.1): simulador integrado en XCode, permite simular las aplicaciones desarrolladas en un dispositivo virtual.
- Jira Software: herramienta desarrollada por Atlassian cuya finalidad es administrar las tareas de un proyecto permitiendo el seguimiento de errores e incidencias.
- Git: software de gestión de versiones.
- Gitlab: servicio web basado en Git que permite la gestión de repositorios, control de versiones y desarrollo de software colaborativo.
- OverLeaf: editor de LaTeX online.

2.2 Tecnologías utilizadas

- Swift (versión 4.2): lenguaje de programación creado por Apple para el desarrollo de aplicaciones iOS y macOS.
- LaTeX: sistema para la composición de textos utilizado para la realización de este documento.

- CoreData: Core Data es un framework proporcionado por Apple que permite administrar los datos de nuestra capa de datos. Este framework tiene como ventajas poder agrupar, filtrar y organizar los datos y reducir el espacio de memoria de estos datos gracias al "faulting", que permite que cuando realizamos una "query", solamente obtenemos la entidad solicitada en vez de todas las entidades relacionadas a la solicitada.

2.3 Recursos hardware

En este apartado se describen los dispositivos necesarios para la realización de este Trabajo de Fin de Grado. Se ha decidido añadir este apartado debido a que, posteriormente, se va realizar una estimación de los costes del proyecto y hemos considerado el conocimiento de los recursos hardware como algo importante para esa estimación.

Ordenador de sobremesa	
Sistema operativo	Windows 10
Procesador	AMD Ryzen 5 1600 3.2GHz
Memoria	12GB DDR4
Disco duro	2TB
Tarjeta gráfica	GeForce GTX 1050Ti

Tabla 2.1: Descripción ordenador de sobremesa

MacBook Pro	
Sistema operativo	macOS Mojave 10.14.3
Procesador	Intel Core i5 2,5GHz
Memoria	16GB DDR3
Disco duro	500GB
Tarjeta gráfica	Intel HD Graphics

Tabla 2.2: Descripción ordenador portátil

Iphone 6	
Sistema operativo	iOS 12.0
Procesador	1.5 GHz de doble núcleo de 64 bits
Memoria	2GB LPDDR4
Disco duro	16GB
Tarjeta gráfica	PowerVR GT7600

Tabla 2.3: Descripción Iphone 6

Iphone 6 Plus	
Sistema operativo	iOS 12.0
Procesador	1.5 GHz de doble núcleo de 64 bits
Memoria	2GB LPDDR4
Disco duro	32GB
Tarjeta gráfica	PowerVR GT7600

Tabla 2.4: Descripción Iphone 6 Plus

Iphone X	
Sistema operativo	iOS 12.0
Procesador	2.4 GHz de seis núcleos de 64 bits
Memoria	3GB LPDDR4
Disco duro	64GB
Tarjeta gráfica	PowerVR GT7600

Tabla 2.5: Descripción Iphone X

2.4 Scrum

Scrum es un modelo para el desarrollo de todo tipo de procesos ágiles. Es una metodología ágil de desarrollo de software que busca flexibilizar el proceso, evitar secuenciación y lograr una estrecha colaboración entre el equipo de desarrollo y el cliente. Sus principios fundamentales son:

- Colaboración con el cliente. El cliente está muy presente en esta metodología debido a que, de esta forma, el producto final será más satisfactorio para el mismo porque puede proponer cambios en cualquier momento del desarrollo.

- Respuesta al cambio. Debido a que el cliente puede cambiar los requisitos en cualquier momento del desarrollo, debemos estar abiertos y preparados para cualquier cambio.
- Desarrollo incremental y entregas funcionales frecuentes. El desarrollo del producto se lleva a cabo en fases periódicas y al final de cada fase el producto debe ser funcional.
- Supresión de artefactos innecesarios en la gestión del proyecto para hacer el proceso más ágil al eliminar tares poco útiles.
- Asignación de un bloque de tiempo y priorización basada en el valor. Cada tarea en una iteración es priorizada.
- Colaboración entre los miembros del equipo y el cliente.

2.4.1 Roles

En el desarrollo con esta metodología aparecen los siguientes roles:

- Product Owner. Su función es definir los objetivos del proyecto y del producto, y planificar y revisar cada Sprint o iteración.
- Scrum Master. Es el responsable de apoyar al equipo de desarrollo, eliminar las barreras organizativas y mantener la consistencia del proyecto, haciendo que se cumplan los principios ágiles.
- Team. Grupo de personas que trabajan en el desarrollo del producto de forma conjunta.

2.4.2 Artefactos

Los proyectos ágiles usan tradicionalmente siete artefactos:

- Establecimientos de la visión del producto. El entorno y los objetivos del producto.
- User Story. Es cada uno de los requisitos del proyecto.
- Product backlog. Es la lista total de los requisitos del proyecto, ordenados por prioridad. Esta lista puede crecer con cada iteración.
- Product roadmap. Es una vista de alto nivel de los requisitos del producto con poca precisón sobre cuando se desarrollarán dichos requisitos.
- Plan de release. Es un calendario para el lanzamiento de software funcional.
- Sprint backlog. Es una lista con las historias de usuario (requisitos) y tareas asociadas con cada iteración.
- Incremento. Es la funcionalidad del producto tras cada iteración.

2.4.3 Eventos

En el desarrollo ágil existen seis tipos de eventos que se llevan a cabo:

- Reunión inicial del proyecto. En ella se planifica a grandes rasgos el proyecto. Incluye una visión inicial del producto y un product roadmap.
- Sprint. Es un ciclo de desarrollo corto con la finalidad de crear funcionalidad. Cada Sprint es una iteración cuya duración es entre una y cuatro semanas.
- Sprint Planning. Una reunión al comenzar cada Sprint dónde se fijan los objetivos, funcionalidad a desarrollar, posibles cambios sobre la funcionalidad de Sprints anteriores o solucionar fallos o errores encontrados durante el anterior Sprint. También se asigna, entre todo el equipo, un valor a cada User Story que refleja el esfuerzo necesario para realizar esa User Story.
- Daily. Reunión diaria de entre 5 o 10 minutos, que se realiza de pie, dónde se indica lo realizado el día anterior y lo que se va a realizar durante este día. También se pueden compartir los problemas encontrados e intentar buscar una solución con los demás miembros del equipo.
- Demo o revisión del Sprint. Se lleva a cabo al final de cada Sprint y tiene como objetivo mostrar las funcionalidades desarrolladas durante el Sprint y comprobar que éste se ha cumplido con éxito.
- Retrospectiva. Reunión al final de cada Sprint dónde se analiza lo qué funcionó bien, qué se debe cambiar y cómo llevar a cabo los cambios.

2.5 Modelo, Vista, Controlador

El Modelo-Vista-Controlador es un patrón de arquitectura software que permite separar la lógica de negocio de la lógica de la vista. Con este patrón se busca potenciar la facilidad de mantenimiento y reutilización de código y la separación de responsabilidades en la aplicación. Se basa fundamentalmente en la separación de estas responsabilidades en tres capas:

- Modelo: representa la lógica de negocio.
- Vista: es la representación de los datos en interfaces.
- Controlador: es el intermediario entre el modelo y la vista. Responde a acciones del usuario, modificando el modelo cuando sea necesario. Se comunica con la vista para que se actualice con los últimos cambios del modelo.

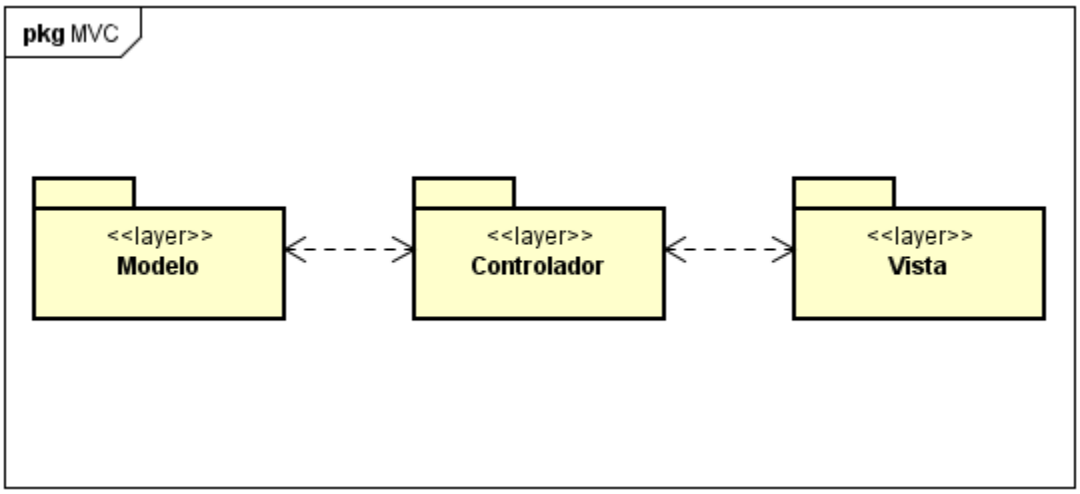

Figura 2.1: Arquitectura MVC

2.6 Modelo, Vista, Vista de Modelo

MVVM (del inglés Model - View - View Model) es un patrón de arquitectura software que, como muchos otros, permite desacoplar la lógica de la aplicación de la lógica de la vista de la aplicación. Esta división de responsabilidades tiene el objetivo de simplificar las tareas de desarrollo para posibilitar tener varios desarrolladores trabajando en cada parte de código, así como una mayor facilidad de reutilización de código y de mantenimiento del código.

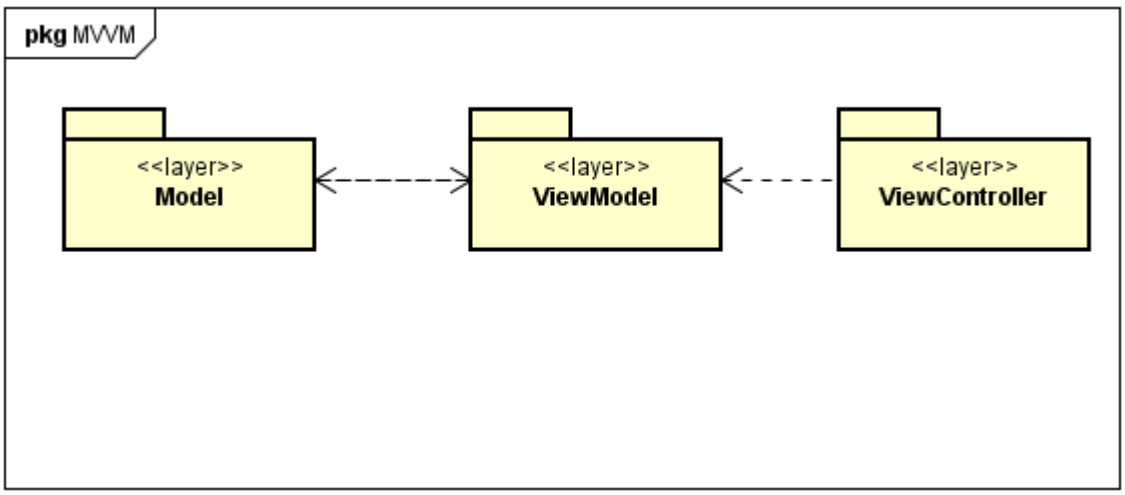

Figura 2.2: Arquitectura MVVM

- Model (Modelo): representa la lógica de negocio. Debe ser una estructura de código

sencilla (class o struct) que define las propiedades de un objeto y no tiene funcionalidad.

- View Controller (Vista): no debe tener lógica, sólo se encarga de modificar los elementos de la vista según lo que ordene el Modelo de la Vista. Incluye las interfaces.
- View Model (Modelo de la Vista): es el intermediario entre el Modelo y la Vista, contiene la lógica de presentación.

2.7 Patrón repositorio

Para el acceso a los datos necesarios por la vista de la aplicación usamos el patrón repositorio. Como definición, el patrón repositorio es una fachada que abstrae el dominio de la persistencia, es decir, es una capa intermedia entre la aplicación y la base de datos. Con este patrón buscamos tener una estructura desacoplada en la que las acciones que se realizan en cada parte estén bien definidas y separadas. Este cuenta con tres partes: Repository, APIClient y Network.

- Repository: Se encarga de guardar los datos recibidos en base de datos, recuperar los datos necesarios de la base de datos y devolver dichos datos a la Vista del Modelo.
- APIClient: Su función es decodificar los datos que devuelve el servidor (JSON) para pasarlos al repositorio en el formato requerido para su almacenamiento.
- Network: Su función es lanzar la solicitud al servidor y, con la respuesta de este, recuperar el JSON con los datos solicitados para pasarlo al APIClient.

2.8 Patrón observador

El patrón observador es un patrón de diseño de software que permite "observar" cambios en los objetos y notificar los cambios que se producen en los mismos. Cuando un objeto es modificado, se notifica a sus observadores para que realicen una acción concreta. Permite desacoplar la vista de la acción a realizar.

Capítulo 3

Gestión y planificación del proyecto

3.1 Introducción

El desarrollo de este proyecto se llevará a cabo con SCRUM, cuya principal característica es que es una metodología ágil de desarrollo que permite un desarrollo incremental, entre otras cosas.

Se ha optado por esta metodología debido a que, en el comienzo del proyecto, el alcance del mismo no está definido.

3.2 Descripción de la metodología elegida

Scrum es una metodología ágil de desarrollo software (aunque se puede usar en otros ámbitos). Pese a que es una metodología diseñada para el trabajo en grupo, se va a usar en este Trabajo de Fin de Grado debido a las ventajas que supone. Estas ventajas son:

- Resultados periódicos: debido a que en cada sprint vamos a tener que modificar partes de la aplicación, es necesario que podamos ver los resultados en cada sprint review.
- Requisitos cambiantes: esto nos ofrece flexibilidad durante el desarrollo.
- Contacto entre el equipo de desarrollo y el Product Owner: las reuniones periódicas son beneficiosas para mantener este contacto y así el producto será de más calidad y siempre adaptado a las pretensiones del cliente.

Rol	**Nombre**
Product Owner	Pablo Carrascal
Scrum Master	Pablo Carrascal y Héctor Rogel
Team	Héctor Rogel

Tabla 3.1: Descripción roles del proyecto

3.3 Gestión de riesgos

En esta sección se van a definir los riesgos potenciales que pueden aparecer durante el desarrollo del proyecto. También se va a definir un plan de contingencia para cada riesgo para que, en caso de que se produzca el riesgo, tengamos una respuesta al mismo.

Los riesgos tienen un identificador, nombre, descripción, nivel de impacto, probabilidad de que suceda y un plan de contingencia. El nivel de impacto puede ser de cuatro tipos:

- Catastrófico: el proyecto fracasaría.
- Crítico: tanto el rendimiento del proyecto como de su desarrollo se verían seriamente afectados.
- Marginal: afecta a problemas secundarios de un proyecto.
- Despreciable: problemas menores que no causen un retraso considerable en el proyecto.

ID	1
Nombre	Pérdida documentos
Descripción	Pérdida de documentos o de código generado en la implementación debido a fallo eléctrico o similar.
Impacto	Crítico
Probabilidad	Baja
Plan de contingencia	Restauración del trabajo de las copias de seguridad en la nube.

Tabla 3.2: R01 - Pérdida de documentos

ID	2
Nombre	Necesidad conocimientos
Descripción	Necesidad de adquirir nuevos conocimientos sobre las tecnologías utilizadas que ocasione disminución de la productividad y problemas de incorrección en las estimaciones iniciales.
Impacto	Marginal
Probabilidad	Alta
Plan de contingencia	Formación individual en los supuestos conocimientos necesarios.

Tabla 3.3: R02 - Necesidad de conocimientos

ID	3
Nombre	Baja miembro del equipo
Descripción	Alguno de los miembros del equipo está de baja
Impacto	Crítico
Probabilidad	Media-baja
Plan de contingencia	Reorganizar y planificar el proyecto.

Tabla 3.4: R03 - Baja miembro del equipo

ID	4
Nombre	Fallo equipo
Descripción	Fallo de alguno de los equipos necesarios para el desarrollo del proyecto.
Impacto	Crítico
Probabilidad	Media
Plan de contingencia	Buscar nuevos equipos, ya sean prestados o comprados (como última opción), reorganizar y planificar el proyecto.

Tabla 3.5: R04 - Fallo equipo

ID	5
Nombre	Retraso en sprint
Descripción	El desarrollo sufre algún retraso en alguno de sus sprints
Impacto	Crítico
Probabilidad	Alta
Plan de contingencia	Planificar el siguiente sprint para realizarlo en menos tiempo y así no sufrir retrasos en el proyecto.

Tabla 3.6: R05 - Retraso en sprint

ID	6
Nombre	Mala estimación historias de usuario
Descripción	Mala estimación de las historias de usuario que provocan retrasos en el desarrollo del proyecto.
Impacto	Crítico
Probabilidad	Media
Plan de contingencia	Utilizar el siguiente sprint para finalizar el sprint actual. Reorganizar y planificar el proyecto.

Tabla 3.7: R06 - Mala estimación historias de usuario

3.4 Estimaciones

3.4.1 Estimación de tiempo

En la reunión inicial determinamos que la duración de cada Sprint es de una semana debido a que es la primera vez que el equipo trabaja con esta tecnología y permite al Product Owner y al Srum Master un mejor seguimiento de los avances realizados por el equipo. Cada User Story se vota por los miembros del equipo usando la sucesión de Fibonacci.

Se ha estimado que el trabajo a realizar en este Trabajo de Fin de Grado se va a llevar a cabo en 16 Sprints (o fases) que se distribuyen en el tiempo como se muestra en la Tabla 3.8. Es importante destacar que todas estas fechas son aproximadas.

Sprint	Fecha de inicio	Fecha de fin
0	11/02/2019	17/02/2019
1	18/02/2019	24/02/2019
2	25/02/2019	03/03/2019
3	04/03/2019	10/03/2019
4	11/03/2019	17/03/2019
5	18/03/2019	24/03/2019
6	25/03/2019	31/03/2019
7	01/04/2019	07/04/2019
8	08/04/2019	14/04/2019
9	15/04/2019	21/04/2019
10	22/04/2019	28/04/2019
11	29/04/2019	05/05/2019
12	06/05/2019	12/05/2019
13	03/06/2019	09/06/2019
14	10/06/2019	16/06/2019
15	17/06/2019	23/06/2019
16	23/06/2019	30/06/2019

Tabla 3.8: Estimación temporal de los Sprints del proyecto

3.4.2 Estimación de costes

Pese a que en trabajo es realizado por un estudiante universitario y no tiene costes reales, se va a realizar una estimación de los mismos. Este Trabajo de Fin de Grado se desarrolla aproximadamente en 600 horas, estimando que el coste de un desarrollador de software es de 10 euros por hora, el coste humano sería de 6.000 euros.

También debemos de tener en cuenta los costes materiales:

- Ordenadores. Se calcula el coste de estos teniendo en cuenta su vida media estimada de 3 años y que se han usado durante 4 meses:
  - Ordenador de sobremesa. Coste en el proyecto: 78 €.
  - MacBookPro. Coste en el proyecto: 133€.
  - Iphone. Coste aproximado de todos los Iphone utilizados: 220€.
- Tasa de desarrollador Apple: 100€.

Por lo que podemos estimar un coste total del proyecto de 6.531€.

3.4.3 Tiempo real

Como en todo proyecto de desarrollo de software, hay ocasiones en las que los tiempos previstos en la planificación no se cumplen, ya sea por imprevistos o problemas de cualquier tipo. Esta sección muestra las modificaciones con respecto a la estimación de tiempo que se han producido durante del desarrollo. Es importante destacar que todas estas fechas son aproximadas.

Sprint	**Fecha de inicio**	**Fecha de fin**
0	11/02/2019	17/02/2019
1	18/02/2019	24/02/2019
2	25/02/2019	03/03/2019
3	04/03/2019	10/03/2019
4	11/03/2019	17/03/2019
5	18/03/2019	24/03/2019
6	25/03/2019	31/03/2019
7	01/04/2019	07/04/2019
8	08/04/2019	21/04/2019
9	22/04/2019	05/05/2019
10	06/05/2019	12/05/2019
11	03/06/2019	09/06/2019
12	10/06/2019	16/06/2019
13	17/06/2019	23/06/2019
14	24/06/2019	30/06/2019

Tabla 3.9: Desviación temporal de los Sprints del proyecto

Capítulo 4

Análisis

4.1 Proyecto existente

4.1.1 Introducción

En esta sección se va a hablar de los resultados del trabajo de análisis realizado sobre la aplicación ya existente. Este análisis incluye el funcionamiento de la aplicación y sus interfaces, el diseño de la misma, la arquitectura empleada, y un análisis en profundidad del código.

Hay que destacar que el código existente es código legado de más de 2 años de antiguedad, escrito en Swift 3 (cuya versión está cerca de no tener soporte con la salida de Swift 5). A estas dificultades le añadimos el desconocimiento de dicho lenguaje de programación.

4.1.2 Descripción de la aplicación existente

Es un aplicación interna a la empresa SolidGear cuya principal función es la solicitud de vacaciones por parte de los empleados de la misma. También incluye otras funciones como la visualización de los tweets de la cuenta de Twitter, perfil de usuario, tablón de noticias, sección para ver los demás trabajadores...

4.1.2.1 Login

La primera pantalla que aparece es la de login. Es necesario introducir un nombre de usuario y contraseña que se autentifican con el LDAP de la empresa.

4.1.2.2 Vista principal

Tras introducir el login en la aplicación, la pantalla principal muestra los tweets de la cuenta de la empresa.

4.1.2.3 Menú lateral

En la parte superior de la pantalla hay un icono que, si lo pulsamos, accedemos al menú lateral. Este menú contiene enlaces a cada sección de la aplicación.

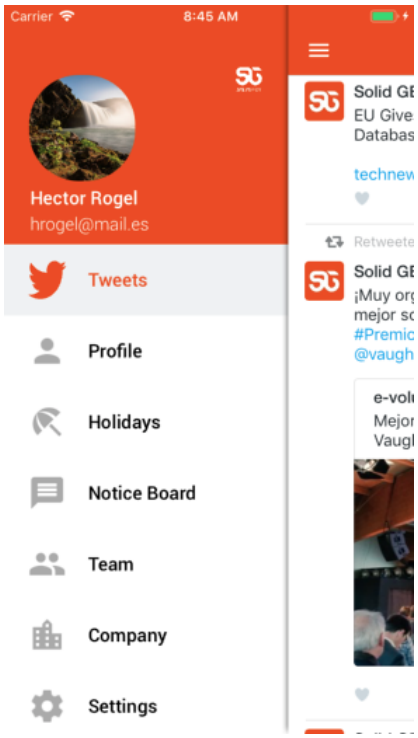

Figura 4.1: Interfaz del Menú Lateral

4.1.2.4 Perfil

En esta sección aparecen los datos de la persona que ha hecho login, es decir, el perfil del usuario. Aparece el nombre y apellidos, la posición en la empresa, fecha de nacimiento, número de télefono, foto de perfil, twitter y linkedin. Todos estos datos son opcionales, sólo aparecen si el usuario los ha introducido (excepto el nombre y apellidos). También hay un botón para editar el perfil y añadir todos los datos al perfil del usuario. Además, ofrece la posibilidad de cerrar sesión de la aplicación.

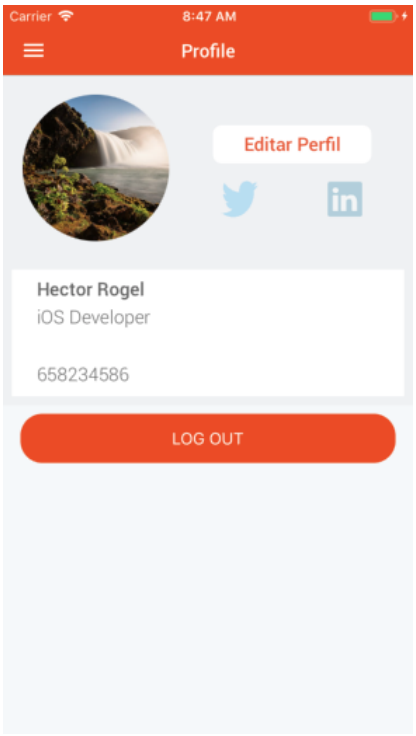

Figura 4.2: Interfaz del Perfil

En la pantalla de editar perfil se pueden editar o añadir los datos descritos anteriormente.

4.1.2.5 Vacaciones

La sección de vacaciones es la más importante de la aplicación. En ella podemos consultar el número de días de vacaciones asignados para el año actual, el número de días de vacaciones restante y una descripción con las solicitudes de vacaciones realizadas. Dichas solicitudes tienen un estado que puede ser: pendiente de aprobación, rechazado, aprobado. Cada solicitud indica el número de días solicitados y las fechas de las vacaciones. Esta pantalla ofrece también la posibilidad de solicitar vacaciones pulsando el botón de la parte superior derecha.

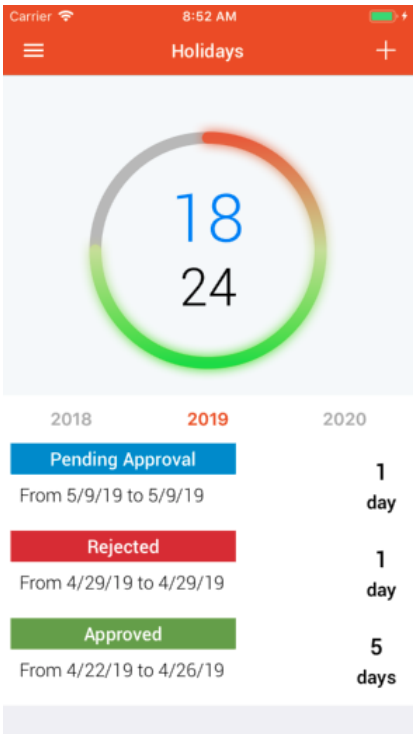

Figura 4.3: Interfaz de Vacaciones

En la pantalla de solicitud de vacaciones podemos solicitar vacaciones eligiendo un rango de fechas.

4.1.2.6 Tablón de anuncios

En esta sección podemos consultar los anuncios de cambio de estado en la solicitud de vacaciones (por ejemplo, de pendiente de aprobación a aprobado). El tablón se completa con avisos relacionados con la empresa.

4.1.2.7 Equipo

En la sección del equipo aparecen los miembros de la empresa, con los datos que ha actualizado cada uno en su perfil: imagen, posición en la empresa, mail, teléfono, twitter y linkedin. Si pulsamos el icono del teléfono de algún compañero (sólo en caso de estar activo) podemos llamar por teléfono a dicho compañero. Si pulsamos el icono de twitter (en caso de estar activo) accedemos a su cuenta de twitter en el navegador web del dispositivo, de la misma forma ocurre con el icono de linkedin.

4.1.2.8 Empresa

En la sección de la empresa podemos ver información relativa a la misma: nombre, teléfono, página web, dirección...

4.1.2.9 Ajustes

La sección de ajustes nos permite activar o desactivar la opción de recibir notificaciones de la aplicación.

4.1.3 Diseño de la aplicación existente

Hay que destacar que los diagramas de diseño de este capítulo provienen del Trabajo de Fin de Grado "SGEmployee: Aplicación iOS para la gestión de las vacaciones laborales". cuyo autor es el tutor en empresa de este Trabajo de Fin de Grado. Dichos diagramas se usan con su consentimiento.

4.1.3.1 Modelo de dominio

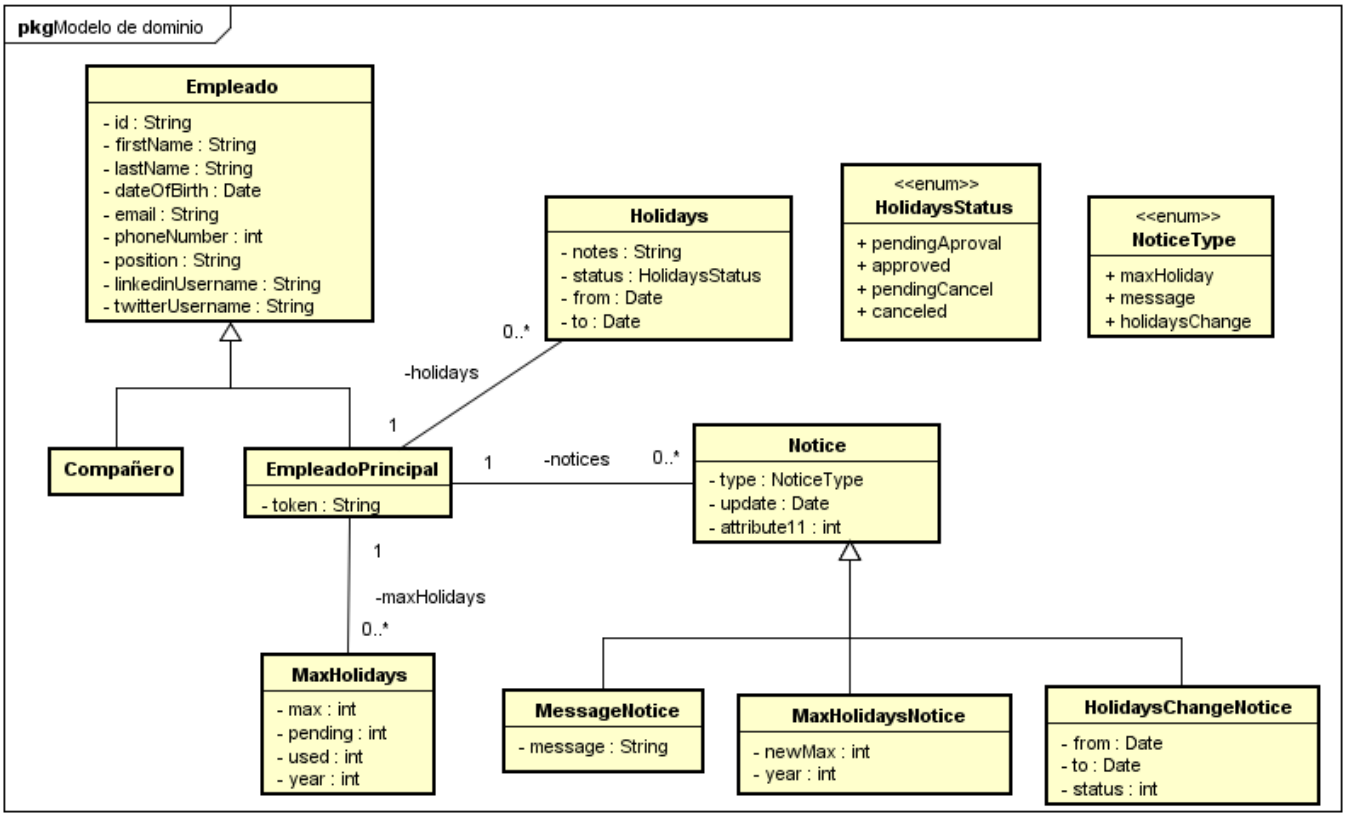

Figura 4.4: Modelo de dominio de la aplicación

4.1.3.2 Arquitectura

La arquitectura utilizada es la más que conocida MVC (modelo, vista controlador) y sigue el esquema que se muestra a continuación.

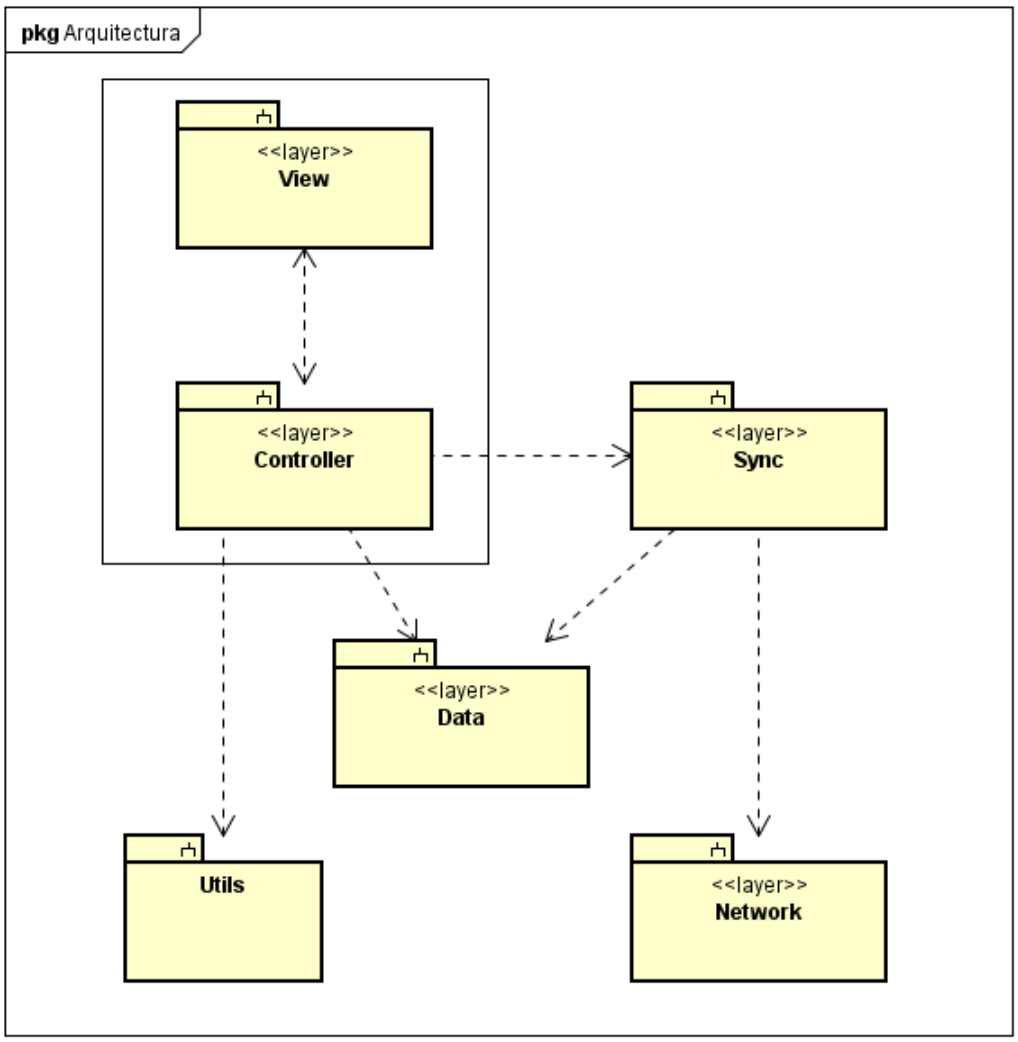

Figura 4.5: Arquitectura MVC de la aplicación existente

Para, más adelante, hacer la comparación con la nueva arquitectura que vamos a implementar, vamos a describir cuál es la funcionalidad de cada una de las capas de esta arquitectura.

- View y Controller: View es la interfaz, y Controller se encarga de gestionar qué elementos y cuando se muestran en la interfaz. Están muy acopladas.
- Sync: En esta capa se piden datos a la capa Network y se actualizan en la base de datos. También se encarga de avisar al Controller de que los datos han sido actualizados.
- Network: Se encarga de realizar las peticiones HTTP al servidor y de devolver la respuesta a la capa Sync.

- Data: Esta capa está compuesta por los "managers" responsables de las operaciones CRUD de la base de datos. Esta capa constituye el Modelo de este MVC.

4.1.3.3 Descomposición modular

En esta sección podemos ver la descomposición modular de la aplicación existente, con las diferentes capas que se han descrito en el apartado anterior.

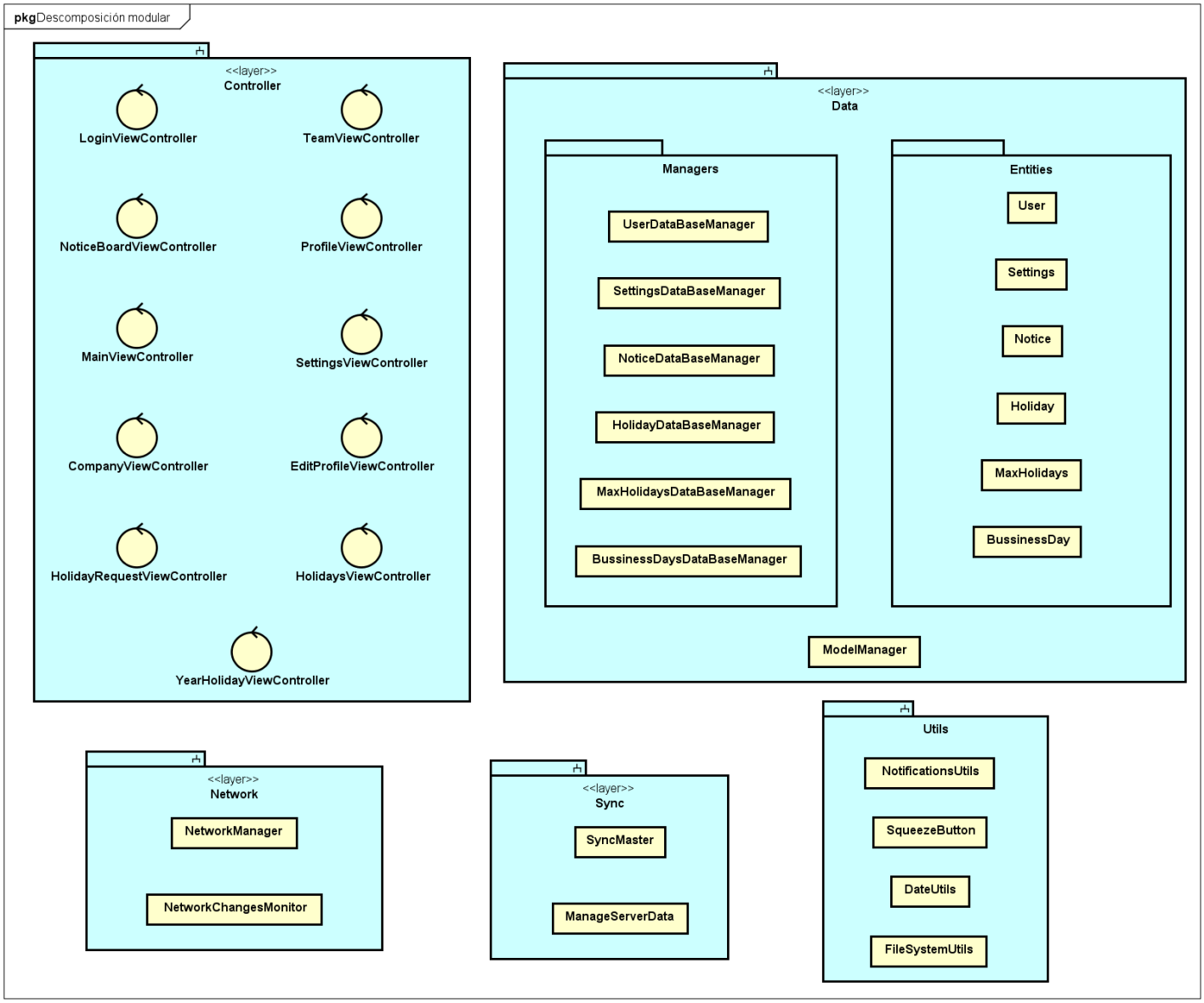

Figura 4.6: Descomposición modular de la aplicación existente

4.1.4 Interfaz

Toda la interfaz de la aplicación se encontraba en un mismo storyboard. Un storyboard es una representación visual de la interfaz de usuario de una aplicación iOS que muestra el contenido de las diferentes pantallas que contiene la aplicación y la conexión entre dichas pantallas mediante segues. Un segue define la transición entre dos controladores de vista. Esta estructura provocaba, a pesar que no es una aplicación extremadamente grande, que XCode funcionara de forma lenta y dificultaba la corrección de bugs o la necesidad de cambio de alguna de las vistas.

4.1.5 Modelo de datos

La siguiente figura muestra el esquema relacional de la base de datos interna de la aplicación.

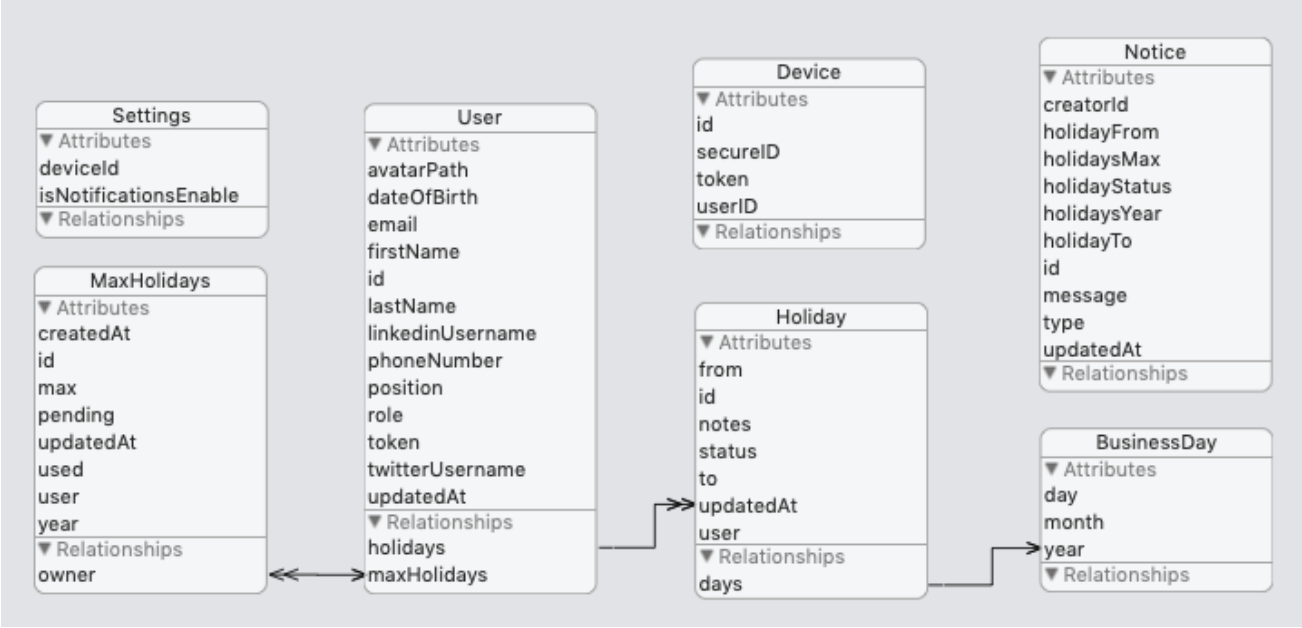

Figura 4.7: Esquema relacional de la base de datos

4.1.6 Acceso a datos

Para el acceso a los datos necesarios para la aplicación, se usa el patrón observador. También se utiliza CoreData como base de datos de la aplicación.

Este patrón es muy utilizado en toda la aplicación, y no únicamente para el acceso a datos, aunque en este Trabajo de Fin de Grado la parte que nos interesa es la del acceso a datos. Dicho patrón es implementado en la aplicación con la ayuda de la clase de tipo

Singleton NSNotificationCenter.

Para el almacenamiento de los datos de la aplicación en el dispositivo, se usa Core Data, descrito con anterioridad en el Capítulo 2.

4.2 Análisis incremento funcionalidad

Para el análisis de esta parte de incremento de la funcionalidad, se parte del análisis de la otra parte de la aplicación, la ya esxistente. Esto es debido a que el análisis es el mismo para ambas partes, por ejemplo, el modelo de dominio va a ser exactamente igual, de la misma forma con el modelo de datos...

4.3 Requisitos del proyecto (Product Backlog)

En esta sección se van a ir añadiendo las Historias de Usuario (o requisitos del proyecto) que se van a llevar a cabo. La gran mayoría de las que aparecen a continuación se han definido antes de comenzar con el desarrollo del proyecto, pero alguna de ellas se ha añadido durante el desarrollo y otras se han modificado durante el mismo. Esto es posible gracias a la flexibilidad del proceso de desarrollo Scrum.

1. Instalación de recursos.
2. Preparar documentación.
3. Aprender swift.
4. Primera aplicación iOS.
5. Clonación repositorio.
6. Entender arquitectura.
7. Entender Modelo Vista Vista Modelo (MVVM).
8. Cambiar la vista de "Settings" a MVVM paso por paso.
9. Cambiar la vista de "Company" a MVVM paso por paso.
10. Cambiar la vista principal a MVVM paso por paso.
11. Cambiar la vista del equipo a MVVM paso por paso.
12. Cambiar la arquitectura de acceso a datos de la sección del equipo al patrón repositorio paso por paso.
13. Añadir tests a la sección del equipo.

14. Rearquitectura sección del "Perfil" de usuario y tests.
15. Rearquitectura de la sección de "Vacaciones" y tests.
16. [BUG] Pérdida de los datos de vacaciones cuando se cambia de año.
17. [BUG] En ocasiones las vacaciones máximas no se muestran cuando se accede a la sección de Vacaciones.
18. [BUG] Después de añadir una solicitud de vacaciones, las vacaciones del usuario se muestran duplicadas.
19. Actualizar código a Swift 4.2.
20. Rearquitectura de la sección de Avisos.
21. Rearquitectura de la sección de Login.
22. [BUG] El año por defecto en la sección de Vacaciones debe ser el actual.
23. [BUG] Datos de vacaciones restantes y máximas no se recargan en el primer acceso.
24. [BUG] Después de modificar la imagen de perfil, esta no se recarga en la aplicación.
25. Ver solicitudes pendientes en nueva sección.
26. Aprobar o rechazar solicitudes de vacaciones.
27. Ver historial de vacaciones del usuario cuando se va a aceptar o rechazar la solicitud.
28. Rediseño sección de Equipo.
29. Ver vacaciones de cada usuario desde la sección de Equipo.

Capítulo 5

Diseño

5.1 Introducción

Como ya se ha indicado con anterioridad, este trabajo se divide en dos partes claramente diferenciadas. En primer lugar, el cambio completo de la arquitectura de la aplicación iOS ya existente; y en segundo lugar, el incremento de la funcionalidad de la misma. En ambas partes está incluida la realización de tests automáticos. En este capítulo se va a describir el nuevo diseño de la rearquitectura y de la nueva funcionalidad.

5.2 Cambio de la arquitectura

5.2.1 Finalidad

Cabe destacar que, de los objetivos expuestos en la sección de Objetivos del Capítulo 1 de este TFG, el más importante es la inclusión de tests a tres niveles, y que el cambio de arquitectura de la aplicación es necesario para la inclusión de estos tests. De la misma manera ocurre con el cambio del acceso a datos. Esto se explicará con más detalle en las siguientes secciones de este Capítulo.

5.2.2 Contexto

A modo de resumen de la sección 4.1 del capítulo anterior:

- La aplicación utilizaba MVC (Modelo Vista Controlador) como patrón de arquitectura de software.
- Toda la interfaz se encontraba en un mismo storyboard, lo que provocaba que XCode funcionara de forma lenta y dificultaba la corrección de bugs o la necesidad de cambio de alguna de las vistas.

- Para el acceso a los datos necesarios para la aplicación, se usa el patrón Observador con la ayuda de la clase NotificationCenter.
- La aplicación era completamente funcional pero se encontraba en un estado de acoplamiento que hacía muy complicada la tarea de incluir tests en varios niveles, por lo que no tenía ningún tipo de test.

A continuación se van a explicar varios conceptos que van a ser utilizados en la rearquitectura de la aplicación.

5.2.3 Arquitectura

La figura 5.1 muestra la nueva arquitectura de la aplicación aplicando MVVM y el patrón repositorio.

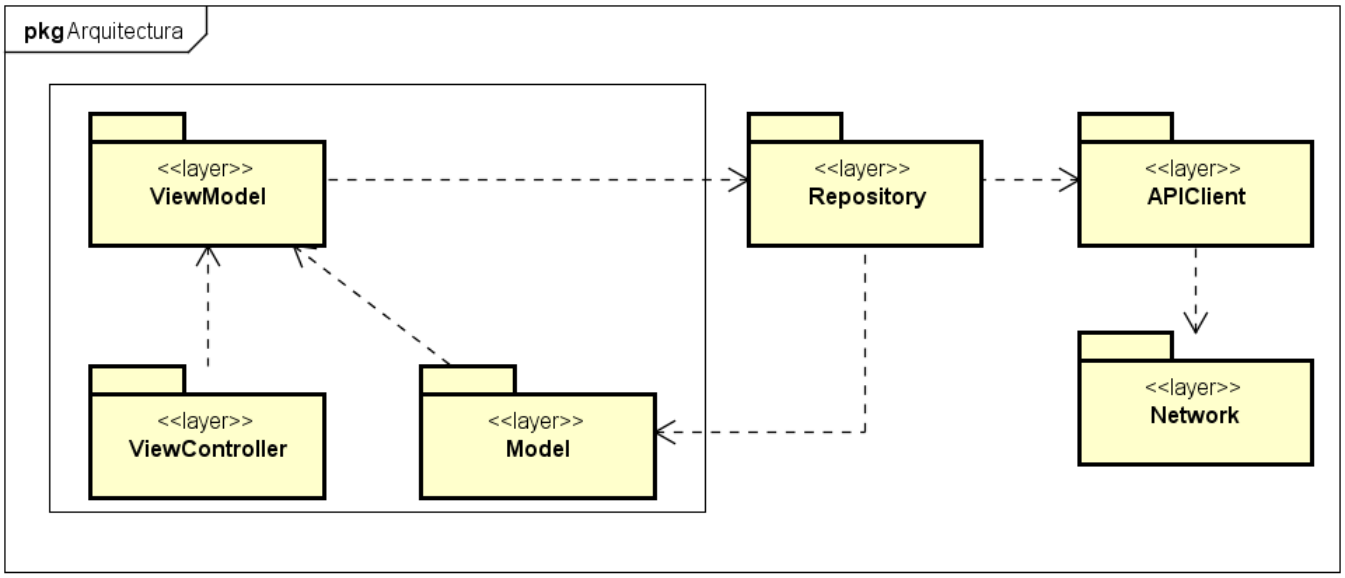

Figura 5.1: Nueva arquitectura de la aplicación

Model: Esta capa está compuesta por la lógica de negocio y los "managers" responsables de las operaciones CRUD de la base de datos.

5.2.4 Descomposición modular

En esta sección podemos ver cómo será la descomposición modular de la aplicación tras la rearquitectura completa.

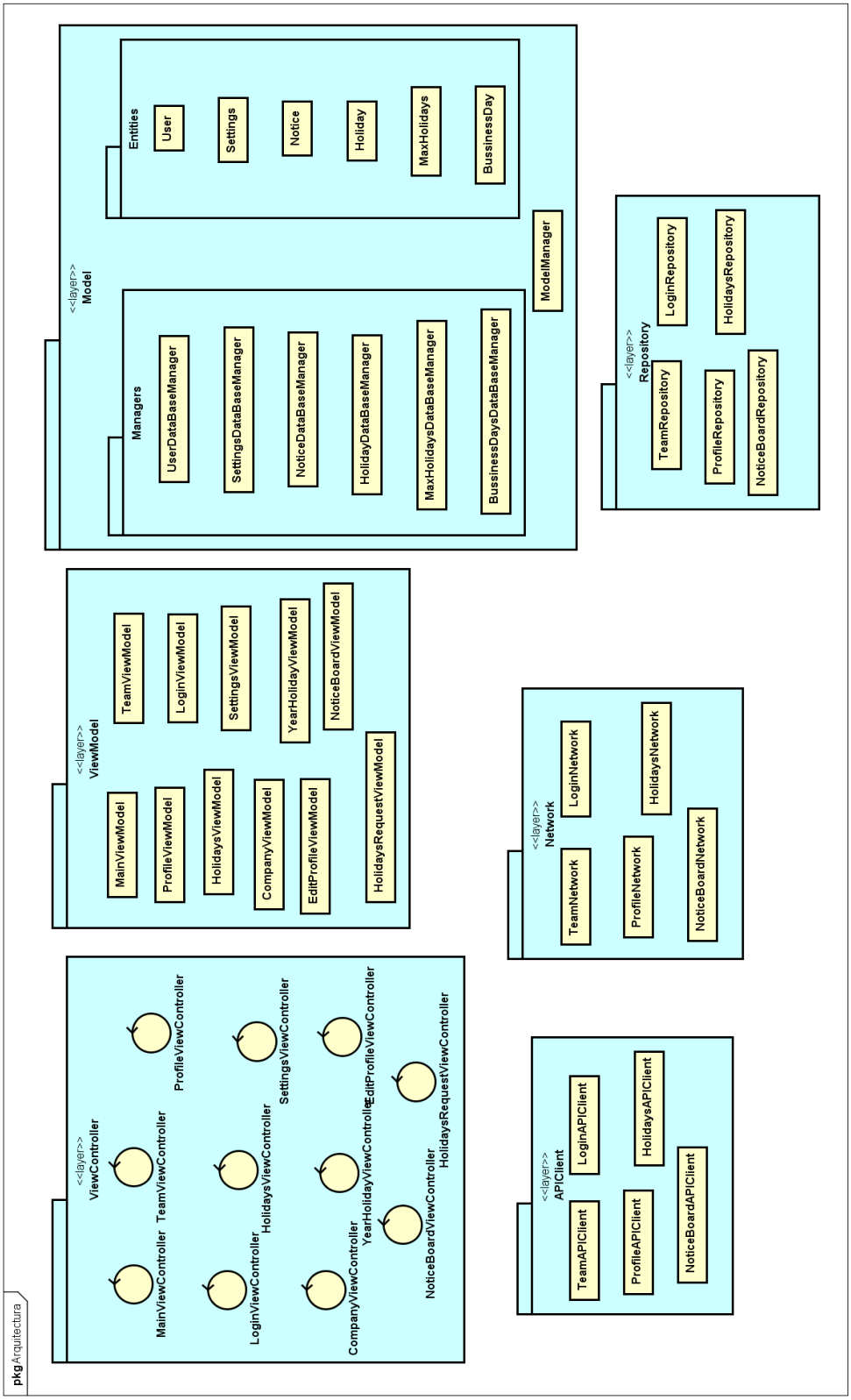

Figura 5.2: Nueva descomposición modular

5.2.5 Diseño modular de cada capa

En todo este apartado, se van describir los módulos de cada capa agrupados por secciones de la aplicación para, de esta forma, tener una mejor visibilidad de cada módulo.

El diseño de estas capas se ha realizado teniendo en cuenta el código ya existente y tomando las decisiones oportunas para su mejor adecuación para la rearquitectura que se está realizando. Estas decisiones son muy importantes ya que en cada iteración la aplicación debe ser funcional y no debe provocar fallos que no existían antes del cambio.

5.2.5.1 Diseño modular ViewController

Sección de Login. Gestiona el éxito o error en la autenticación del usuario, mostrando un mensaje de error o accediendo a la aplicación, además de la posiblilidad de recuperar la contraseña.

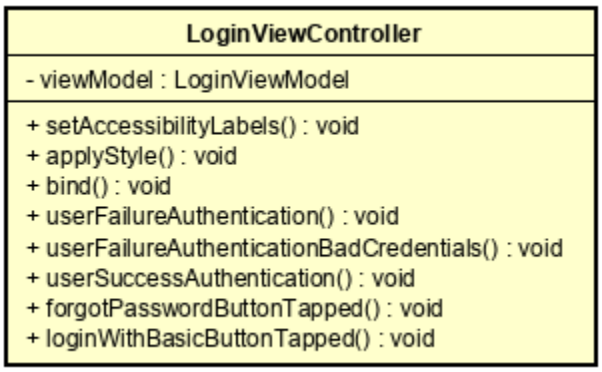

Figura 5.3: Diseño detallado rearquitectura, capa ViewController, sección de Login

ViewController sección Principal. Se encarga de decir a la vista que tiene que aplicar un estilo concreto a la misma y mostrar el TimeLine de twitter.

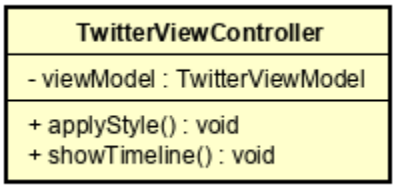

Figura 5.4: Diseño detallado rearquitectura, capa ViewController, sección Principal (twitter)

ViewController sección perfil. Tenemos dos ViewController, uno para el perfil del usuario y otro para la interfaz de Editar Perfil.

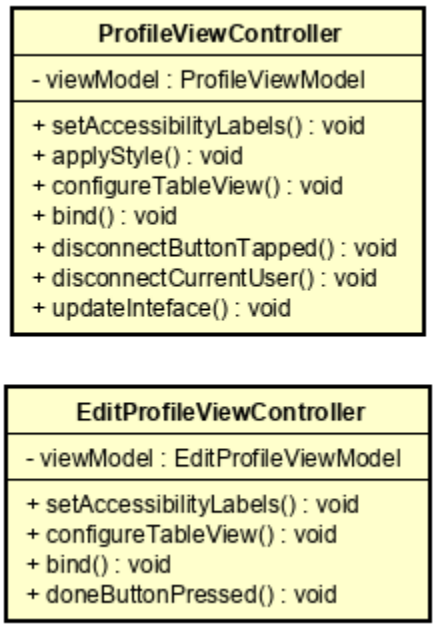

Figura 5.5: Diseño detallado rearquitectura, capa ViewController, sección de Perfil

ViewController Vacaciones. Tenemos tres ViewController debido a que la interfaz de Vacaciones tiene dos partes (o sub-interfaces): ViewController y YearHolidayViewModel. La tercera clase es para la interfaz de solicitud de Vacaciones. Deben mostrar en la vista los datos del número de vacaciones máximas, datos de vacaciones y enviar nuevas solicitudes a su ViewController.

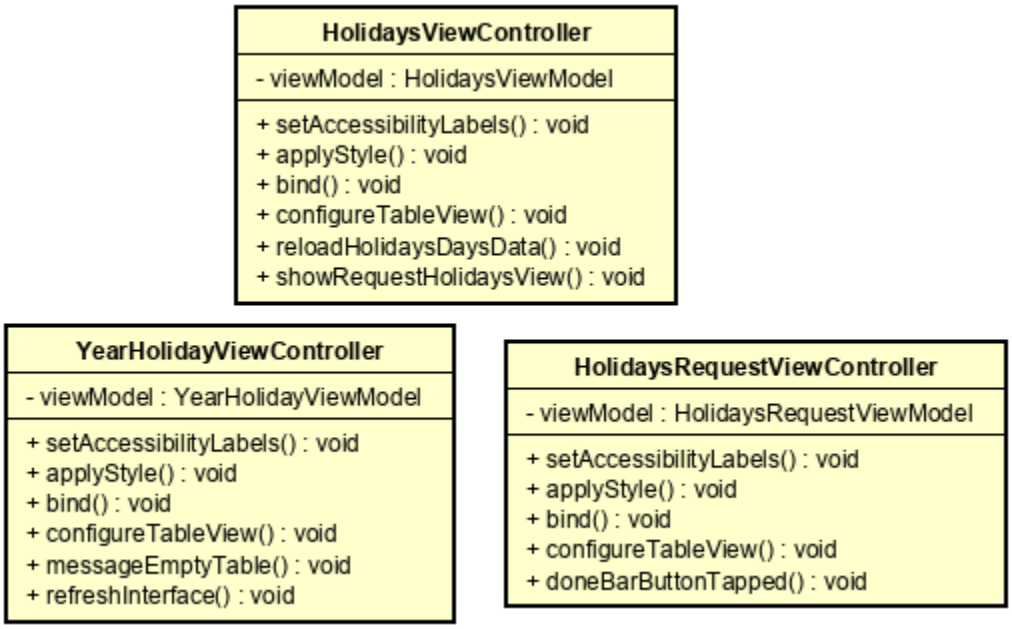

Figura 5.6: Diseño detallado rearquitectura, capa ViewController, sección de Vacaciones

ViewController sección Tablón de Anuncios. Se encarga de comunicar a la vista los anuncios que debe mostrar y cuándo debe hacerlo.

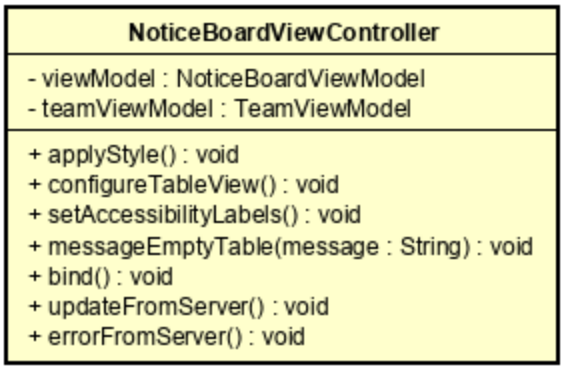

Figura 5.7: Diseño detallado capa ViewController, sección del Tablón de Anuncios

ViewController sección Equipo. Comunica la información sobre los empleados a la vista y gestiona la interacción con los diferentes botones que aparecen en la celda de cada trabajador.

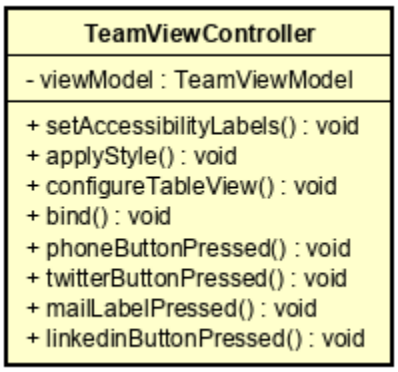

Figura 5.8: Diseño detallado rearquitectura, capa ViewController, sección de Equipo

ViewController sección Empresa. Su función es comunicar la información de la empresa que debe mostrar la vista y gestionar la detección de "gestos" (tocar la pantalla, deslizar...).

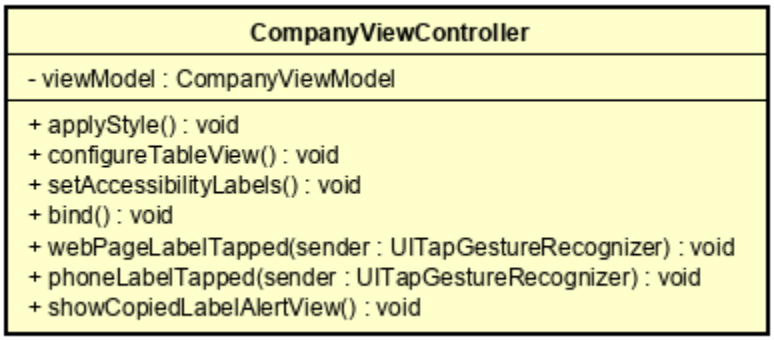

Figura 5.9: Diseño detallado rearquitectura, capa ViewController, sección de Empresa

ViewController sección Settings. Gestiona las notificaciones dependiendo de un "switch" cuyo estado es reconocido por este ViewController.

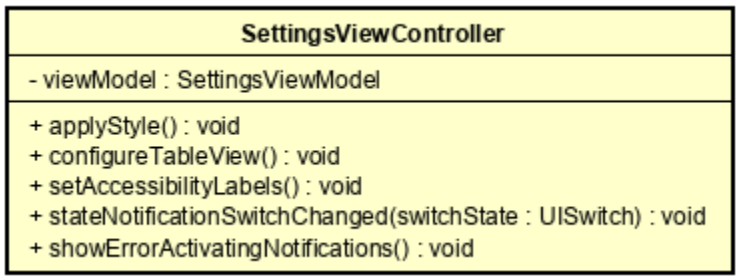

Figura 5.10: Diseño detallado rearquitectura, capa ViewController, sección de Settings

5.2.5.2 Diseño modular ViewModel

En primer lugar, veremos la sección de Login en la aplicación, la cual tiene que comunicar a su ViewController correspondiente si la autenticación se ha realizado con éxito y recuperar la imagen de perfil del usuario identificado.

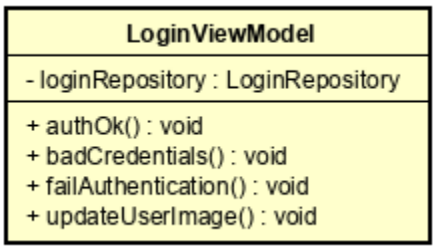

Figura 5.11: Diseño detallado rearquitectura, capa ViewModel, sección de Login

Sección de Twitter. Únicamente tiene que llevar la cuenta de los tweets que aparecen en la interfaz para que, en caso de que no tenga tweets, su ViewController muestre un mensaje que avise de que no se han podido cargar los tweets.

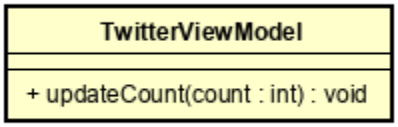

Figura 5.12: Diseño detallado rearquitectura, capa ViewModel, sección Principal (twitter)

Sección de Perfil. Tenemos dos ViewModel debido a que uno es para la interfaz del Perfil y otro para Editar Perfil. Debe recuperar los datos del perfil y enviar nuevos datos para almacenarlos en el servidor.

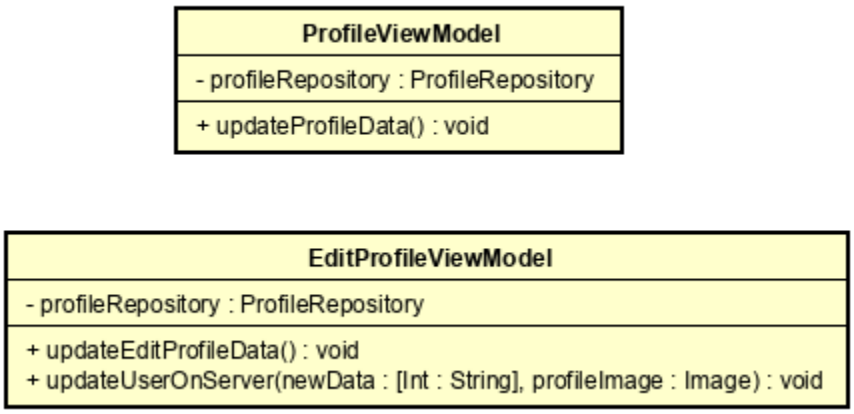

Figura 5.13: Diseño detallado rearquitectura, capa ViewModel, sección de Perfil

Sección de Vacaciones. Tenemos tres ViewModel debido a que la interfaz de Vacaciones tiene dos partes (o sub-interfaces): HolidaysVIewModel y YearHolidayViewModel. La tercera clase es para la interfaz de solicitud de Vacaciones. Deben recuperar los datos del número de vacaciones máximas, datos de vacaciones y subir nuevas solicitudes al servidor.

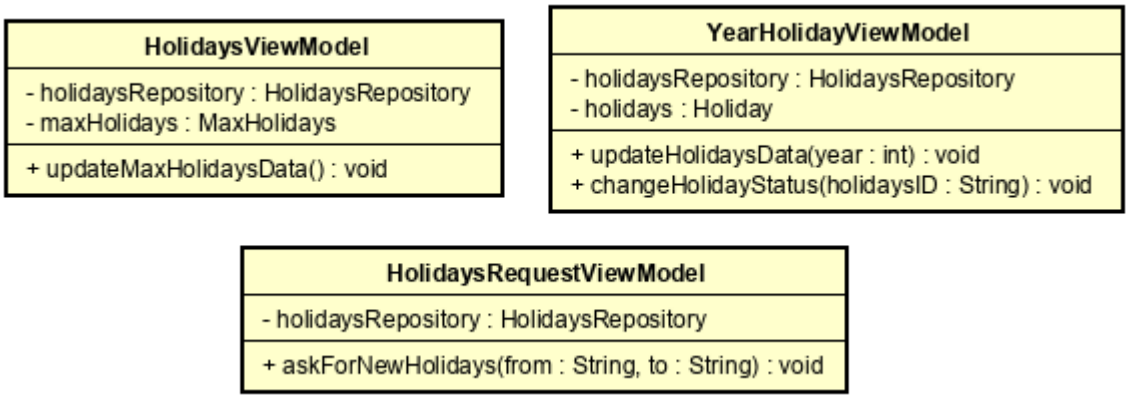

Figura 5.14: Diseño detallado rearquitectura, capa ViewModel, sección de Vacaciones

Sección del Tablón de Anuncios. Se encarga de enviar los datos de las "noticias", que vienen de su Repositorio, a su ViewController.

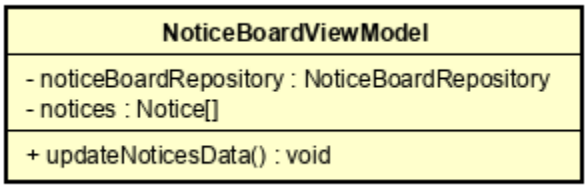

Figura 5.15: Diseño detallado rearquitectura, capa ViewModel, sección de Tablón de Anuncios

ViewModel sección Equipo. Se encarga de comunicar a su ViewController los datos de los trabajadores.

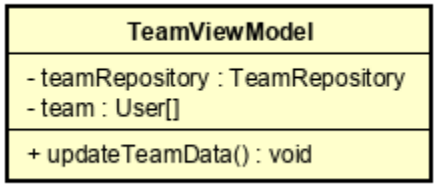

Figura 5.16: Diseño detallado rearquitectura, capa ViewModel, sección de Equipo

Sección de información de la Empresa.

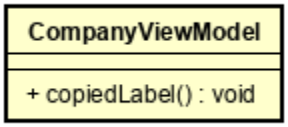

Figura 5.17: Diseño detallado rearquitectura, capa ViewModel, sección de Empresa

Sección de Settings. Debe saber cuáles son las "settings" iniciales que ha configurado el usuario para comunicar a su ViewController, cuál es el estado del "switch" que debe de mostrar y saber cuál es el estado del "switch" para comunicarlo a su ViewController.

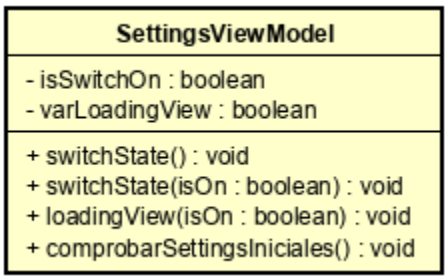

Figura 5.18: Diseño detallado rearquitectura, capa ViewModel, sección de Settings

5.2.5.3 Diseño modular capa Model

En la siguiente figura podemos ver el diseño detallado de la capa Model. Esta capa no ha cambiado porque ya existía en la aplicación. Aún así se ha decidido realizar el diagrama para la mejor comprensión de esta capa.

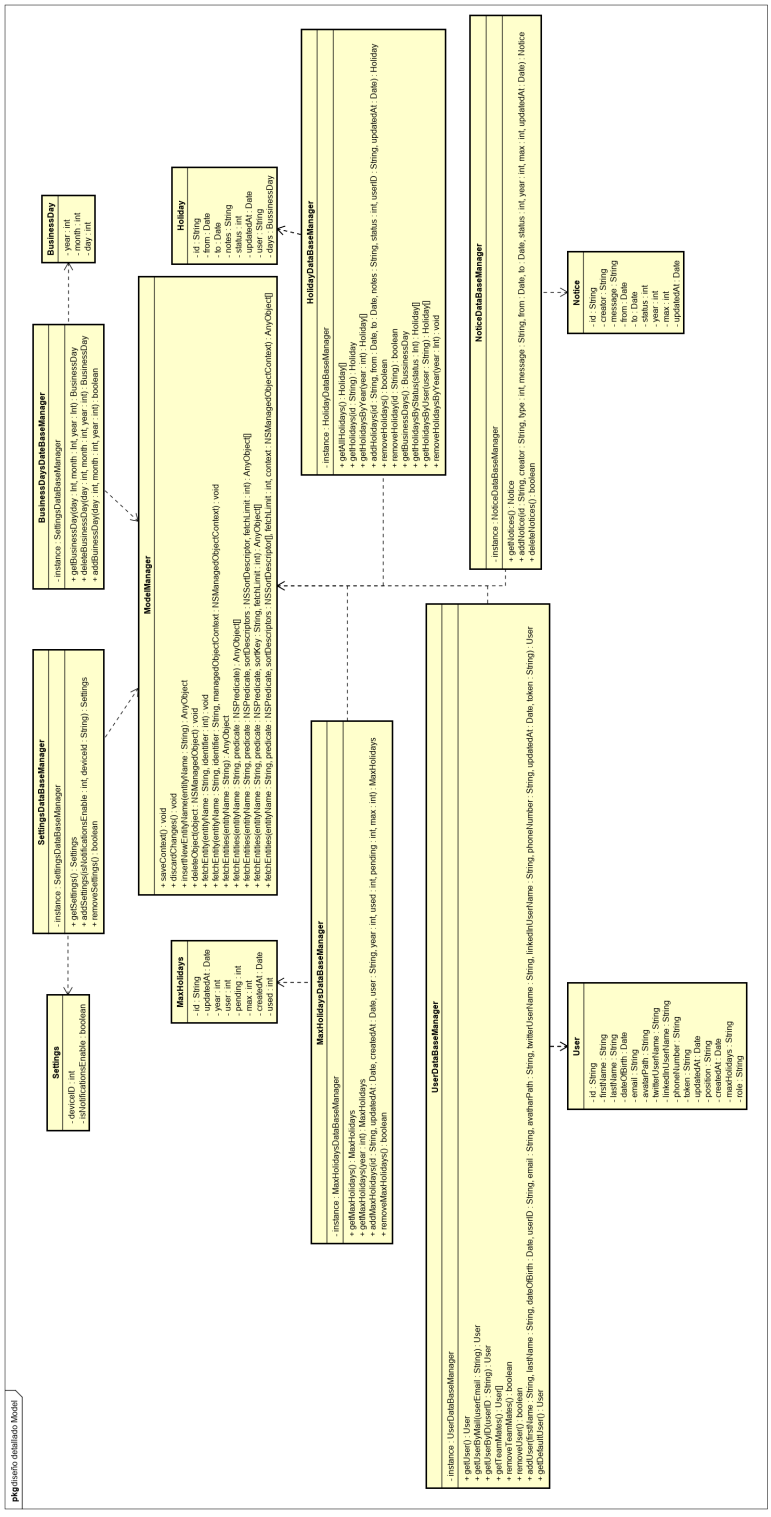

Figura 5.19: Diseño detallado capa Model

5.2.5.4 Diseño modular capa Repository

En la capa Repository tenemos las clases que requieren datos del servidor para almacenarlos y/o extraerlos de la Base de Datos del dispositivo (CoreData). También se encuentran aquellas que únicamente requieren extraer datos, que de una u otra manera ya están almacenados en la Base de Datos, como es el caso de MainMenuRepository.

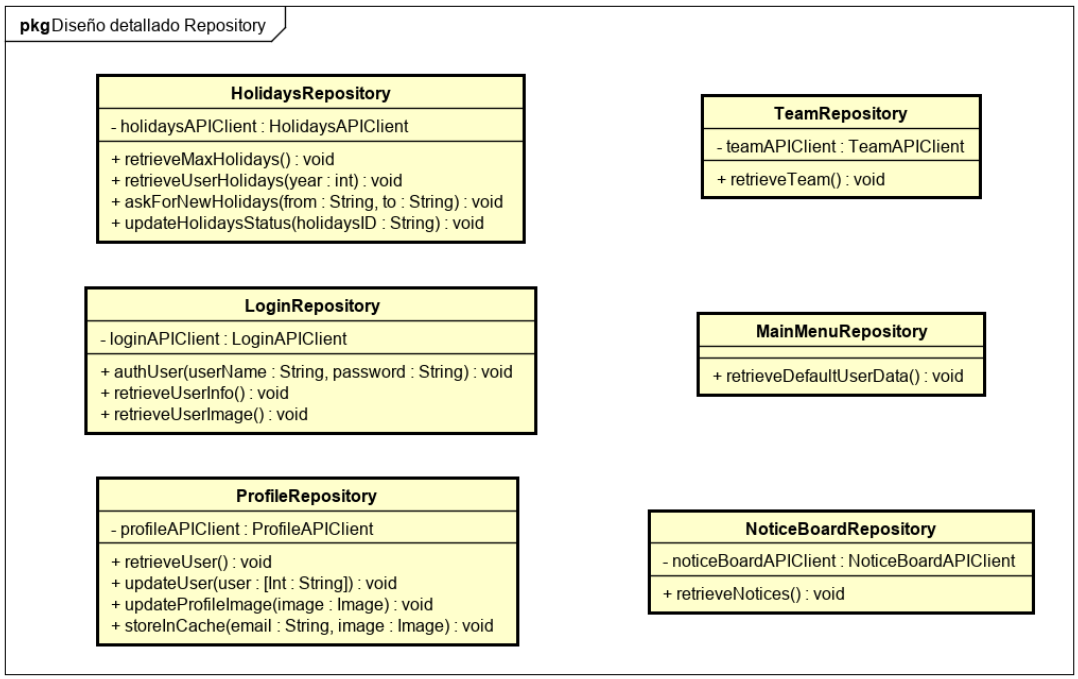

Figura 5.20: Diseño detallado rearquitectura, capa Repository

5.2.5.5 Diseño modular capa APIClient

En la capa APIClient tenemos las clases que requieren de información del servidor, que tiene que pasar por esta capa para su decodificación.

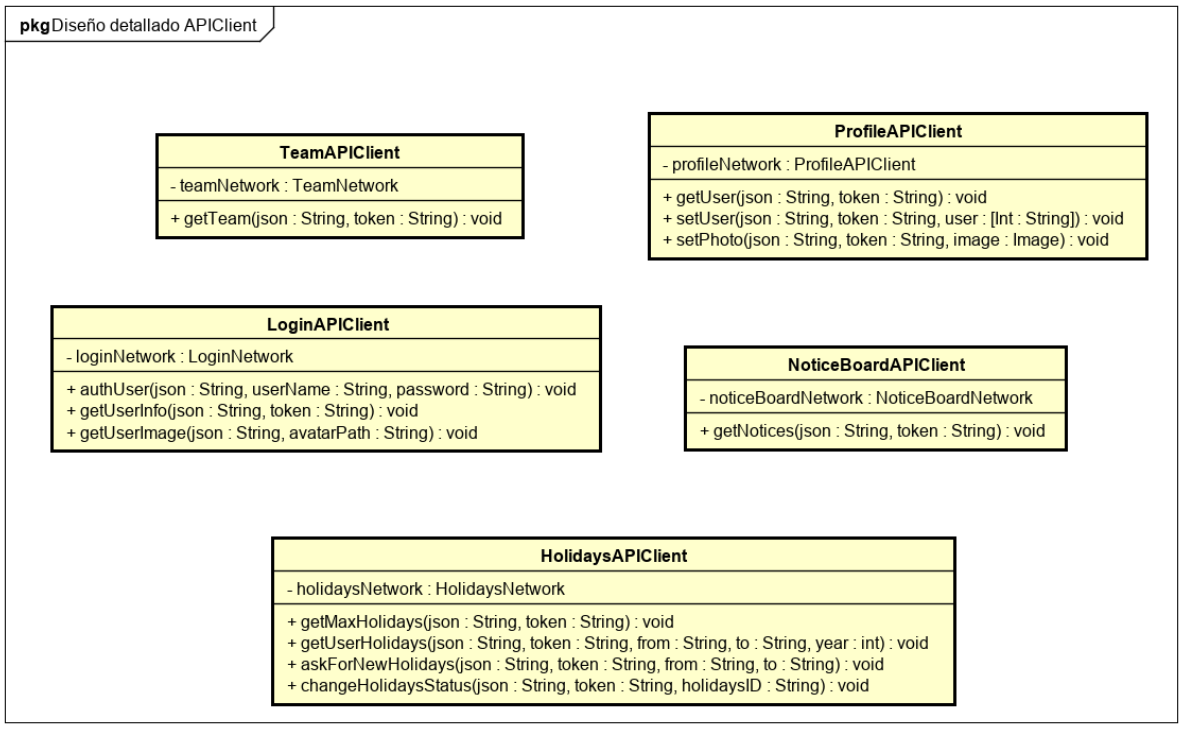

Figura 5.21: Diseño detallado rearquitectura, capa APIClient

5.2.5.6 Diseño modular capa Network

La capa Network tiene una clase general que es usada por las otras cinco para establecer la conexión con la librería Alamofire. Las clases restantes corresponden a las secciones de la aplicación que requieren de datos del servidor. En estas se realizan las peticiones al servidor.

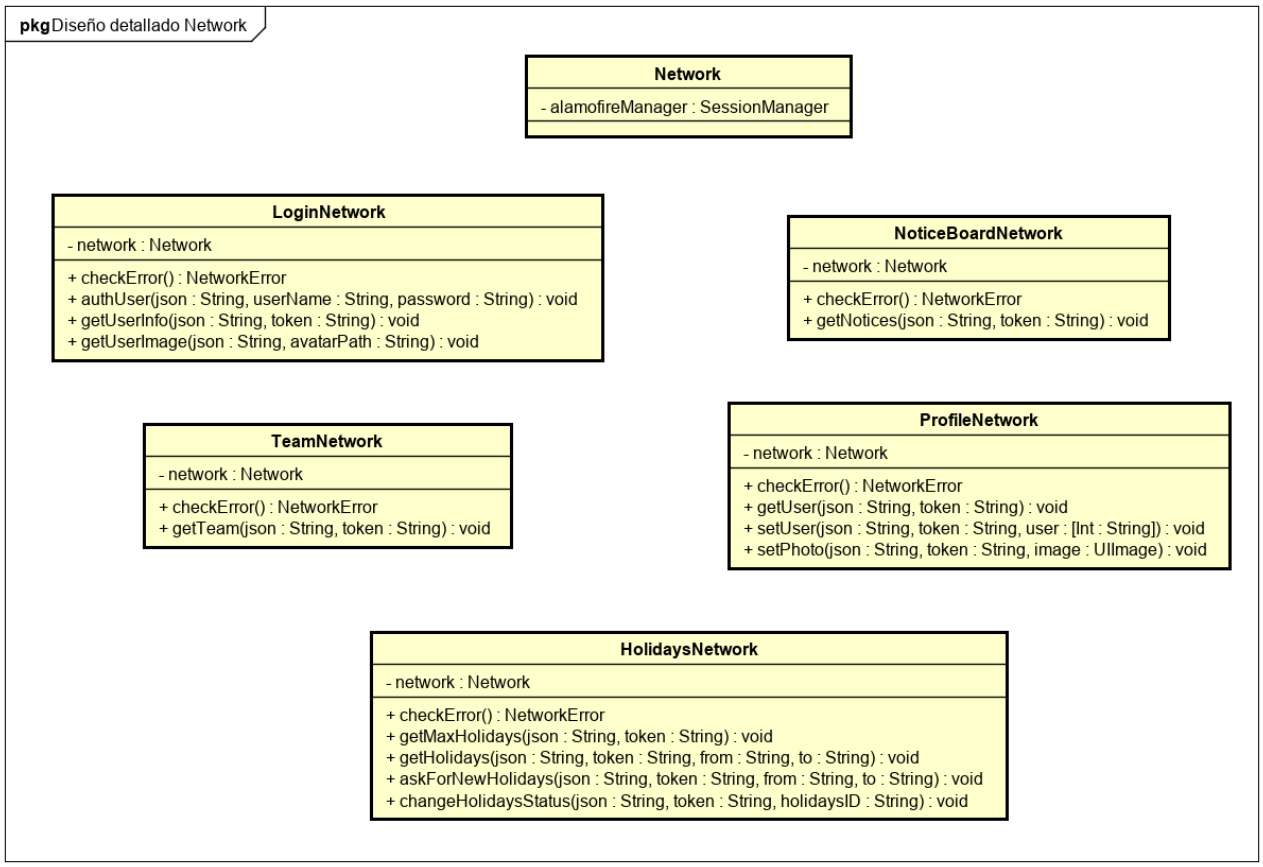

Figura 5.22: Diseño detallado rearquitectura, capa Network

5.3 Diseño incremento funcionalidad

5.3.1 Arquitectura general

La arquitectura que se va a usar para el incremento de la nueva funcionalidad es la misma que se ha ultilizado al realizar la rearquitectura de la aplicación existente. Es decir, utilizando los patrones MVVM y repositorio de la misma forma que se ha descrito en la Figura 5.1.

5.3.2 Descomposición modular

En esta sección no se va a representar la descomposición modular de toda la aplicación ya que sería repetitivo porque gran parte ya está descrito en la Figura 5.2. Se va a representar la descomposición modular de los nuevos elementos que son necesarios para esta nueva

funcionalidad, obviando aquellos ya descritos en la sección anterior. Las clases que aparecen son aquellas que han aparecido con las nuevas funcionalidades implementadas. En el caso de la la capa Model, no se ha añadido ninguna clase.

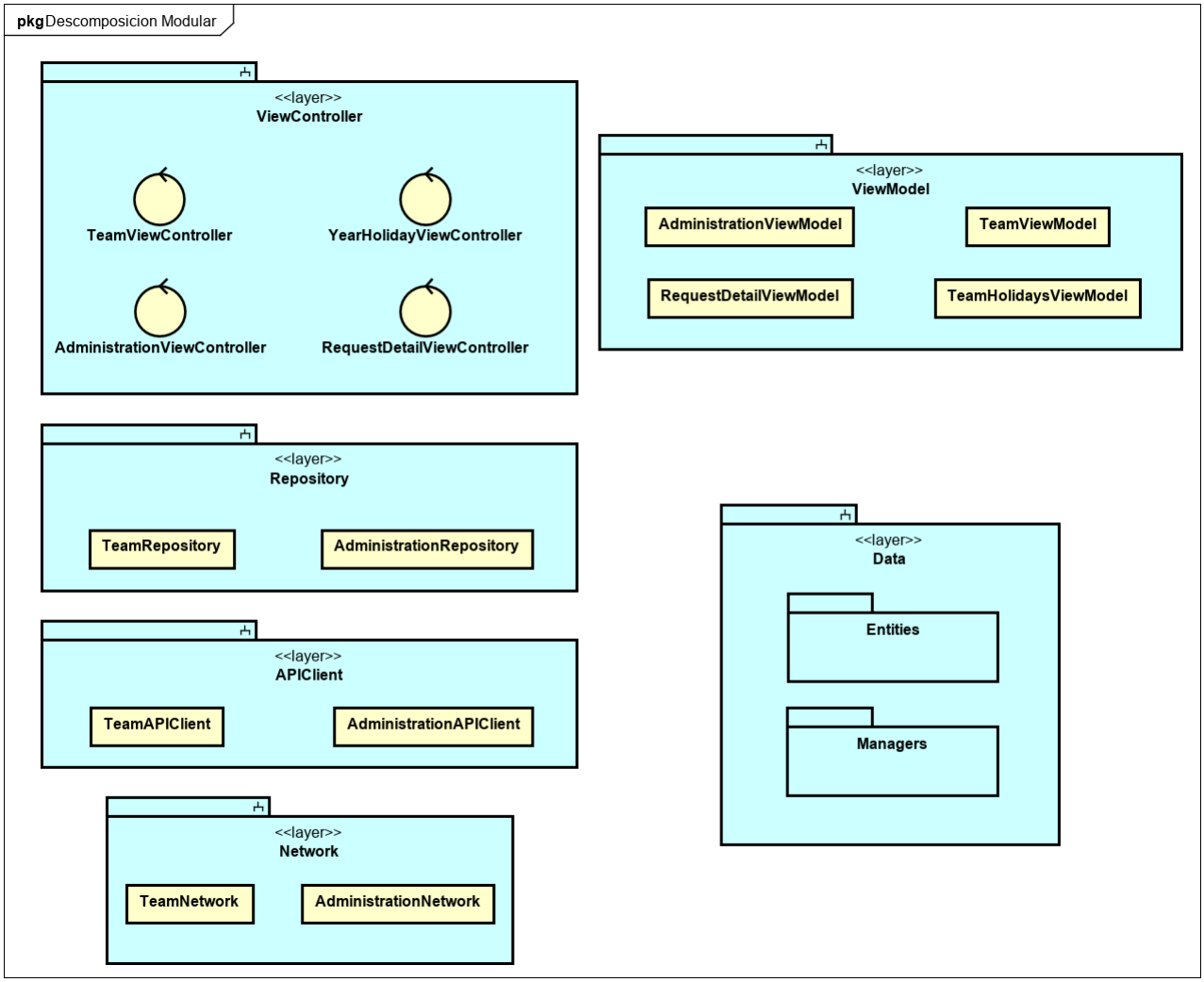

Figura 5.23: Descomposición modular del incremento de la funcionalidad

5.3.3 Diseño modular de cada capa

En este apartado únicamente vamos a mostrar las clases de cada capa que aparecen nuevas o han sido modificadas. Al aparecer una nueva sección de Administración, también son necesarias sus respectivas clases en cada capa para cumplir la arquitectura deseada. Al modificar la sección de Equipo, también se modifican sus clases de cada capa y aparecen nuevas (con la nueva interfaz).

5.3.3.1 Diseño modular capa ViewController

ViewController sección Administración. Para la interfaz de administración, se comunica las vacaciones de todos los empleados y las opciones de estado de cada solicitud. Para el detalle de una solicitud pendiente (RequestDetail) comunica los datos a mostrar de la solicitud y getiona las acciones de aceptar o rechazar la solicitud, decisión que comunica a su respectivo ViewModel.

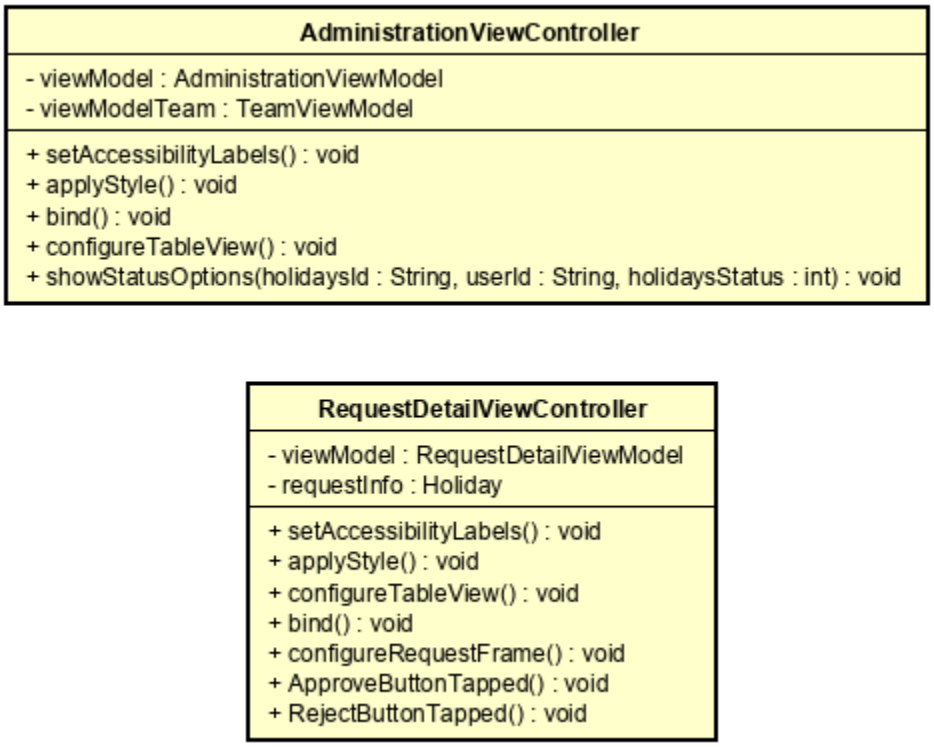

Figura 5.24: Diseño detallado capa ViewController, nueva sección de Administración

ViewController Equipo. En la clase TeamViewController aparece la responsabilidad de mostrar el número de trabajadores activos y mostrar las nuevas opciones al pulsar sobre un empleado. En el otro ViewController comunica los datos a mostrar en la vista: días máximos, historial de vacaciones, imagen del usuario...

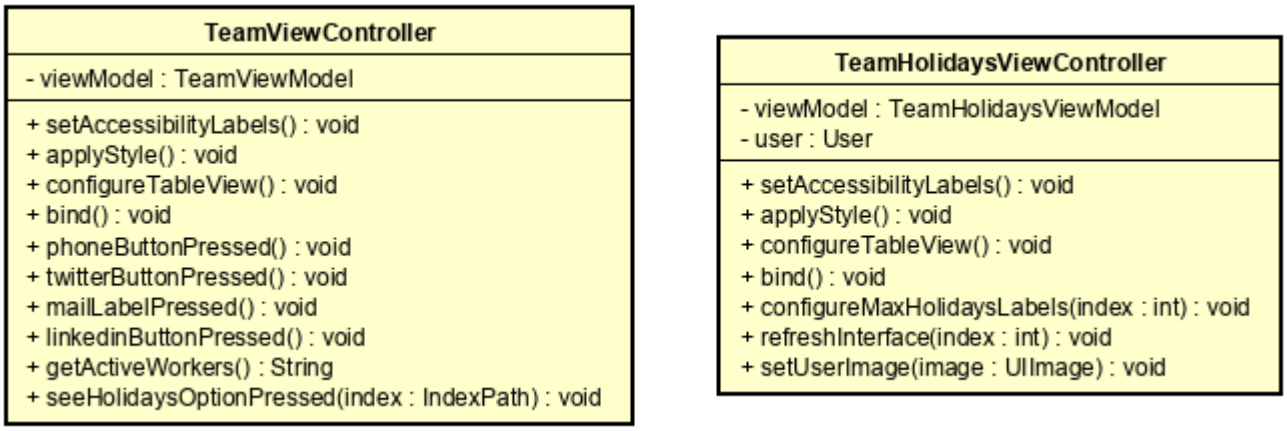

Figura 5.25: Diseño detallado capa ViewController, nueva sección de Equipo

5.3.3.2 Diseño modular capa ViewModel

En el siguiente diagrama podemos ver el diseño detallado de la capa ViewModel para la nueva sección de administración. En él podemos ver dos ViewModel: uno es para la pantalla principal de administración y el otro para el detalle de una solicitud. Se encargan de enviar los datos necesarios al ViewController: vacaciones, vacaciones máximas, usuario, mandar nuevo estado de una solicitud al Repositorio para su posterior guardado en el servidor...

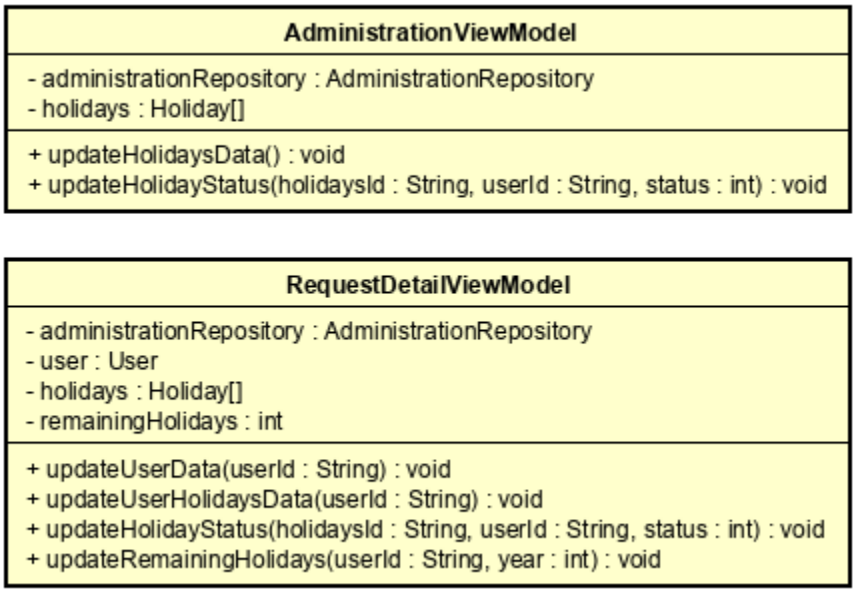

Figura 5.26: Diseño detallado capa ViewModel, nueva sección de Administración

ViewModelEquipo. Podemos ver que TeamViewModel se ha modificado para devolver el usuario por defecto a su ViewController. La otra clase debe facilitar las vacaciones y vacaciones máximas para todos los años en los que el usuario ha estado trabajando, así como su imagen de perfil.

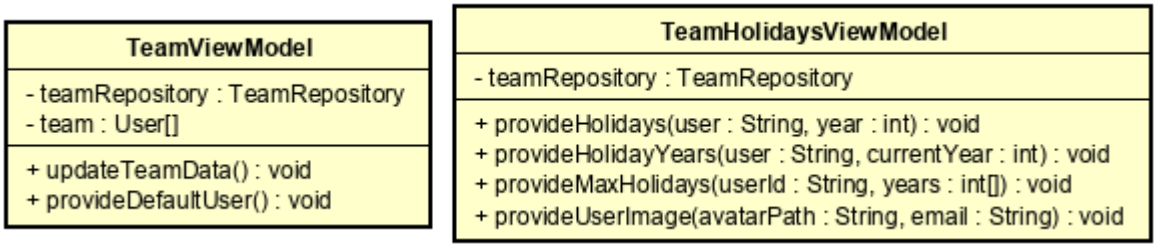

Figura 5.27: Diseño detallado capa ViewModel, nueva sección de Equipo

5.3.3.3 Diseño modular capa Repository

En el siguiente diagrama podemos ver cómo ha sido modificada la clase TeamRepository con la el incremento de la funcionalidad y la nueva clase: AdministrationRepository. Vemos

que en TeamRepository es necesrio almacenar en base de datos y recuperar nuevos datos: usuario por defecto, vacaciones del ususario, vacaciones máximas... En la nueva clase de administración se necesitan las vacaciones, el usuario, vacaciones restantes, actualizar el estado de una solicitud...

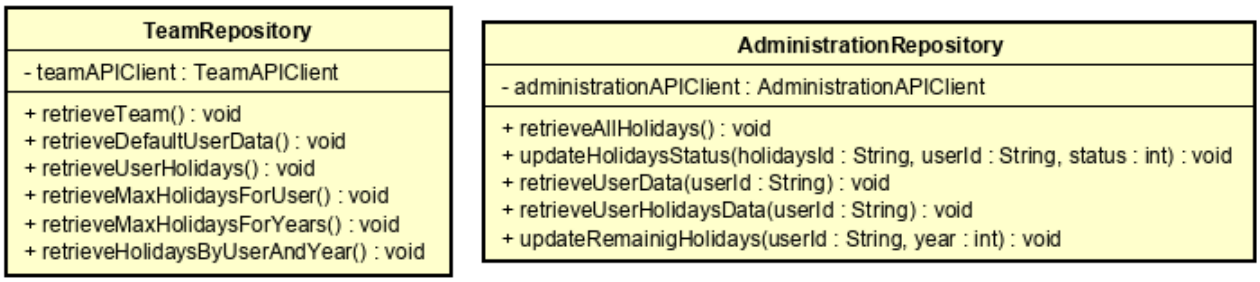

Figura 5.28: Diseño detallado capa Repository

5.3.3.4 Diseño modular capa APIClient

La siguiente figura muestra el diseño detallado de la capa APIClient. Como ya hemos descrito con aterioridad, esta capa se encarga de decodificar los datos que recibe de la capa Network (los mismos que después recibe el Repository).

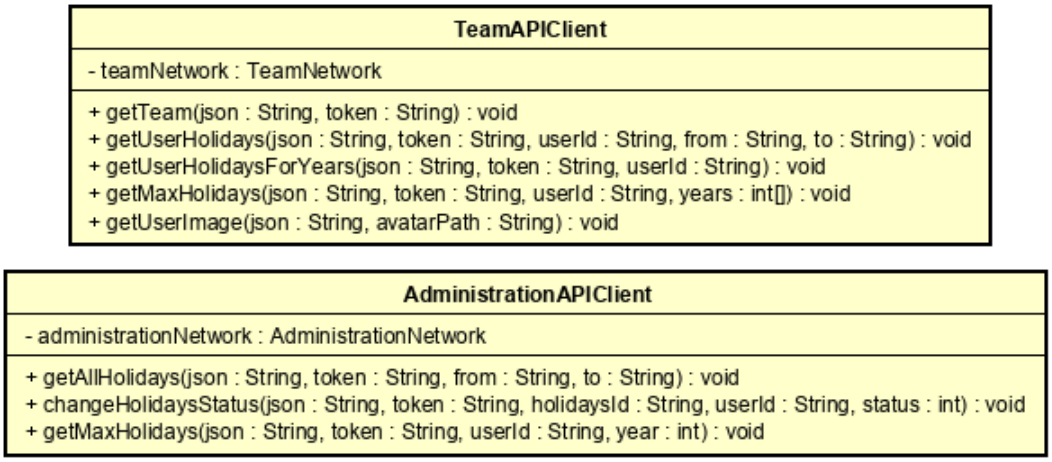

Figura 5.29: Diseño detallado capa APIClient

5.3.3.5 Diseño modular capa Network

La siguiente figura muestra el diseño detallado de la capa Network. Se encarga de recuperar del servidor los datos necesarios ya descritos en el resto de capas.

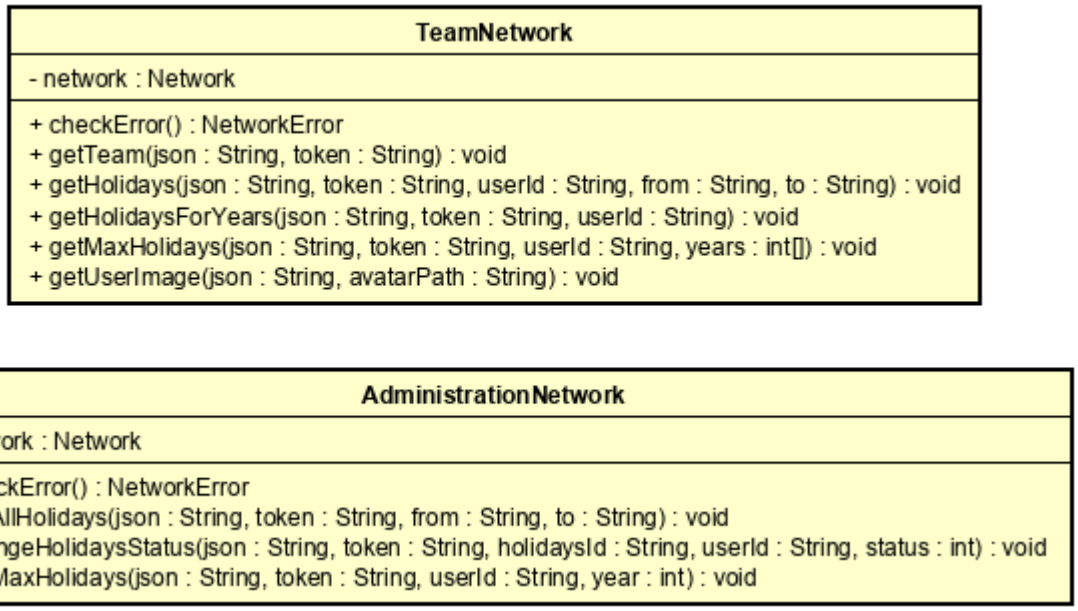

Figura 5.30: Diseño detallado capa Network

5.3.4 Relación entre capas

En este apartado se van a describir la relación entre las diferentes capas de la arquitectura de la aplicación para cada Historia de Usuario, de este apartado de incremento de la funcionalidad.

5.3.4.1 US25 - Ver solicitudes

En la siguiente figura podemos ver la relación entre las diferentes capas para la Historia de Usuario Ver solicitudes.

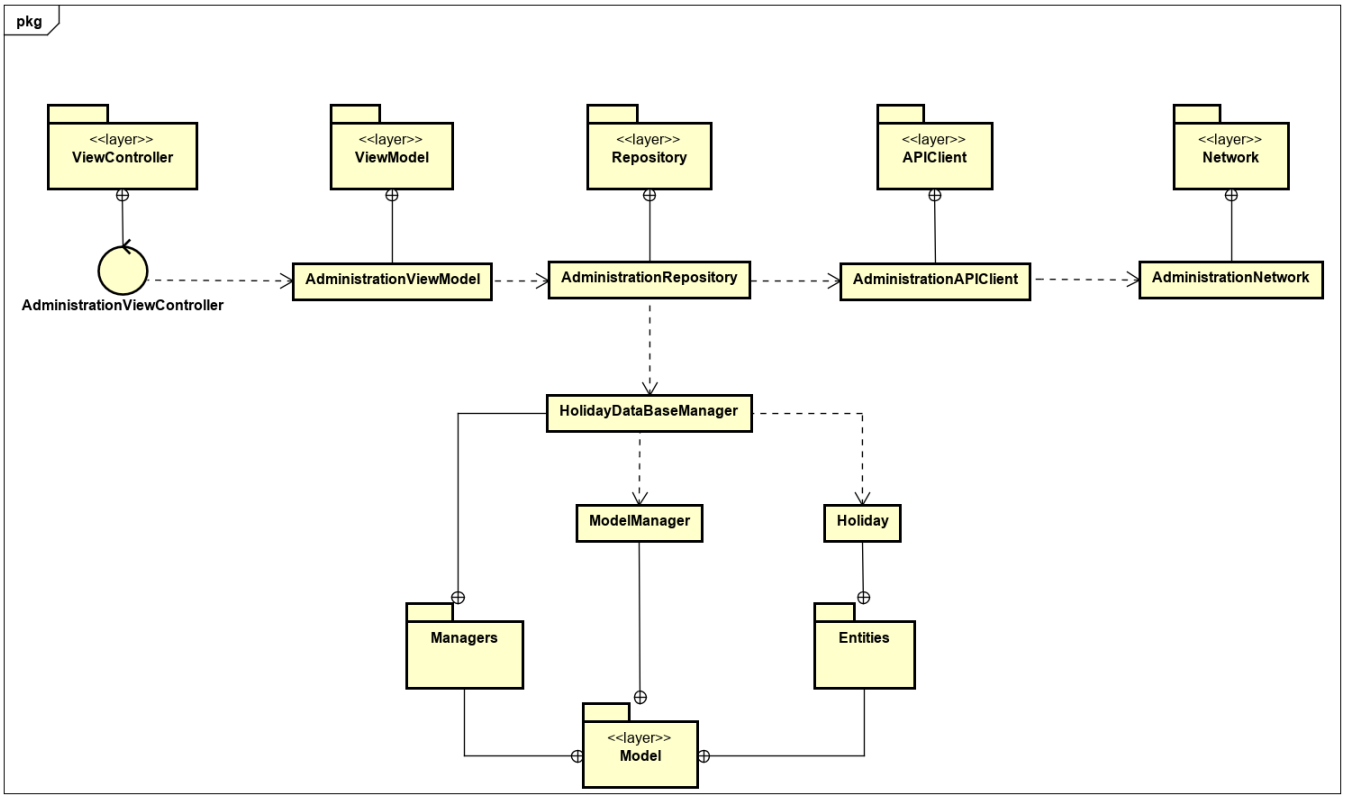

Figura 5.31: Relación entre capas US25 - Ver solicitudes

5.3.4.2 US26 - Gestión de vacaciones

En la siguiente figura podemos ver la relación entre las diferentes capas para la Historia de Usuario Gestionar vacaciones.

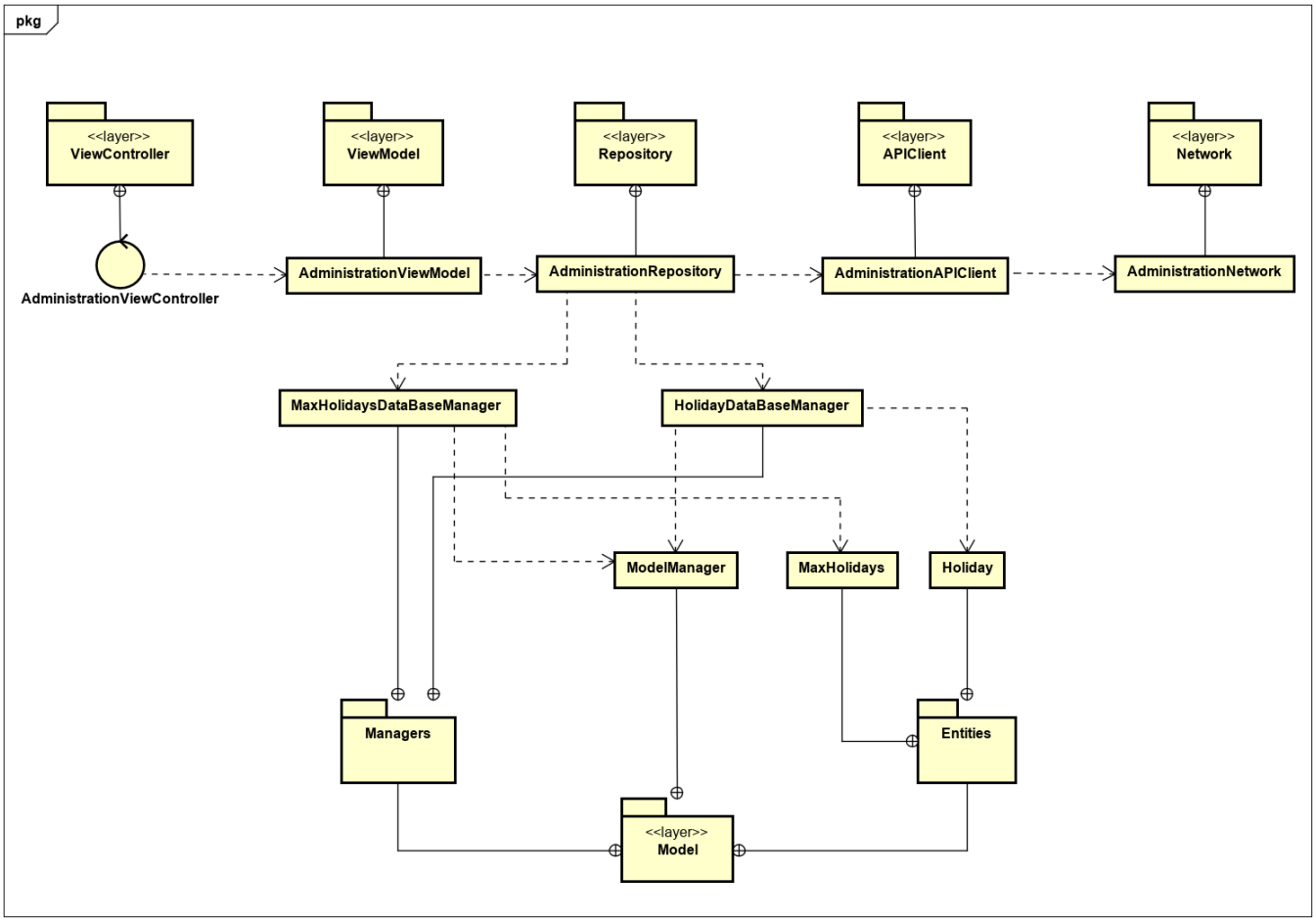

Figura 5.32: Relación entre capas US26 - Gestionar vacaciones

5.3.4.3 US27 - Historial usuario solicitud

En la siguiente figura podemos ver la relación entre las diferentes capas para la Historia de Usuario Historial usuario solicitudes.

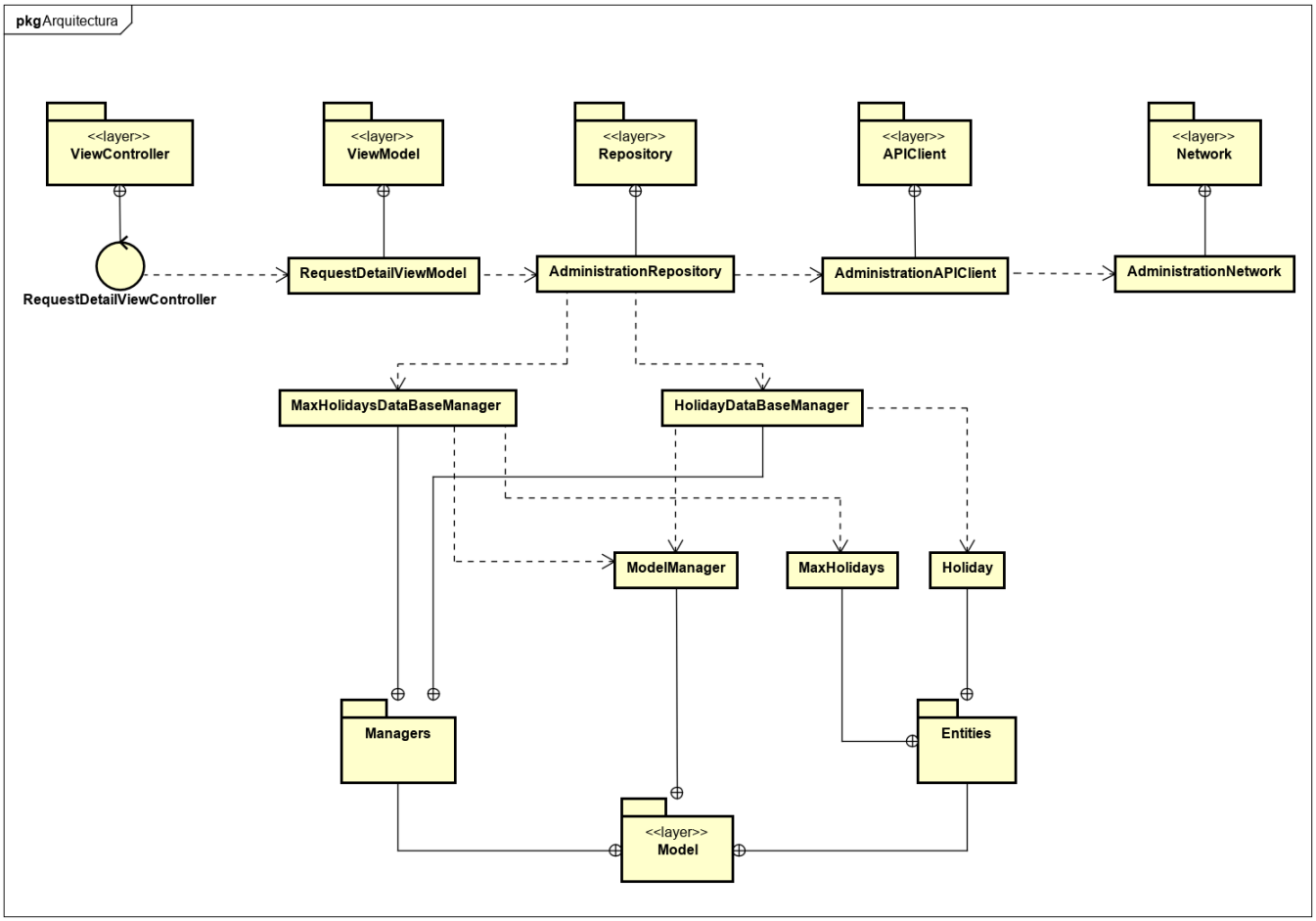

Figura 5.33: Relación entre capas US27 - Historial usuario solicitudes

5.3.4.4 US28 - Rediseño sección de Equipo

En la siguiente figura podemos ver la relación entre las diferentes capas para la Historia de Usuario Rediseño sección de Equipo.

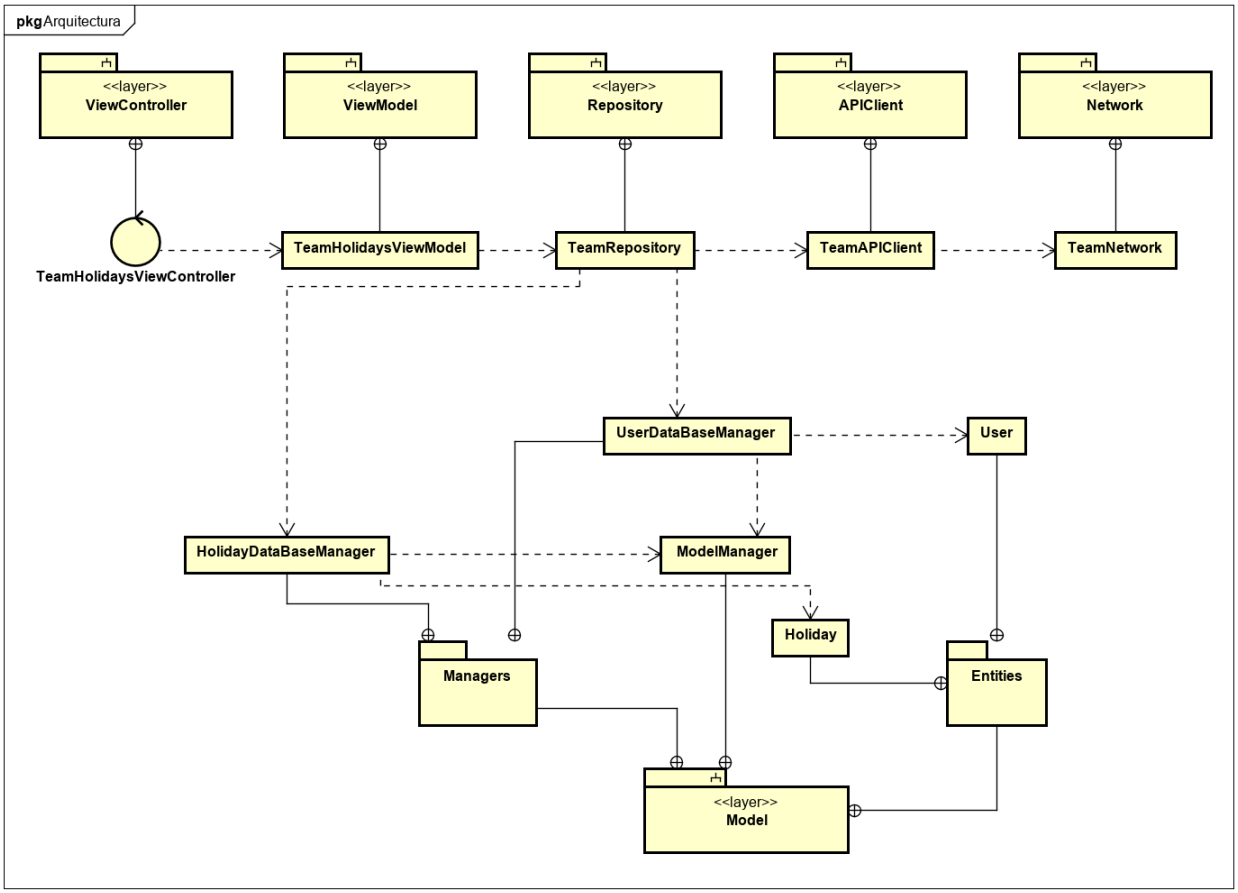

Figura 5.34: Relación entre capas US28 - Rediseño sección de Equipo

5.3.4.5 US29 - Ver vacaciones de cada usuario desde la sección de Equipo

En la siguiente figura podemos ver la relación entre las diferentes capas para la Historia de Usuario Historial vacaciones en Equipo.

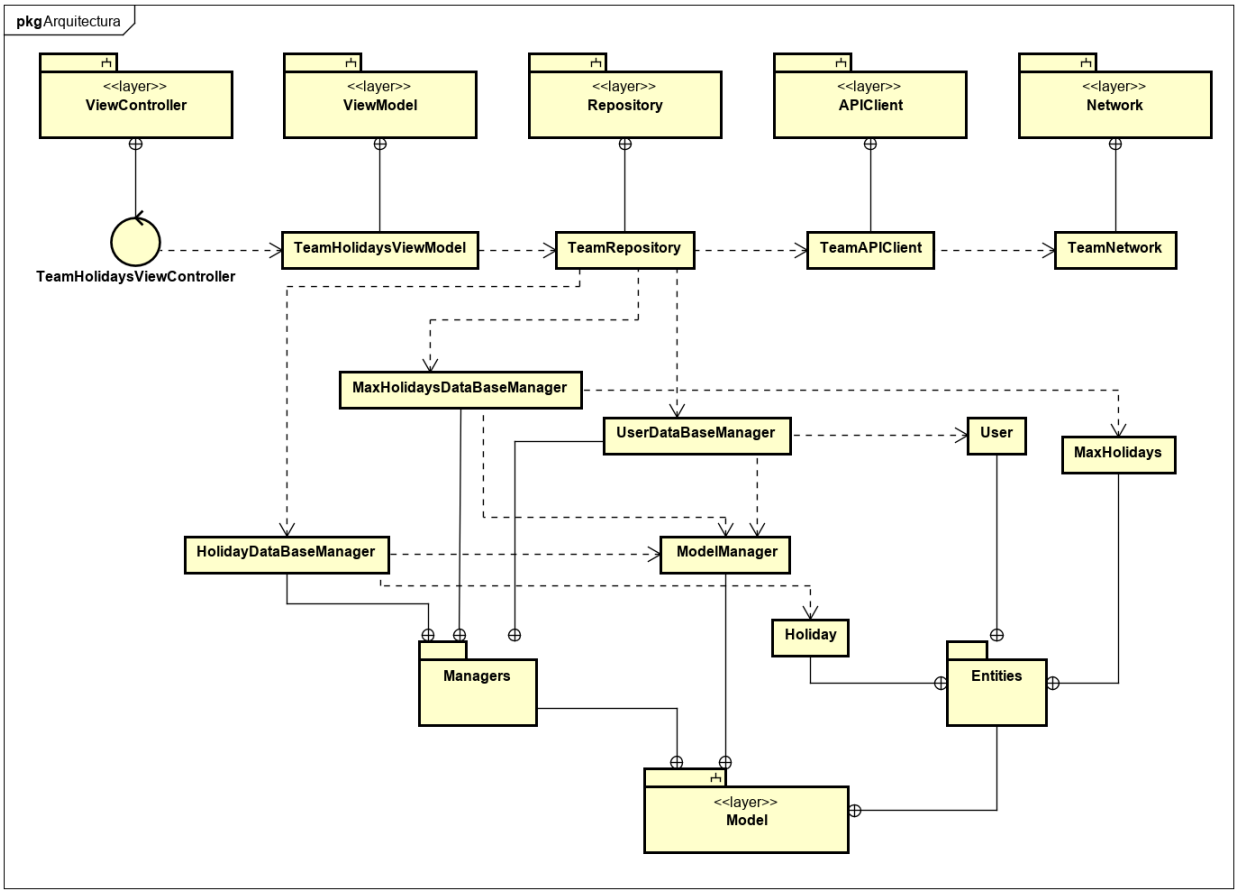

Figura 5.35: Relación entre capas US28 - Historial vacaciones en Equipo

5.3.5 Diseño de la interfaz

Para cada Historia de Usuario, se diseña un prototipo de interfaz que se utilizará a la hora de implementar dicha interfaz.

5.3.5.1 US25 - Ver solicitudes

Para esta Historia de Usuario, se ha diseñando el prototipo que se muestra a continuación. Aparecerán las solicitudes de vacaciones de todos los usuarios de la forma en que se ha representado en el prototipo. En esta Historia de Usuario la interfaz es únicamente para consulta, no requiere ninguna interacción con el usuario.

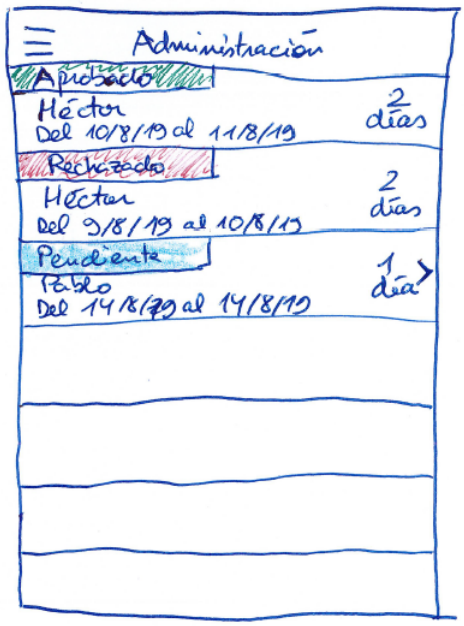

Figura 5.36: P01 - Prototipo interfaz US25 Ver solicitudes

5.3.5.2 US26 - Gestión de vacaciones

En el siguiente prototipo, al deslizar sobre la celda hacia la izquierda en una solicitud de Vacaciones con estado pendiente, aparecerán en la parte derecha de la celda dos opciones: aceptar y rechazar. Si la solicitud de vacaciones de la celda está en estado aprobado, al deslizar hacia la derecha sobre ella aparecerá la opción de cambiar el estado a pendiente.

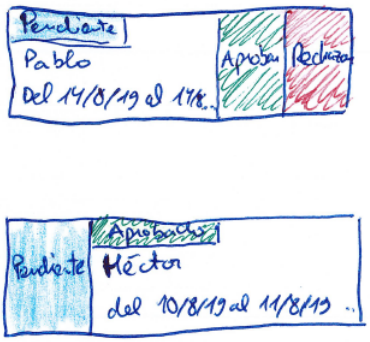

Figura 5.37: P02 - Prototipo interfaz US26 Gestión de Vacaciones

5.3.5.3 US27 - Historial usuario solicitud

Para esta Historia de Usuario, si se pulsa sobre una solucitud pendiente en la pantalla de Administración, aparecerá el siguiente prototipo. Se muestra el detalle de la solicitud pendiente pulsada, así como el resto de solicitudes del usuario en cuestión para el año en curso. Hay dos botones para aceptar o rechazar la solicitud. En la parte superior izquierda hay un botón de cancelar para volver a la pantalla anterior.

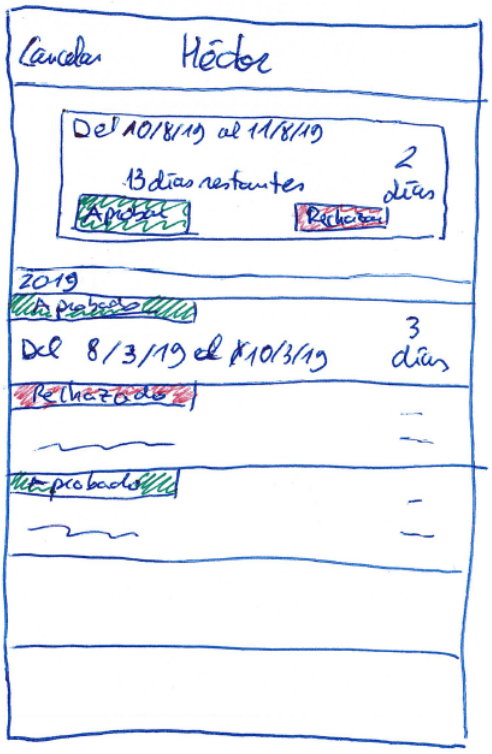

Figura 5.38: P03 - Prototipo interfaz US27 Historial usuario solicitud

5.3.5.4 US28 - Rediseño sección de Equipo

Para este rediseño de la interfaz se ha optado por el prototipo que se muestra a continuación. Aparecerán todos los trabajadores con su imagen de perfil, su posición en la empresa y su email. Además tendrán una banda de color verde o roja en la parte izquierda de la celda para distinguir aquellos trabajadores activos en la empresa de los que no lo son. En la parte inferior vemos el número de trabajadores totales y aquellos que están activos.

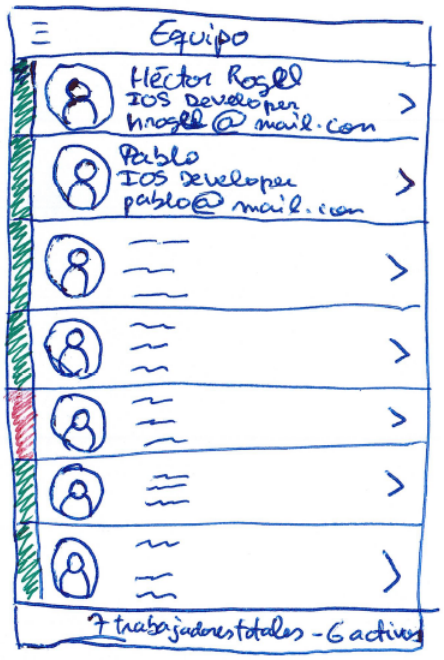

Figura 5.39: P04 - Prototipo interfaz US28 Rediseño sección de Equipo

El siguiente menú de opciones aparecerá al tocar sobre cualquier celda de trabajador de la interfaz anterior. Si el usuario identificado en la aplicación es Administrador, aparecerá la opción de las vacaciones. El resto de opciones aparecerán para todos los usuarios independientemente del rol que tengan.

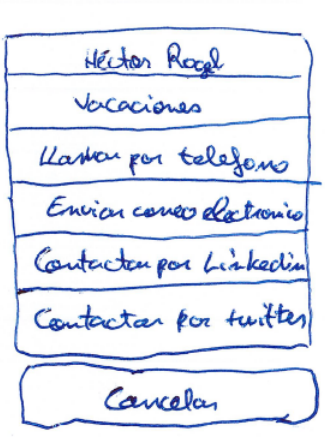

Figura 5.40: P05 - Prototipo interfaz US28 Rediseño sección de Equipo bis

5.3.5.5 US29 - Ver historial en sección de vacaciones

El administrador tendrá acceso a la siguiente interfaz de consulta de vacaciones del trabajador seleccionado. En ella se podrá ver la foto del trabajador en cuestión junto con su nombre y posición. También se podrá navegar entre los diferentes años para ver las solicitudes de vacaciones y los días de vacaciones máximos en cada año.

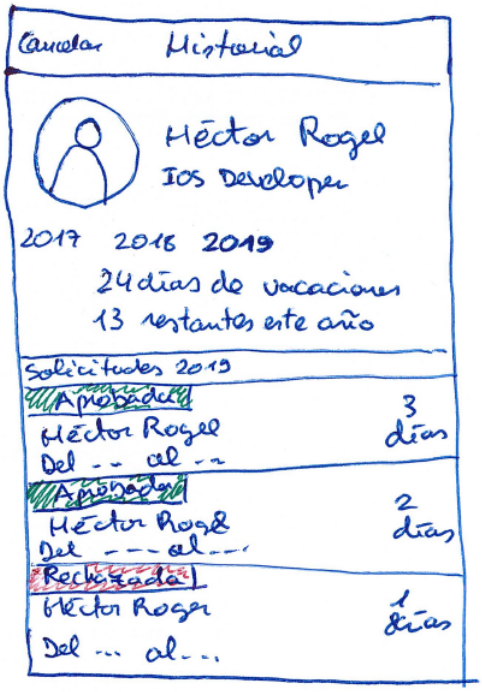

Figura 5.41: P06 - Prototipo interfaz US29 Ver historial en sección de vacaciones

5.3.6 Sprints

En esta sección se va a realizar un análisis exhaustivo de todos los Sprints del proyecto. Para cada sprint se describen las historias de usuario realizadas en cada uno de ellos con gran detalle:

- Estimación del esfuerzo necesario para llevar a cabo cada Historia de Usuario.
- Criterios de aceptación de la Historia de Usuario.
- Desglose detallado de las tareas para cada una de las Historias.
- Incremento del Sprint, dónde se describe las características añadidas al producto (basadas en las Historias de Usuario) y los problemas encontrados.

Esta sección tiene mucha importancia debido a que muestra con mayor precisión y detalle el trabajo desarrollado en este Trabajo de Fin de Grado de forma periódica, y los problemas que han podido surgir durante el mismo.

5.3.6.1 Sprint 0

Fecha de inicio: 11/02/2019
Fecha de fin: 17/02/2019
Sprint backlog:

Historia de Usuario		
Nombre: Instalación de recursos		
ID: 1	**Usuario:** Desarrollador	**Estimación:** 2
Descripción: Como Desarrollador quiero tener instalados todas las herramientas y tecnologías necesarias para la realización del proyecto.		
Criterios de Aceptación: - XCode (versión 10.1) instalado. - Simulator (versión 10.1) instalado. - Jira software configurado para comenzar con su uso. - Git instalado. - Gitlab configurado y preparado para usar. - Swift (versión 4.2) instalado.		

Tabla 5.1: US01 - Instalación de recursos

Estimación: Se ha decidido estimar esta historia con un 2 ya que sólo implica instalaciones. Podría haber sido estimada con un 1 pero no se ha hecho debido a que puede haber alguna complicación durnate la instalación, como por ejemplo incompatibilidades en el software.

Tareas:

- Instalar XCode y Simulator
- Crear cuenta en Jira software y configurarla para su uso inmediato
- Instalar git
- Crear cuenta en Gitlab y configurarla para su uso
- Instalar Swift

Historia de Usuario		
Nombre: Preparar documentación.		
ID: 2	**Usuario:** Desarrollador	**Estimación:** 5
Descripción: Como Desarrollador quiero tener configuradas y disponibles todas las herramientas de redacción de la documentación de este proyecto para poder completar dicha documentación durante el desarrollo del mismo.		
Criterios de Aceptación: - OverLeaf configurado y listo para su uso. - Conocer los conceptos báscicos de LaTeX. - La introducción de la documentación debe estar realizada. - El entorno tecnológico del proyecto debe estar redactado en la documentación.		

Tabla 5.2: US02 - Preparar documentación

Estimación: Se ha estimado con un 5 debido al desconocimiento de la tecnología LaTeX.

Tareas:

- Crear cuenta en OverLeaf
- Estudiar tutorial básico de LaTeX
- Redactar la introducción del Trabajo de Fin de Grado así como el entorno tecnológico que se va a usar para la realización del mismo.

Incremento Sprint: Todas las instalaciones han sido completadas con éxito. Se ha comenzado con la redacción de la documentación. A partir del siguiente Sprint las tareas de redacción de la documentación va a estar implícito dentro del mismo sin la necesidad de incluirlas en el mismo. Como excepción, sólo se incluirán tareas de documentación en el caso de que el sprint tenga únicamente historias de usuario de redacción de documentación.

5.3.6.2 Sprint 1

Fecha de inicio: 18/02/2019
Fecha de fin: 24/02/2019
Sprint backlog:

Historia de Usuario		
Nombre: Aprender swift		
ID: 3	**Usuario:** Desarrollador	**Estimación:** 5
Descripción: Como Desarrollador quiero ser capaz de dominar los aspectos básicos del lenguaje swift para poder programar aplicaciones iOS.		
Criterios de Aceptación: - Dominar la mayoría de estructuras de datos y tipos de swift. - Controlar las estructuras de control del lenguaje.		

Tabla 5.3: US03 - Aprender Swift

Estimación: Se ha estimado con un 5 debido a que es el primer contacto con Swift.

Tareas:

- Estudiar tutoriales de sintaxis básica de swift.
- Estudiar tutoriales de tipos de datos, variables, opcionales ...
- Estudiar tutoriales de estructuras de control, funciones, extensiones ...

Historia de Usuario		
Nombre: Primera aplicación iOS		
ID: 4	**Usuario:** Desarrollador	**Estimación:** 5
Descripción: Como Desarrollador quiero ser capaz de escribir una aplicación iOS básica.		
Criterios de Aceptación: - Construir una aplicación con un botón que realice alguna acción al ser pulsado. - La aplicación construida debe seguir la arquitectura Modelo Vista Controlador.		

Tabla 5.4: US04 - Primera aplicación iOS

Estimación: Se ha estimado con un 5 debido a que es la primera aplicación iOS y requiere aprendizaje.

Tareas:

- Aprender cómo funcionan los ViewController y los Storyboards en desarrollo iOS
- Estudiar tutoriales de cómo se aplica la arquitectura MVC en iOS.
- Construir una aplicación simple aplicando el modelo MVC.

Incremento sprint: Se han completado una cantidad considerable de tutoriales y se ha realizado una aplicación con la arquitectura MVC y que está compuesto por una pantalla con un botón que al ser pulsado muestra una imagen.

5.3.6.3 Sprint 2

Fecha de inicio: 25/02/2019
Fecha de fin: 03/03/2019
Sprint backlog:

Historia de Usuario		
Nombre: Clonación repositorio		
ID: 5	**Usuario:** Desarrollador	**Estimación:** 1
Descripción: Como Desarrollador quiero clonar el repositorio existente a mi máquina para poder desarrollar el proyecto.		
Criterios de Aceptación: - Clonar el código swift de la aplicación existente. - Clonar y configurar el backend ya existente. - Clonar y configurar el frontend ya existente.		

Tabla 5.5: US05 - Clonación repositorio

Estimación: Se ha estimado con un 1 debido a que es una tarea sencilla.

Tareas:

- Clonar de Gitlab el código de la aplicación existente.
- Descargar las librerías del proyecto y abrirlo con XCode.
- Clonar de Gitlab el backend y configurarlo para su uso.
- Clonar de Gitlab el frontend y configurarlo para su uso.

Historia de Usuario		
Nombre: Entender arquitectura		
ID: 6	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero entender la arquitectura de la aplicación.		
Criterios de Aceptación: - Comprender el funcionamiento de interfaz de la aplicación. - Comprender la arquitectura interna de la aplicación.		

Tabla 5.6: US06 - Entender arquitectura

Estimación: Se ha estimado con un 8 debido a que la aplicación tiene un tamaño considerable.

Tareas:

- Ejecutar la aplicación con Simulator para familiarizarse con ella.
- Entender cómo está estructurado el código internamente.
- Añadir una nueva sección a las ya existentes dentro de la aplicación, ya que esta tarea implica un conocimiento medio de la arquitectura de la aplicación.

Historia de Usuario		
Nombre: Entender Modelo Vista Vista Modelo (MVVM)		
ID: 7	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero entender la arquitectura MVVM para poder aplicarla en la aplicación.		
Criterios de Aceptación: - Comprender la estructura de MVVM. - Estudiar algunos casos prácticos. - Aplicar MVVM sobre la sección creada en el anterior sprint.		

Tabla 5.7: US07 - Entender Modelo Vista Vista Modelo

Estimación: Se ha estimado con un 8 debido a que es una parte muy importante de este Trabajo de Fin de Grado y hay que estudiarlo con detenimiento.

Tareas:

- Estudiar como funciona la arquitectura MVVM de forma teórica.
- Estudiar algunos ejemplos online de MVVM aplicado en desarrollo iOS.
- Modificar la sección creada en el anterior sprint aplicando la arquitectura MVVM.

Incremento sprint: Se ha clonado el repositorio y se ha creado una nueva sección en la aplicación. Se ha aplicado la arquitectura MVVM en la sección creada en el sprint anterior a modo de práctica.

5.3.6.4 Sprint 3

Fecha de inicio: 04/03/2019
Fecha de fin: 10/03/2019
Sprint backlog:

Historia de Usuario		
Nombre: Cambiar la vista de "Settings" a MVVM paso por paso.		
ID: 8	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección "Settings" a la arquitectura MVVM, paso por paso.		
Criterios de Aceptación: - La vista "Settings" sigue funcionando después del cambio a MVVM.		

Tabla 5.8: US08 - Cambiar la vista de "Settings" a MVVM paso por paso

Estimación: Se ha estimado con un 8 debido a que es el primer contacto real con la nueva arquitectura.

Tareas:

- Cambiar la interfaz de la Storyboard a un archivo xib.
- Cambiar la arquitectura de la sección (de MVC a MVVM).

Incremento sprint: La arquitectua de la sección "Settings" es MVVM.

5.3.6.5 Sprint 4

Fecha de inicio: 11/03/2019
Fecha de fin: 17/03/2019
Sprint backlog:

Historia de Usuario		
Nombre: Cambiar la vista de "Company" a MVVM paso por paso.		
ID: 9	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección de información acerca de la compañia a la arquitectura MVVM, paso por paso.		
Criterios de Aceptación: - La vista "Company" sigue funcionando después del cambio a MVVM.		

Tabla 5.9: US09 - Cambiar la vista de "Company" a MVVM paso por paso

Estimación: Se ha estimado con un 8 debido a que ya tengo algo de experiencia con la tarea, pese que es una sección más complicada que la anterior.

Tareas:

- Cambiar la interfaz de la Storyboard a un archivo xib.

- Cambiar la arquitectura de la sección (de MVC a MVVM).

Incremento sprint: La arquitectura de la sección "Company" es MVVM.

5.3.6.6 Sprint 5

Fecha de inicio: 18/03/2019
Fecha de fin: 24/03/2019
Sprint backlog:

Historia de Usuario		
Nombre: Cambiar la vista principal a MVVM paso por paso.		
ID: 10	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección principal (Twitter) a la arquitectura MVVM, paso por paso.		
Criterios de Aceptación: - La vista principal sigue funcionando después del cambio a MVVM.		

Tabla 5.10: US10 - Cambiar la vista principal a MVVM paso por paso

Estimación: Se ha estimado con un 8 debido a que pese a la experiencia en los anteriores sprints, las tareas son más complicadas y requiere actualizar librerías ya que se desconoce si van a seguir siendo compatibles después de dos años.

Tareas:

- Buscar y actualizar las librerías necesarias para esta sección.
- Cambiar la interfaz de la Storyboard a un archivo xib.
- Cambiar la arquitectura de la sección (de MVC a MVVM).

Incremento sprint: La arquitectua de la sección pricipal es MVVM. Se actualizaron las librerías sin grandes complicaciones, aunque la nueva versión implicó un cambio en el código (algunos métodos de la librería cambiaron).

5.3.6.7 Sprint 6

Fecha de inicio: 25/03/2019
Fecha de fin: 31/03/2019
Sprint backlog:

Historia de Usuario		
Nombre: Cambiar la vista del equipo a MVVM paso por paso.		
ID: 11	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección del equipo de la empresa a la arquitectura MVVM, paso por paso.		
Criterios de Aceptación: - La vista del equipo sigue funcionando después del cambio a MVVM.		

Tabla 5.11: US11 - Cambiar la vista del equipo a MVVM paso por paso

Estimación: Se ha estimado con un 8 debido a que es más complicada que la del anterior sprint, pero ya tengo más experiencia y requerir menos esfuerzo.

Tareas:

- Cambiar la interfaz de la Storyboard a un archivo xib.
- Cambiar la arquitectura de la sección (de MVC a MVVM).

Historia de Usuario		
Nombre: Cambiar la arquitectura de acceso a datos de la sección del equipo al patrón repositorio paso por paso.		
ID: 12	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero cambiar la arquitectura de acceso a datos de la sección del equipo de la empresa al patrón repositorio, paso por paso.		
Criterios de Aceptación: - La vista del equipo sigue funcionando después del cambio al patrón repositorio.		

Tabla 5.12: US12 - Cambiar la arquitectura de acceso a datos de la sección del equipo al patrón repositorio paso por paso

Estimación: Se ha estimado con un 8 debido a que es la primera vez que voy a realizar estos cambios.

Tareas:

- Aprender y estudiar ejemplos de patrón repositorio y de la forma de acceso a datos que vamos a utilizar.
- Separar responsabilidades entre el modelo, la vista del modelo y el repositorio.

Incremento sprint: La arquitectua de la sección del equipo es MVVM. La arquitectura de acceso a datos es el patrón repositorio. A la hora de separar las responsabilidades, hubo que reescribir gran parte del código para hacerlo funcional y compatible con la nueva arquitectura.

5.3.6.8 Sprint 7

Fecha de inicio: 01/04/2019
Fecha de fin: 07/04/2019
Sprint backlog:

Historia de Usuario		
Nombre: Añadir tests a la sección del equipo.		
ID: 13	**Usuario:** Desarrollador	**Estimación:** 8
Descripción: Como Desarrollador quiero añadir tests a la sección del equipo, en tres niveles: de interfaz de usuario, unitarios y de red.		
Criterios de Aceptación: - Todas las posibles respuestas del servidor han sido testeadas. - Tests de APIClient comprueban el funcionamiento de los módulos con diferentes tipos de JSON. - Tests de interfaz de usuario.		

Tabla 5.13: US13 - Añadir tests a la sección del equipo

Estimación: Se ha estimado con un 8 debido a que es la primera vez que voy a realizar estos tests.

Tareas:

- Aprender y estudiar ejemplos de tests en tres niveles.
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios.
- Realizar tests de red.

Incremento sprint: Se han implementado los tests en tres niveles, los cuáles han funcionado correctamente y no han detectado ningún error en el código testeado de la aplicación.

5.3.6.9 Sprint 8

Fecha de inicio: 08/04/2019
Fecha de fin: 22/04/2019
Sprint backlog:

Historia de Usuario		
Nombre: Rearquitectura sección del Perfil de usuario y tests		
ID: 14	**Usuario:** Desarrollador	**Estimación:** 13
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección Perfil y aplicar tests en tres niveles.		
Criterios de Aceptación: - La sección de Perfil sigue funcionando correctamente después del cambio a MVVM. - La sección de Perfil sigue funcionando correctamente después del cambio al patrón repositorio. - Tests de interfaz de usuario, unitarios y de red.		

Tabla 5.14: US14 - Rearquitectura sección del Perfil de usuario y tests

Estimación: Se ha estimado con un 13 debido a que esta historia de usuario se compone de lo realizado en los tres sprints anteriores (para una sección diferente).

Tareas:

- Cambiar la interfaz de la Storyboard a un archivo xib.
- Cambiar la arquitectura de la sección (de MVC a MVVM).
- Cambiar el acceso a datos a patrón repositorio.
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios.
- Realizar tests de red.

Incremento sprint: Se ha cambiado la sección Perfil a la arquitectura MVVM y al patrón repositorio y se han aplicado los tests en tres niveles para las dos partes de la sección: perfil y editar perfil. Este sprint ha tenido grandes dificultades debido a las dimensiones del mismo y que gran parte del código ha tenido que ser reescrito para cumplir las características del patrón repositorio. Debido a estas dificultades, el sprint se tuvo que alargar una semana más.

5.3.6.10 Sprint 9

Fecha de inicio: 22/04/2019
Fecha de fin: 05/05/2019
Sprint backlog:

Historia de Usuario		
Nombre: Rearquitectura sección de Vacaciones y tests		
ID: 15	**Usuario:** Desarrollador	**Estimación:** 13
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección Vacaciones y aplicar tests en tres niveles.		
Criterios de Aceptación: - La sección de Vacaciones sigue funcionando correctamente después del cambio a MVVM. - La sección de Vacaciones sigue funcionando correctamente después del cambio al patrón repositorio. - Tests de interfaz de usuario, unitarios y de red.		

Tabla 5.15: US15 - Rearquitectura sección de Vacaciones y tests

Estimación: Se ha estimado con un 13 debido a que esta historia de usuario se compone de lo realizado en los tres sprints anteriores (para una sección diferente).

Tareas:

- Cambiar la interfaz de la Storyboard a un archivo xib.
- Cambiar la arquitectura de la sección (de MVC a MVVM).
- Cambiar el acceso a datos a patrón repositorio.
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios.
- Realizar tests de red.

Incremento sprint: Se ha cambiado la sección de vacaciones a la arquitectura MVVM y al patrón repositorio y se han realizado tests en tres niveles para esta sección. Este sprint también ha tenido retraso debido a la dificultad, por lo que también se ha ampliado a dos semanas.

5.3.6.11 Sprint 10

Fecha de inicio: 06/05/2019
Fecha de fin: 12/05/2019
Sprint backlog:

Bug
Nombre: Pérdida de los datos de vacaciones cuando se cambia de año.
ID: 16 | **Usuario:** Desarrollador
Descripción: En la sección de vacaciones, cuando se cambia de un año a otro, los datos de las vacaciones se ponen en valor nulo y provoca que la aplicación se cierre de forma inesperada.

Tabla 5.16: BUG - Pérdida de los datos de vacaciones cuando se cambia de año

Bug
Nombre: En ocasiones las vacaciones máximas no se muestran cuando se accede a la sección de Vacaciones.
ID: 17 | **Usuario:** Desarrollador
Descripción: Al acceder a la sección de vacaciones, el número de días máximo de vacaciones no se muestra siempre

Tabla 5.17: BUG - En ocasiones las vacaciones máximas no se muestran cuando se accede a la sección de Vacaciones

Bug
Nombre: Después de añadir una solicitud de vacaciones, las vacaciones del usuario se muestran duplicadas.
ID: 18 | **Usuario:** Desarrollador
Descripción: Después de realizar una solicitud de vacaciones, cuando visualizamos nuestras vacaciones, estas se muestran duplicadas.

Tabla 5.18: BUG - Después de añadir una solicitud de vacaciones, las vacaciones del usuario se muestran duplicadas

Historia de Usuario		
Nombre: Actualizar código a swift 4.2.		
ID: 19	**Usuario:** Desarrollador	**Estimación:** 3
Descripción: Como Desarrollador quiero actualizar el código a swift 4.2.		
Criterios de Aceptación: - La aplicación completa compila en swift 4.2 y su funcionalidad es la misma que antes del cambio.		

Tabla 5.19: US19 - Actualizar código a swift 4.2

Estimación: Se ha estimado con un 3 debido a que, pese a que debería ser una tarea sencilla, puede tener incompatibilidades que retrasen la tarea.

Tareas:

- Actualizar las versiones de las librerías de terceros utilizadas.
- Actualizar el código a swift 4.
- Actualizar a swift 4.2.

Historia de Usuario		
Nombre: Rearquitectura sección de Avisos y tests.		
ID: 20	**Usuario:** Desarrollador	**Estimación:** 1
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección de Avisos.		
Criterios de Aceptación: - La sección de Avisos sigue funcionando correctamente después del cambio a MVVM. - La sección de Avisos sigue funcionando correctamente después del cambio al patrón repositorio. - Tests de interfaz de usuario, unitarios y de red.		

Tabla 5.20: US20 - Rearquitectura sección de Avisos y tests

Estimación: Se ha estimado con un 1 debido a que es una tarea sencilla.

Tareas:

- Cambiar la interfaz de la Storyboard a un archivo xib.
- Cambiar la arquitectura de la sección (de MVC a MVVM).
- Cambiar el acceso a datos a patrón repositorio.
- Realizar tests de interfaz de usuario.

- Realizar tests unitarios.
- Realizar tests de red.

Historia de Usuario		
Nombre: Rearquitectura sección de Login y tests.		
ID: 21	**Usuario:** Desarrollador	**Estimación:** 3
Descripción: Como Desarrollador quiero cambiar la arquitectura de la sección de Login.		
Criterios de Aceptación: - La sección de Login sigue funcionando correctamente después del cambio a MVVM. - La sección de Login sigue funcionando correctamente después del cambio al patrón repositorio. - Tests de interfaz de usuario, unitarios y de red.		

Tabla 5.21: US21 - Rearquitectura sección de Login y tests

Estimación: Se ha estimado con un 1 debido a que es una tarea de dificultad media - baja.

Tareas:

- Cambiar la interfaz de la Storyboard a un archivo xib.
- Cambiar la arquitectura de la sección (de MVC a MVVM).
- Cambiar el acceso a datos a patrón repositorio.
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios
- Realizar tests de red

Bug	
Nombre: El año por defecto en la sección de vacaciones debe ser el actual.	
ID: 22	**Usuario:** Desarrollador
Descripción: Cuando se accede a la sección de vacaciones, la pestaña por defecto para las vacaciones es 2018 y debería de mostrarse 2019 (el año actual).	

Tabla 5.22: BUG - El año por defecto en la sección de vacaciones debe ser el actual

Bug
Nombre: Datos de vacaciones restantes y máximas no se recargan en el primer acceso.
ID: 23 | **Usuario:** Desarrollador
Descripción: Cuando se inicia la aplicación, la primera vez que se accede a la sección de vacaciones, no se cargan el número de dias de vacaciones máximas y restantes.

Tabla 5.23: BUG - Datos de vacaciones restantes y máximas no se recargan en el primer acceso

Bug
Nombre: Después de modificar la imagen de perfil, esta no se recarga en la aplicación.
ID: 24 | **Usuario:** Desarrollador
Descripción: Cuando se accede a la sección de perfil y se cambia la imagen del usuario, esta nueva imagen no se recarga en el resto de la aplicación.

Tabla 5.24: BUG - Después de modificar la imagen de perfil, esta no se recarga en la aplicación

Incremento sprint: Se han resuelto todos los bugs con éxito, rearquitectura de las secciones de avisos y login con éxito, así como sus respectivos tests. El código de la aplicación está en la versión 4.2 de swift y no se puede pasar a swift 5, debido a incompatibilidades con librerías de terceros.

5.3.6.12 Sprint 11

Fecha de inicio: 03/06/2019
Fecha de fin: 09/06/2019
Sprint backlog:

Historia de Usuario
Nombre: Ver solicitudes pendientes en nueva sección.
ID: 25 | **Usuario:** Administrador | **Estimación:** 8
Descripción: Como Administrador quiero ver las solicitudes de vacaciones pendientes en una nueva sección.
Criterios de Aceptación: - La nueva sección de administración sólo aparece si el usuario es administrador. - Sólo aparecen las solicitudes de vacaciones pendientes. - Tests de interfaz de usuario, unitarios y de red.

Tabla 5.25: US25 - Ver solicitudes pendientes en nueva sección

Estimación: Se ha estimado con un 8 debido a que esta historia de usuario supone cambiar el funcionamiento interno de la aplicación para añadir tipos de usuarios y la nueva sección, a parte de la fase de diseño de esta sección.

Tareas:

- Hacer un prototipo de la nueva interfaz.
- Cambiar el modelo para que los usuarios tengan un atributo rol.
- Crear nueva sección en el menú lateral.
- La nueva sección aparece en función del rol del usuario identificado.
- Implementar la interfaz de la nueva sección.
- Obtener todos los datos necesarios del servidor usando el patrón repositorio.
- Mostrar los datos en la interfaz usando MVVM.
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios.
- Realizar tests de red.

Incremento sprint: Sólo los usuarios administradores pueden acceder a la nueva sección para consultar las solicitudes, aquellos que no tienen rol de administrador no pueden ver esta sección. Al acceder a la nueva sección, aparecen las solicitudes de vacaciones pendientes (no aparecen las aceptadas o rechazadas).

5.3.6.13 Sprint 12

Fecha de inicio: 10/06/2019
Fecha de fin: 16/06/2019
Sprint backlog:

Historia de Usuario		
Nombre: Aprobar o rechazar solicitudes de vacaciones.		
ID: 26	**Usuario:** Administrador	**Estimación:** 8
Descripción: Como Administrador quiero aprobar o rechazar las solicitudes de vacaciones pendientes.		
Criterios de Aceptación: - El administrador deslizará sobre una solicitud hacia la izquierda para aceptar o rechazar. - El administrador deslizará sobre una solicitud, con estado de aceptada, hacia la derecha para cambiar el estado a pendiente. - Las solicitudes con estado cancelado no pueden volver a cambiar de estado.		

Tabla 5.26: US26 - Aprobar o rechazar solicitudes de vacaciones

Estimación: Se ha estimado con un 5 debido a que esta historia de usuario supone cambiar la interfaz del sprint anterior y añadir el "deslizamiento" en cada solicitud.

Tareas:

- Hacer un prototipo de la nueva interfaz.
- Cambiar la interfaz de la sección para añadir el deslizamiento.
- Actualizar el cambio de estado en el servidor al aceptar o rechazar una solicitud.
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios.
- Realizar tests de red.

Incremento sprint: La interfaz ha sido modificada para cumplir los nuevos requisitos. Al deslizar hacia la izquierda o la derecha en una solicitud, aparecen en la celda unas opciones: aceptar, rechazar o pasar a pendiente, según el estado de la solicitud. Ese cambio de estado se guarda en el servidor y se actualiza la interfaz con el cambio.

5.3.6.14 Sprint 13

Fecha de inicio: 17/06/2019
Fecha de fin: 23/06/2019
Sprint backlog:

Historia de Usuario		
Nombre: Ver historial de vacaciones del usuario cuando se va a aceptar o rechazar la solicitud.		
ID: 27	**Usuario:** Administrador	**Estimación:** 8
Descripción: Como Administrador quiero ver el historial de vacaciones del usuario mientras acepto o rechazo una solicitud.		
Criterios de Aceptación: - El administrador accederá a un detalle de la solicitud al pulsar sobre la misma. - El administrador verá todas la solicitudes acpetadas o rechazadas mientras que acepta la misma. - Aparecerán el número de días de vacaciones restantes para ese trabajador.		

Tabla 5.27: US27 - Ver historial de vacaciones del usuario cuando se va a aceptar o rechazar la solicitud

Estimación: Se ha estimado con un 8 debido a que esta historia de usuario supone añadir una nueva interfaz y nueva funcionalidad.

Tareas:

- Hacer que las celdas de la interfaz admitan interacción con el usuario, crear estructura para nueva interfaz.
- Hacer un prototipo de la nueva interfaz.
- Implementar nueva interfaz.
- Obtener datos del servidor usando el patrón Repositorio
- Mostrar los datos en la interfaz usando MVVM
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios.
- Realizar tests de red.

Incremento sprint: La interfaz ha sido modificada para cumplir los nuevos requisitos. Al presionar sobre una celda "pendiente" aparece una nueva interfaz con el detalle de la solicitud y una lista con el resto de solicitudes del trabajador en cuestión. También aparecen el número de días de vacaciones restantes y dos botones para aceptar o rechazar.

5.3.6.15 Sprint 14

Fecha de inicio: 24/06/2019
Fecha de fin: 30/06/2019
Sprint backlog:

Historia de Usuario		
Nombre: Rediseño sección de Equipo.		
ID: 28	**Usuario:** Desarrollador	**Estimación:** 3
Descripción: Como Desarrollador quiero implementar un nuevo diseño de la sección Equipo de acuerdo a un prototipo de este nuevo diseño.		
Criterios de Aceptación: - La nueva interfaz cumple los criterios del prototipo. - La nueva interfaz mantiene la funcionalidad anterior al cambio.		

Tabla 5.28: US28 - Rediseño sección de Equipo

Estimación: Se ha estimado con un 3 debido a que esta historia de usuario es sencilla y únicamente requiere cambiar la interfaz y mantener la funcionalidad.

Tareas:

- Hacer un prototipo de la nueva interfaz.
- Implementar nueva interfaz.
- Asegurarse de que la funcionalidad se mantiene.

Historia de Usuario		
Nombre: Ver vacaciones de cada usuario desde la sección de Equipo.		
ID: 29	**Usuario:** Administrador	**Estimación:** 8
Descripción: Como Administrador quiero, desde la sección de Equipo, consultar el historial de vacaciones de cada trabajador.		
Criterios de Aceptación: - La nueva interfaz cumple los criterios del prototipo. - Implementar nueva interfaz. - Obtener datos del servidor usando el patrón Repositorio.		

Tabla 5.29: US29 - Ver vacaciones de cada usuario desde la sección de Equipo

Estimación: Se ha estimado con un 8 debido a que esta historia de usuario requiere crear una nueva interfaz y una nueva funcionalidad.

Tareas:

- Hacer un prototipo de la nueva interfaz.
- Implementar nueva interfaz.
- Mostrar los datos en la interfaz usando MVVM
- Realizar tests de interfaz de usuario.
- Realizar tests unitarios.
- Realizar tests de red.

Incremento sprint: Se ha lleva a cabo el rediseño sin ningún problema. La nueva interfaz permite consultar el historial de vacaciones de todos los trabajadores de la empresa, incluso aquellos que no están activos en la empresa.

Capítulo 6

Implementación y tests

6.1 Cambio del software existente

6.1.1 Reestructuración interfaces

Como se ha analizado anteriormente, las interfaces de la aplicación necesitaban una reestructuración. Esta reestructuración tiene como objetivo la mejora de la eficiencia de las interfaces. Para lograr esto, hay que volver a crear las interfaces ya existentes pero en archivos separados (ficheros con extensión xib).

El procedimiento para realizar esta tarea es el siguiente (para cada sección):

- Analizar en profundidad el funcionamiento de la interfaz en cuestión.
- Realizar la misma interfaz en un fichero xib, o varios ya que cada interfaz de la sección requiere varios ficheros, conservando todas sus características y dependencias con otras interfaces.
- Comprobar que, tras la reeestructuración, las nuevas interfaces se comporten de la misma forma que antes del cambio.

6.1.2 Rearquitectura

Como ya se ha indicado en varias ocasiones, hay que cambiar la arquitectura completa de la aplicación al modelo MVVM. Esta rearquitectura se lleva a cabo en varias partes (como podemos ver en la sección 4.4), cada parte es una de las secciones de la aplicación.

Para cada una de estas secciones, el trabajo de rearquitectura se divide en dos partes: cambio a la arquitectura MVVM y cambio del modo de acceso a datos.

6.1.2.1 Cambio a MVVM

Como hemos visto en el Capítulo 4, apartado 4.1, cada sección de la aplicación se compone de uno o varios controladores. El cambio de arquitectura consiste en eliminar responsabilidades del ViewController y añadirlas al ViewModel. Este cambio se lleva a cabo en los siguientes pasos:

- Analizar y comprender el funcionamiento del controlador sobre el que se va a realizar la rearquitectura.
- Identificar responsabilidades: la lógica que se encarga de modificar los elementos de la vista.
- Mover dicha lógica al ViewModel, para que sea esta la que decida cuándo se muestran o no los datos en la interfaz.

En resumen, todos los elementos de la vista que requieran algo de lógica para mostrarse o no en la vista, hay que moverlos al ViewModel.

Pero esta tarea no supone únicamente mover código de una clase a otra. En la mayoría de casos, era necesario añadir nuevo código y modificar el código existente para adecuarlo a la nueva arquitectura.

6.1.2.2 Cambio patrón de acceso a datos

La siguiente tarea a realizar en la rearquitectura es el cambio del patrón de acceso a datos. Queremos una arquitectura aún más desacoplada para poder implementar los tests. El patrón Repositorio, que hemos definido en la sección 5.5.5, nos brinda esta posibilidad.

Para esta tarea tenemos que desacoplar lo máximo posible el código ya existente. Dicho desacoplamiento se hace en las clases ya mencionadas: Repository, APIClient y Network. También añadimos la clase Router aunque no es muy importante ya que esta parte no se va a testear, pero forma parte también del desacoplamiento. Esta clase Router se encarga de construir la URL de cada petición HTTP al servidor. Esto es importante, ya que nos permite tener esta funcionalidad aislada de forma que son más sencillas las funciones de modificación y depuración del código.

Esta tarea no requiere únicamente mover código de unas clases a otras, también existe un gran trabajo de modificación del código existente (a parte del desacoplamiento) para adecuarlo al nuevo patrón. Esto es debido a que, pese a que la funcionalidad es la misma que con el nuevo patrón, hay partes que no se realizaban del mismo modo y es necesario modificarlas. A continuación se describen los cambios necesarios para cada clase del patrón. Hay que destacar que se van a explicar los cambios que han sido más "generales" en cada

tarea de reestructuración (cada sección) y no se va a entrar en el detalle de cada problema o cambio específico de cada sección.

- Network. En esta clase se construye la solicitud HTTP para el servidor, con ayuda de la clase Router antes mencionada, y se gestiona la respuesta que devuelve el servidor (para cada petición necesaria al servidor). Si la solicitud se realiza con éxito, se devuelve el JSON de respuesta y si no es exitosa, se gestiona el tipo de error con su código. La gestión de estos errores es muy importante para los tests de Red (o de Network) que vamos a explicar más adelante, por lo que creamos una clase NetworkError que nos permite identificar cada error con su código y mensaje de error.

- APIClient. El contenido de esta clase es completamente nuevo en la aplicación. Esta clase tiene la función de decodificar los datos del JSON de respuesta para su posterior almacenamiento en CoreData. Anteriormente estos datos no se decodificaban, por lo que fue necesario un cambio importante en el modelo para hacerlo "decodable". Este cambio supuso una gran dificultad debido a que el modelo estaba adecuado para su uso junto a CoreData y este no es decodable, por lo que hubo que adecuarlo para soportar las dos funciones necesarias. Para cada funcionalidad, devuelve un objeto con sus datos decodificados para su posterior tratamiento por el Repositorio.

- Repository. En esta clase vamos a actualizar la base de datos con los datos enviados por el APIClient y recuperar los datos necesarios de la base de datos. Estas funciones ya se realizaban antes, y únicamente suponen una reubicación y reorganización de código (con cambios menores). Por último, se encarga de enviar los datos a la vista del modelo.

6.2 Nueva funcionalidad

Como se ha descrito con anterioridad, la implementación de la nueva funcionalidad se ha realizado en diferentes fases (Sprints), logrando una nueva versión funcional y sin errores de la aplicación. Cada versión es testeada manualmente antes de la Demo de la aplicación, y de forma automática, comprobando que supera con éxito todos los tests automáticos implementados, como se va a describir en el siguiente capítulo de este Trabajo de Fin de Grado.

Para cada cada fase, ha sido necesario seguir una serie de tareas (descritas en cada sprint de la sección 5.3.6). Estas tareas o pautas a seguir son similares en cada Sprint a pesar de que cada una tiene unas tareas concretas y únicas dependiendo de las necesidades de cada Historia de Usuario. Estas pautas a seguir son las siguientes:

- Implementar prototipo. Plasmar el diseño del prototipo en código listo para poder ser usado. Debe cumplir todas las características impuestas para ese prototipo: colores, disposición de los elementos, tamaño de letras...

- Implementar patrón Repositorio. Preparar la estructura de nuestro patrón Repositorio (sección 2.4.4) para recuperar los datos del servidor, decodificarlos y guardarlos en la base de datos, como hemos descrito en la sección del patrón Repositorio.
- Implementar Modelo - Vista - Vista de Modelo. Preparar la estructura de esta arquitectura (sección 2.4.3) para poder "conectar" la interfaz con el MVVM y el acceso a datos (Repositorio).
- Mostrar datos en la interfaz. Tras la "conexión" de los tres elementos anteriores, es necesario mostrar los datos recogidos del servidor y mostrarlos en la interfaz correspondiente usando la arquitectura MVVM y el patrón repositorio.
- Implementar tests. El último paso es implementar los tests para cada Historia de Usuario. Esta parte será explicada en el siguiente capítulo.

Hay que destacar la posiblilidad de que se puedan seguir diferentes pautas o estas mismas pero en diferente orden al descrito. Se ha decidido seguir estas pautas debido a que se ha considerado como la mejor forma de realizar la implementación y la más cómoda para el autor.

Pese a que en la aplicación existente se utilizaban librerías de terceros en numerosas ocasiones, para el incremento de la funcionalidad no ha sido así. Se ha intentado no tener dependencias de librerías de terceros lo máximo posible. Esta decisión es debido que las librerías pueden dejar de tener soporte y con el paso del tiempo dejar de ser compatibles con siguientes versiones del lenguaje de programación. Es por esto que, salvo alguna librería como *Alamofire*, que facilita mucho el trabajo, se ha optado por implementar el máximo código posible de forma nativa o con librerías propias del sistema.

Algunas de las librerías utilizadas son las siguientes:

- Alamofire: librería que permite realizar llamadas o peticiones asíncronas a la red.
- AlamofireImage: extensión de *Alamofire* que permite la gestión de imágenes.

6.3 Tests

En esta sección se va a describir una parte muy importante de este Trabajo de Fin de Grado, como es el Testing. Se van a definir todos los tipos de tests realizados, explicar las libreías utilizadas para la realización de estos tests y alguno de los numerosos tests llevados a cabo.

6.3.1 Introducción

La importancia de los tests automáticos radica en que dan calidad al software desde la fase de desarrollo y reducen los tiempos de publicación de las aplicaciones en cada versión que se publica. Esto es debido a que el trabajo de QA se ve reducido. QA es Quality Assurance, cuya función, a modo de resumen, es asegurar la calidad del software aplicando Testing.

Todos los tipos de tests que se van a llevar a cabo en este Trabajo de Fin de Grado deben seguir cuatro reglas:

- Deben ser automatizables, es decir, que no se requiera interacción humana con la aplicación durante las pruebas.
- Deben ser completos: verificar el mayor porcentaje de código posible, con casos de prueba para cada método de la aplicación y el mayor número de ellos posible.
- Deben ser independientes de todos los demás casos de prueba.
- Deben ser repetibles, con el mismo resultado en diferentes ejecuciones.
- Deben de llevarse a cabo sin necesidad de conexión a Internet o con el servidor.

6.3.2 UI tests

Los tests de interfaz de usuario tienen como objetivo comprobar que los datos o elementos mostrados en la pantalla son los esperados. También deben garantizar que las interacciones con dicha interfaz deben funcionar. Este tipo de tests permite asegurar que lo que se muestra al usuario es lo que debería, al igual que la funcionalidad.

Para llevar a cabo este tipo de tests debemos tener interfaces desacopladas del origen de los datos, es decir, que sean independientes del resto del código de la aplicación. Otro aspecto muy importante es que debemos ser capaces de cargar estas interfaces individualmente, sin que dependan de otras. Esto se consigue con la arquitectura MVVM y nuestro patrón Repositorio, como hemos descrito anteriormente.

De forma práctica, en los tests de interfaz de usuario somos capaces de lanzar una interfaz de forma independiente, sin necesidad de que un usuario haga login (algo imprescindible para que le resto de la aplicación funcione en condiciones normales, fuera de tests) o de otro tipo de datos necesarios. Esto es posible gracias a los Test Doubles, que son, en este caso, clases que reemplazan a las originales con el objetivo de probar el código. A estas clases las llamamos clases "mock". El procedimiento es el siguiente:

- Creamos clases "mock" para las clases Repository y APIClient.

  - MockRepository: esta clase hereda de su respectivo Repository, tiene los mismos métodos que el padre pero pasa, como argumento, un JSON al MockAPIClient respectivo. Este simula el JSON de respuesta del servidor con los datos que queremos que se usen para el test. Estos JSON van a ser muy importantes en todos los tests para poder testear varios escenarios de la aplicación.
  - MockAPIClient: esta clase se va a encargar, en vez de pedir el JSON con los datos a la clase Network, de transformar el JSON para su posterior decodificación.

- Para mostrar la interfaz, vamos a inicializar el ViewController "mock" con las clases MockRepository y con MockAPIClient (en vez de la normales como se hace en un flujo normal). Esto nos permite tener las interfaces independientes y mostrarlas cuándo se desee y sin necesidad de conexión a internet o al servidor.

Con todo esto, lo que queda es pensar e implementar los tests de interfaz de usuario: identificar todos los elementos de la interfaz y comprobar que se muestran, al igual que los datos esperados en pantalla (que serán los de nuestro JSON). También podemos hacer click sobre botones o deslizar hacia cualquier lado... Hay muchas opciones y su uso depende de las características de la interfaz y de lo que se quiere probar.

6.3.3 Unit tests

Los tests unitarios son los que permiten comprobar, de forma individual, cada funcionalidad de la aplicación por separado. Es decir, probar el correcto funcionamiento de cada método del código fuente.

Este tipo de pruebas son utilizadas habitualmente para comprobar que, después de añadir cambios en la aplicación, todo sigue funcionanado de la misma manera.

Estos tests unitarios se realizan con la misma estructura de clases Mock que hemos descrito con anterioridad.

6.3.4 Network tests

Este tipo de tests, a los que llamamos "Network tests " o "tests de red", son los que comunmente se denomina tests de integración, aunque en este Trabajo de Fin de Grado los utilizamos para verificar la integración con nuestra API de red.

Los tests de red permiten simular la respuesta del servidor para comprobar que el comportamiento de nuestra API de red es el esperado. El procedimiento es el siguiente: con la librería Hippolyte generamos un stub, o respuesta "falsa" del servidor, que se construye con un código HTTP y un JSON que pasamos como parámetro (simulando la respuesta JSON

del servidor).

De esta forma somos capaces de generar una respuesta para cada uno de los códigos de error del servidor que queramos. Esto permite comprobar que el comportamiento de nuestro código es el deseado para el caso de éxito y cada uno de los mensajes y códigos de error que devuelve el servidor cuando algo falla. Es muy importante que el flujo de ejecución de nuestra aplicación no se vea interrumpido por una respuesta de error del servidor no esperada.

6.3.5 Librerías necesarias

En este apartado se van a describir las librerías requeridas para la realización de los tests descritos anteriormente. Hay que remarcar que estos tests se pueden realizar con otras librerías pero estas son las elegidas para la realización de este Trabajo de Fin de Grado debido a la calidad de la documentación y a la facilidad de suso de las mismas.

- XCTest: Framework de Apple que permite la realización de tests automáticos. En este Trabajo de Fin de Grado se va a usar para los tests unitarios, de red y para los tests de Interfaz de usuario, como veremos a continuación.
- KIF: Es un framework que, junto con XCTest, ayuda en la creación de tests de Interfaz de Usuario automáticos ya que permite simular el flujo completo de la interfaz. Permite itroducir credenciales, tocar sobre una parte de la pantalla, identificar los elementos de la interfaz deslizar sobre la pantalla para que aparezcan las opciones de una celda (swipe)... y todo esto automatizado.
- Hippolyte: esta librería nos permite crear el "stub" para, con la ayuda de XCTest realizar los tests de red.

6.3.6 Tests automáticos realizados

Los tests realizados en este Trabajo de Fin de Grado, se han realizado para cada sección de la aplicación y en los tres niveles descritos anteriormente (unitarios, de interfaz y de red). Únicamente se van a reflejar un par de tests por sección y tipo para no sobrecargar la sección con la especificación de tests que pueden llegar a ser repetitivos para el lector debido al gran parecido entre ellos.

Para los tests de Red (o Network) se van a describir todos los tests realizados pero solamente en una de las secciones debido a que estos tests son siempre los mismos para cada sección y para cada una de las conexiones con la base de datos de las secciones. La única diferencia entre unos tests de red y otros son la construcción del "stub" que depende de la solicitud.

6.3.6.1 Perfil

UI tests:

Estos tests de interfaz de usuario se dividen en dos partes ya que esta sección se compone de dos interfaces diferentes: la interfaz del perfil y la de editar en perfil.

Nombre: testProfileViewWithAllTheFields	
ID: 1	**Escenario:** Perfil
Descripción: Comprobar que todos lo elementos de la interfaz se muestran en la misma.	
Resultados esperados: Todos los elementos de la vista mostrados en la interfaz.	
Resultados obtenidos: Todos los elementos son detectados.	
Resultado prueba: Éxito	

Tabla 6.1: T01 - UI test Perfil

Nombre: testHeaderEditProfileViewWithAllTheFields	
ID: 2	**Escenario:** Editar perfil
Descripción: Comprobar que los elementos de la celda se muestran en la interfaz	
Resultados esperados: Todos los elementos de la celda mostrados en la interfaz.	
Resultados obtenidos: Todos los elementos son detectados.	
Resultado prueba: Éxito	

Tabla 6.2: T02 - UI test Editar Perfil

Unit tests:

Nombre: testGetUserShouldReturnOneUser	
ID: 3	**Escenario:** Perfil y editar perfil
Descripción: Comprobar que la funcionalidad "getUser" devuelve un usuario pasando como argumento un JSON dado (equivalente a la respuesta del servidor).	
Resultados esperados: Número de usuarios igual a 2	
Resultados obtenidos: Número de usuarios igual a 2	
Resultado prueba: Éxito	

Tabla 6.3: T03 - Unit test Perfil 01

Nombre: testSetUserShouldReturnNameProperty	
ID: 4	**Escenario:** Perfil
Descripción: Comprobar que la funcionalidad "setUser" devuelve el valor de la propiedad "Nombre" esperado.	
Resultados esperados: "nombre" = "Héctor"	
Resultados obtenidos: "nombre" = "Héctor"	
Resultado prueba: Éxito	

Tabla 6.4: T04 - Unit test Perfil 02

Network tests:

Nombre: testShouldReturnOkWhenGetUserHTTPCodeReturnIsSuccess	
ID: 5	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error 200.	
Resultados esperados: La solicitud se realiza sin errores de red, simulando un código 200.	
Resultados obtenidos: La solicitud se realiza sin errores de red.	
Resultado prueba: Éxito	

Tabla 6.5: T05 - Network test 200

Nombre: testShouldReturnFailWhenGetUserHTTPCodeReturnInvalidRequestCode	
ID: 6	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error 400.	
Resultados esperados: InvalidRequestCode	
Resultados obtenidos: InvalidRequestCode	
Resultado prueba: Fracaso	

Tabla 6.6: T06 - Network test 400

Nombre: testShouldReturnFailWhenGetUserHTTPCodeReturnInvalidCredentials	
ID: 7	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error 401.	
Resultados esperados: InvalidCredentials	
Resultados obtenidos: InvalidCredentials	
Resultado prueba: Fracaso	

Tabla 6.7: T07 - Network test 401

Nombre: testShouldReturnFailWhenGetUserHTTPCodeReturnPropertyAlreadySet	
ID: 8	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error 403.	
Resultados esperados: PropertyAlreadySet	
Resultados obtenidos: PropertyAlreadySet	
Resultado prueba: Fracaso	

Tabla 6.8: T08 - Network test 403

Nombre: testShouldReturnFailWhenGetUserHTTPCodeReturnNotFound	
ID: 9	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error 404.	
Resultados esperados: NotFound	
Resultados obtenidos: NotFound	
Resultado prueba: Fracaso	

Tabla 6.9: T09 - Network test 404

Nombre: testShouldReturnFailWhenGetUserHTTPCodeReturnServerError	
ID: 10	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error 500.	
Resultados esperados: ServerError	
Resultados obtenidos: ServerError	
Resultado prueba: Fracaso	

Tabla 6.10: T10 - Network test 500

Nombre: testShouldReturnFailWhenGetUserHTTPCodeReturnUnknowmError	
ID: 11	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error 9999999.	
Resultados esperados: UnknowmError	
Resultados obtenidos: UnknowmError	
Resultado prueba: Fracaso	

Tabla 6.11: T11 - Network test 9999999

Nombre: testShouldReturnFailWhenGetUserNotConnectedWithServer	
ID: 12	**Escenario:** Perfil
Descripción: Comprobar que la respuesta de nuestra API de red es la esperada simulando que el servidor devuelve un código de error -1009.	
Resultados esperados: NotConnectedWithServer	
Resultados obtenidos: NotConnectedWithServer	
Resultado prueba: Fracaso	

Tabla 6.12: T12 - Network test -1009

La primera vez que se llevaron a cabo estos tests de Red, todos fracasaron excepto el primero, con código 200. Esto permitió descubrir que se estaba haciendo mal la gestión de los errores devueltos por parte del servidor. Este problema ha sido solucionado y estos tests no han vuelto a fallar para ninguna otra funcionalidad que requiera una petición al servidor. Estos tests han permitido descubir un problema en el código de la aplicación antes de lanzarse a un entorno "real" que, en determinadas situaciones, habría supuesto un cierre inesperado de la aplicación. Esto es lo que se quiere evitar con estos tests automáticos, cómo se ha descrito con anterioridad.

6.3.6.2 Vacaciones

UI tests:

Nombre: testTotalDaysEqualToExpected	
ID: 13	**Escenario:** Vacaciones
Descripción: Comprobar que los días de vacaciones totales son los esperados.	
Resultados esperados: 23	
Resultados obtenidos: 0	
Resultado prueba: Fracaso	

Tabla 6.13: T13 - UI test Vacaciones 01

Este test no tuvo éxito y nos permitió descubir que el JSON que se estaba usando como datos para la interfaz no estaba bien definido. Este test no permitió mejorar ninguna funcionalidad pero si mejorar la forma de escribir los tests.

Nombre: testHolidaysViewWithAllTheFields	
ID: 14	**Escenario:** Vacaciones
Descripción: Comprobar que todos los elementos de la interfaz se muestran en la misma.	
Resultados esperados: Todos los elementos de la vista se muestran en la interfaz.	
Resultados obtenidos: Todos los elementos son detectados.	
Resultado prueba: Éxito	

Tabla 6.14: T14 - UI test Vacaciones 02

Unit tests:

Nombre: testGetUserHolidaysShouldReturnTwoElements	
ID: 15	**Escenario:** Holidays
Descripción: La funcionalidad que permite obtener las vacaciones debe devolver dos elementos (de vacaciones).	
Resultados esperados: 2	
Resultados obtenidos: 2	
Resultado prueba: Éxito	

Tabla 6.15: T15 - Unit test Vacaciones 01

Nombre: testAskForHolidaysShouldReturnExpectedStatus	
ID: 16	**Escenario:** Vacaciones
Descripción: La funcionalidad de pedir vacaciones devuelve una respuesta del servidor (nuestro JSON) con una solicitud de vacaciones con estado pendiente (valor: 1)	
Resultados esperados: 1	
Resultados obtenidos: 1	
Resultado prueba: Éxito	

Tabla 6.16: T16 - Unit test Vacaciones 02

6.3.6.3 Tablón de anuncios

UI tests:

Nombre: testNoticeBoardFourthCellMustHavePropertyCollection	
ID: 17	**Escenario:** Tablón de anuncios
Descripción: La cuarta celda debe mostrar el mensaje esperado.	
Resultados esperados: "aviso de prueba"	
Resultados obtenidos: "aviso de prueba"	
Resultado prueba: Éxito	

Tabla 6.17: T17 - UI test Tablón de anuncios 01

Nombre: testNoticeBoardMustHaveFiveNotices	
ID: 18	**Escenario:** Tablón de anuncios
Descripción: La vista debe mostrar cinco celdas con sus elementos correspondientes.	
Resultados esperados: 5	
Resultados obtenidos: 5	
Resultado prueba: Éxito	

Tabla 6.18: T18 - UI test Tablón de anuncios 02

Unit tests:

Nombre: testGetNoticesShouldReturnZeroNotices	
ID: 19	**Escenario:** Tablón de anuncios
Descripción: La funcionalidad debe devolver cero anuncios. Este test sirve para comprobar que el comportamiento de la funcionalidad es el adecuado con un JSON vacío.	
Resultados esperados: 0	
Resultados obtenidos: Error	
Resultado prueba: Fracaso	

Tabla 6.19: T19 - Unit test Tablón de anuncios 01

Este test permitió descubir que un JSON de respueta vacío del servidor provocaría un cierre inesperado de la aplicación.

Nombre: testGetNoticesShouldReturnCreatorProperty	
ID: 20	**Escenario:** Tablón de anuncios
Descripción: Comprobar que la respuesta falsa (JSON) devuelve los datos esperados.	
Resultados esperados: [”1”, ”1”, ”1”, ”999”, ”2”]	
Resultados obtenidos: [”1”, ”1”, ”1”, ”999”, ”2”]	
Resultado prueba: Éxito	

Tabla 6.20: T20 - Unit test Tablón de anuncios 02

6.3.6.4 Equipo

UI tests:

Nombre: testPhoneButtonFirstRowEnabled	
ID: 21	**Escenario:** Equipo
Descripción: Comprobar que la primera celda (primer trabajador) tiene el botón de teléfono activado.	
Resultados esperados: True	
Resultados obtenidos: True	
Resultado prueba: Éxito	

Tabla 6.21: T21 - UI test Equipo 01

Nombre: testSecondTeamMateIsHector	
ID: 22	**Escenario:** Equipo
Descripción: Comprobar que el nombre del trabajador que aparece en la segunda celda es el esperado.	
Resultados esperados: "Héctor"	
Resultados obtenidos: "default"	
Resultado prueba: Fracaso	

Tabla 6.22: T22 - UI test Equipo 02

Los datos que se muestran en la interfaz no se ordenan bajo ningún criterio al ser solicitados a la base de datos, por lo que cada vez se colocan de una forma en la interfaz y en este caso el test ha fallado.

Unit tests:

Nombre: testGetTeamShouldReturnNameProperty	
ID: 23	**Escenario:** Equipo
Descripción: Comprobar que la funcionalidad se comporta como debería, devolviendo el valor de la propiedad "nombre" de trabajador esperada.	
Resultados esperados: ["hector", "user", "default"]	
Resultados obtenidos: ["hector", "user", "default"]	
Resultado prueba: Éxito	

Tabla 6.23: T23 - Unit test Equipo 01

Nombre: testGetTeamShouldReturnThirtyTwoUsers	
ID: 24	**Escenario:** Equipo
Descripción: La funcionalidad que permite obtener todos los trabajadores debe devolver 32 elementos.	
Resultados esperados: 32	
Resultados obtenidos: 32	
Resultado prueba: Éxito	

Tabla 6.24: T24 - Unit test Equipo 02

6.3.6.5 Información de la empresa

UI tests:

Nombre: testCompanyInfoCellWithAllTheFIelds	
ID: 25	**Escenario:** Empresa
Descripción: Comprobar que todos los elementos de la interfaz se muestran en la misma.	
Resultados esperados: Todos los elementos de la vista se muestran en la interfaz.	
Resultados obtenidos: Todos los elementos no son detectados.	
Resultado prueba: Éxito	

Tabla 6.25: T25 - UI test Empresa 01

Nombre: testCompanyHeaderCellWithAllTheFIelds	
ID: 26	**Escenario:** Empresa
Descripción: Comprobar que todos los elementos de la celda se muestran en la interfaz.	
Resultados esperados: Todos los elementos de la celda se muestran en la interfaz.	
Resultados obtenidos: Todos los elementos son detectados.	
Resultado prueba: Fracaso	

Tabla 6.26: T26 - UI test Empresa 02

Este test no se realizó con éxito debido a que los elementos no estaban correctamente identificados, por lo que el test no los detectó a pesar de que, efectivamente, estaban presentes en la intefaz.

Unit tests:

La sección de Empresa no tiene tests unitarios debido a que no necesita datos del servidor para mostrar en su interfaz.

6.3.6.6 Ajustes

UI tests:

Nombre: testSettingsViewWithAllTheFIelds	
ID: 27	**Escenario:** Settings
Descripción: Comprobar que todos los elementos de la interfaz se muestran en la misma.	
Resultados esperados: Todos los elementos de la interfaz se muestran en la misma.	
Resultados obtenidos: Todos los elementos son detectados.	
Resultado prueba: Éxito	

Tabla 6.27: T27 - UI test Ajustes

Unit tests:

La sección de Settings no tiene tests unitarios debido a que no necesita datos del servidor para mostrar en su interfaz.

6.3.6.7 Administración

UI tests:

Nombre: testChangeStatusSecondRow	
ID: 28	**Escenario:** Administración
Descripción: Comprobar que en la segunda celda, al hacer swipe aparecen las opciones para esa solicitud.	
Resultados esperados: Se hace el swipe (deslizamiento) automáticamente y aparecen las opciones.	
Resultados obtenidos: El swipe se realiza y aparecen las opciones esperadas.	
Resultado prueba: Éxito	

Tabla 6.28: T28 - UI test Administración 01

Nombre: testFirstTableRowMustHavePropertyCollection	
ID: 29	**Escenario:** Administración
Descripción: Comprobar que la primera celda tiene el valor de las propiedades esperadas.	
Resultados esperados: Nombre: "Hector Rogel", fecha: "Del 4/23/19 al 4/24/19"	
Resultados obtenidos: Nombre: "Hector Rogel", fecha: "Del 4/23/19 al 4/24/19"	
Resultado prueba: Éxito	

Tabla 6.29: T29 - UI test Administración 02

Unit tests:

Nombre: testChangeHolidayStatusShouldReturnExpectedPropertyCollection	
ID: 31	**Escenario:** Administración
Descripción: Comprobar que la funcionalidad de cambiar el estado de una solicitud devuelve el valor de las propiedades esperadas.	
Resultados esperados: id: 99, userId: 999, status: 1	
Resultados obtenidos: id: 99, userId: 999, status: 1	
Resultado prueba: Éxito	

Tabla 6.30: T30 - Unit test Administración 01

Nombre: testGetMaxHolidaysShouldReturnZeroElements	
ID: 31	**Escenario:** Administración
Descripción: Comprobar que la funcionalidad de obtener el número máximo de vacaciones devuelve cero elementos.	
Resultados esperados: 0	
Resultados obtenidos: 0	
Resultado prueba: Éxito	

Tabla 6.31: T31 - Unit test Administración 02

6.3.7 Tests manuales realizados

En esta parte se van a describir los tests que no se pueden hacer de forma automática. Estos tests se deben realizar de forma manual debido a que requieren acciones que no son posibles de realizar por ninguna librería. Necesitan de acciones del usuario para llevarlos a cabo.

Estos tests manuales deben pasar satisfactoriamente para el posterior lanzamiento de la aplicación. En proyectos más grandes, esta tarea es realizada por un QA pero en este Trabajo de Fin de Grado no hay ningún desarrollador con estas características, por lo que ha llevado a cabo con la ayuda de un QA para entender los conceptos báscios. A continuación se muestran algunos de los tests manuales llevados a cabo antes del lanzamiento de la aplicación al entorno real.

- **Tests de dispositivo:**
  - **ID:** 01
  - **Descripción:** Login con Iphone 6, acceder a cada sección de la aplicación.
  - **Acción:** Introducir credenciales y navegar por cada sección.
  - **Resultado esperado:** Todas las secciones se muestran correctamente.

  - **ID:** 02
  - **Descripción:** Con todos los dispositivos, abrir aplicación tras minimizar.
  - **Acción:** Minimizar aplicación y volver a abrirla.
  - **Resultado esperado:** Aplicación se abre en el estado que se minimizó.
- **Tests de sección Perfil:**
  - **ID:** 03
  - **Descripción:** Cerrar sesión y login.

  - **Acción:** Cerrar sesión e introducir credenciales.
  - **Resultado esperado:** Login realizado con éxito y aplicación iniciada en sección Perfil.

- **Tests de sección Vacaciones:**
  - **ID:** 04
  - **Descripción:** Añadir nuevas vacaciones.
  - **Acción:** Acceder a la solicitud de vacaciones y solicitar vacaciones.
  - **Resultado esperado:** Se muestra la sección vacaciones con una nuva entrada pendiente de aprobación.

- **Tests sección Ajustes:**
  - **ID:** 05
  - **Descripción:** Desactivar notificaciones y comprobar que el cambio es persistente.
  - **Acción:** Switch off las notificaciones y comprobar en la base de datos que el dispositivo ha desaparecido.
  - **Resultado esperado:** El "switch" cambia de estado y la entrada del dispositvo en la base de datos ha desaparecido.

- **Tests de la Vista Principal:**
  - **ID:** 06
  - **Descripción:** Comprobar que se muestran tweets.
  - **Acción:** Acceder a la vista principal y comprobar que se muestren tweets.
  - **Resultado esperado:** Los tweets se muestan en la interfaz.

Capítulo 7

Conclusiones

Podemos concluir que se han alcanzado satisfactoriamente todos los objetivos propuestos en el inicio de este Trabajo de Fin de Grado. Se ha conseguido dominar las herramientas necesarias para la correcta realización del mismo y el dominio de un nuevo leguaje de programación como Swift. Se ha logrado cambiar completamente la arquitectura de una aplicación iOS con código legado a una mucho más eficiente para la inclusión de tests automáticos manteniendo la funcionalidad de la misma después de cada iteración. Se ha conseguido añadir tests automásticos que mejoran la calidad del software. Y, por último, se ha hecho un rediseño de una parte de la aplicación y se han implementado nuevas funciones de administración.

A título personal, este Trabajo de Fin de Grado ha sido muy útil como complemento a la formación universitaria, ya que me ha permitido aplicar y poner el práctica gran parte de los conocimientos aprendidos durante el transcurso de esta formación. Me ha permitido reforzar conceptos y conocimientos con un nuevo enfoque: el mundo laboral. Me ha permitido aprender un nuevo lenguaje de programación y aplicar técnicas a mi código que están "a la orden del día", y todo ello enfocado a mejorar la calidad del código, cosa que he entendido como imprescindible en el desarrollo de software.

Como posibles líneas futuras para esta aplicación están: migrar la aplicación a Swift 5, o añadir la posibilidad de que un usuario envíe mensajes a los administradores avisando de faltas médicas o personales. Otra idea sería el rediseño completo de la aplicación para hacerla más similar a las aplicaciones iOS actuales, ya que, hay que recordar, que el diseño de esta aplicación es de 2017.

Bibliografía

[1] Apple , *"Model-View-Controller"*. [En línea, última vez consultado: 14/04/2019] Disponible en: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[2] Rui Peres and Felipe Laso-Marsetti, *"Model-View-Controller (MVC) in iOS – A Modern Approach"*. [En línea, última vez consultado: 15/04/2019] Disponible en: https://www.raywenderlich.com/1000705-model-view-controller-mvc-in-ios-a-modern-approach

[3] Jay Strawn, "Design Patterns by Tutorials: MVVM". [En línea, última vez consultado: 17/04/2019] Disponible en: https://www.raywenderlich.com/34-design-patterns-by-tutorials-mvvm

[4] Apple, "Core Data". [En línea, última vez consultado: 20/04/2019] Disponible en: https://developer.apple.com/documentation/coredata

[5] Per-Erik Bergman, "Repository Design Pattern". [En línea, última vez consultado: 28/04/2019] Disponible en: https://medium.com/tiendeo-tech/ios-repository-pattern-in-swift-85a8c62bf436

[6] Raul Peña Alonso, "iOS: Repository pattern in Swift". [En línea, última vez consultado: 28/04/2019] Disponible en: https://medium.com/tiendeo-tech/ios-repository-pattern-in-swift-85a8c62bf436

[7] Wikipedia, "Observer (patrón de diseño)". [En línea, última vez consultado: 02/05/2019] Disponible en: https://es.wikipedia.org/wiki/Observer_(patr\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\accent19o\egroup\spacefactor\accent@spacefactorn_de_dise~no)

[8] Atlassian, "Scrum, Aprende a utilizar scrum con lo mejor de él". [En línea, última vez consultado: 02/05/2019] Disponible en: https://es.atlassian.com/agile/scrum https://es.wikipedia.org/wiki/Observer_(patr\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\accent19o\egroup\spacefactor\accent@spacefactorn_de_dise~no)

[9] Jira, "La herramienta de desarrollo de software líder de los equipos ágiles". [En línea, última vez consultado: 02/05/2019] Disponible en: https://es.atlassian.com/software/jira

[10] Apple, "XCode". [En línea, última vez consultado: 02/05/2019] Disponible en: https://developer.apple.com/xcode/

[11] Git, "Git –fast-version-control". [En línea, última vez consultado: 03/05/2019] Disponible en: https://git-scm.com

[12] Latex, "The LateX project". [En línea, última vez consultado: 03/05/2019] Disponible en: https://www.latex-project.org

[13] Apple, "Swift: The powerful programming language that is also easy to learn". [En línea, última vez consultado: 06/05/2019] Disponible en: https://developer.apple.com/swift/

[14] Swift, "About Swift". [En línea, última vez consultado: 06/05/2019] Disponible en: https://docs.swift.org/swift-book/

[15] Alamofire GitHub, "Alamofire, elegant networking in swift". [En línea, última vez consultado: 08/05/2019] Disponible en: https://github.com/Alamofire/Alamofire

[16] Bruno Muniz, "UI Testing on iOS: KIF vs XCUITests". [En línea, última vez consultado: 28/05/2019] Disponible en: https://medium.com/supercharges-mobile-product-guide/ui-testing-on-ios-kif-vs-xcuitests-caf31a254428

[17] Apple, "XCTest". [En línea, última vez consultado: 05/06/2019] Disponible en: https://developer.apple.com/documentation/xctest

[18] Pablo Carrascal Muñoz, "SGEmployee: Aplicación iOS para la gestión de las vacaciones laborales". [En línea, última vez consultado: 01/07/2019] Disponible en: http://uvadoc.uva.es/handle/10324/27510

Anexos

Anexo A

Manual de Usuario

Este documento detalla la forma de utilizar la aplicación SGEmployee por parte de un Administrador. Un usuario con rol diferente de administrador no podrá ver la sección de Administración que vamos a mostrar a continuación.

En primer lugar, tras instalar la aplicación en un dispositivo móvil, el usuario debe hacer login en ella. La siguiente figura muestra la pantalla de Login:

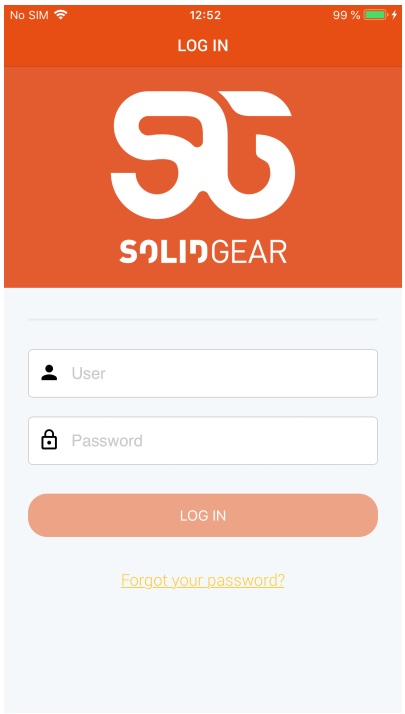

Figura A.1: Pantalla de Login

Una vez identificado aparecerá la pantalla principal con los tweets de la cuenta de la empresa como se muestra a continuación:

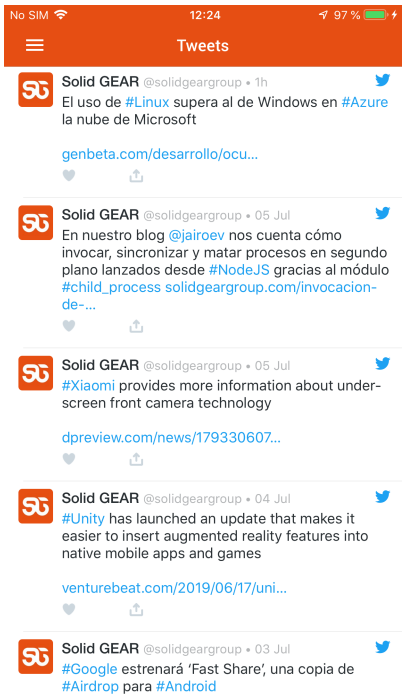

Figura A.2: Pantalla principal

Al pulsar sobre el icono en la parte superior izquierda de la pantalla se muestra el menú lateral:

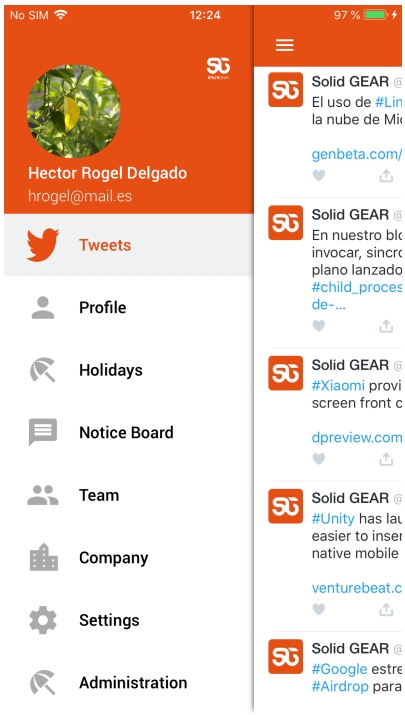

Figura A.3: Menú lateral

En este menú lateral aparece, como última sección, una opción de Administración. Esta opción aparece únicamente si el usuario es administrador. Si se pulsa sobre dicha función aparecerá la pantalla de Administración siguiente:

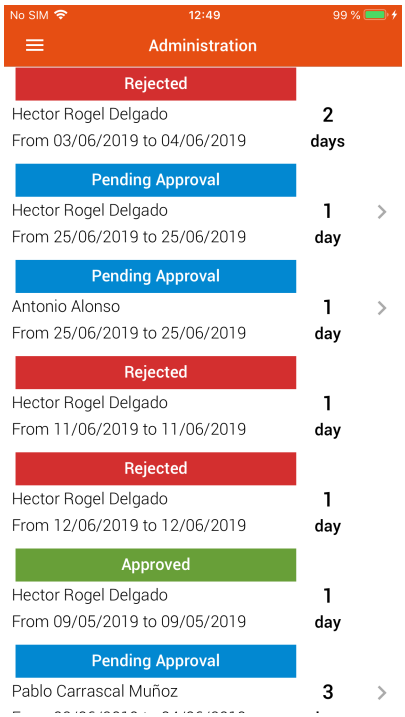

Figura A.4: Pantalla de Administración

En esta pantalla aparecen todas las solicitudes de vacaciones de todos los trabajadores sea cuál sea su estado. Sobre las celdas de las solicitudes, podemos deslizar hacia la derecha para que aparezca la opción de la Figura A.5 (en caso de solicitud aprobada). También podemos deslizar hacia la izquierda sobre una solicitud pendiente para que aparezcan las opciones de aceptar o rechazar, como se muestra en la Figura A.6:

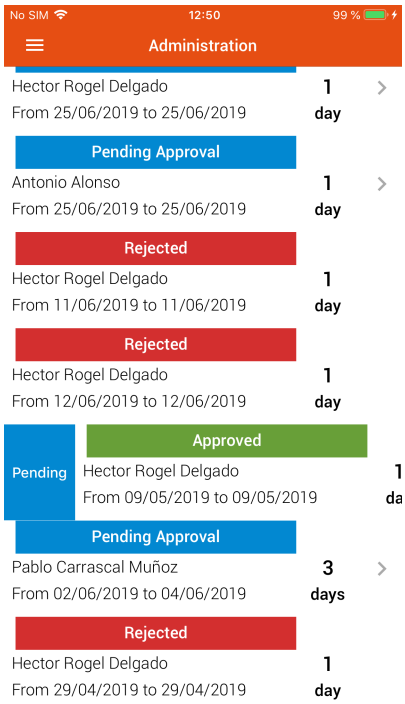

Figura A.5: Deslizamiento derecha solicitud

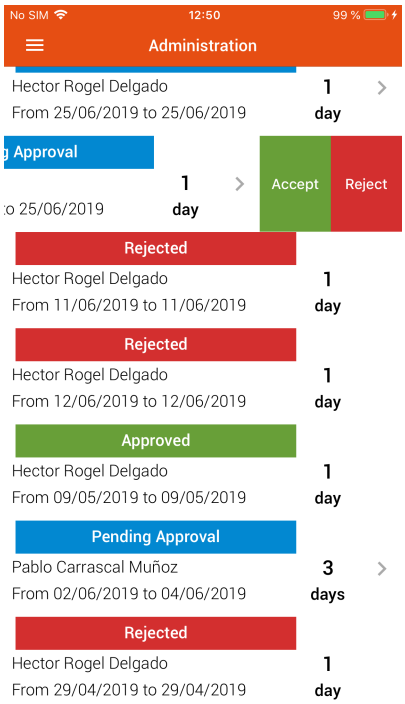

Figura A.6: Deslizamiento izquierda solicitud

Otra opción disponible en esta sección es pulsar sobre una solicitud pendiente para ver en detalle esa solicitud, con un historial de solicitudes, número de días de vacaciones restantes y las opciones de aceptar o rechazar, como se muestra a continuación:

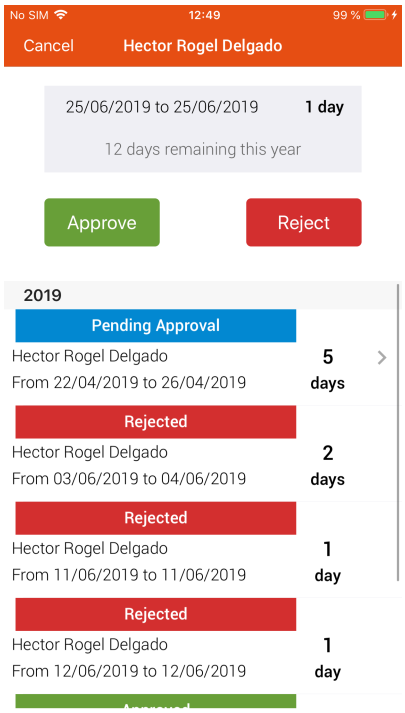

Figura A.7: Pantalla de detalle de solicitud

Si volvemos a la pantalla del menú lateral (Figura A3), otra nueva opción desarrollada en este Trabajo de Fin de Grado es el rediseño de la sección Equipo, que queda de la forma que se muestra en el Figura A.8. Si pulsamos sobre cualquiera de los empleados, aparecerán las opciones que se muestran el la Figura A.9. La opción de Ver las Vacaciones únicamente se muestra si el usuario es administrador.

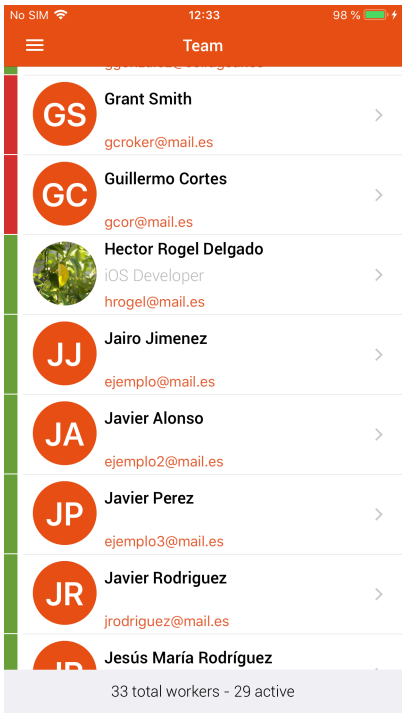

Figura A.8: Pantalla Equipo

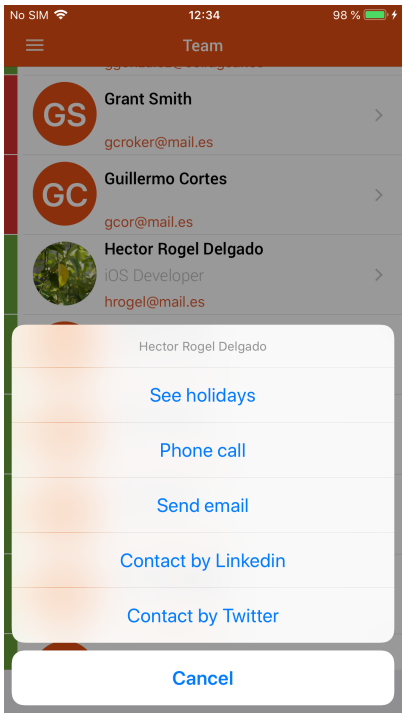

Figura A.9: Opciones Equipo

Al pulsar sobre la opción de Ver Vacaciones, aparecerá un historial completo por años de las solicitudes del empleado seleccionado, así cómo los días de vacaciones totales o restantes, y unas pestañas para navegar por cada año para ver los datos del mismo, como se muestra en la figura siguiente:

Figura A.10: Pantalla Historial Empleado

Anexo B

Manual de Instalación

Requisitos:

- Equipo informático con sistema operativo MacOS Mojave
- XCode 10.2 o superior
- Hay que tener instalado Ruby y la gema CocoaPods

Instalación:

- Descomprimir el proyecto del CD.
- Abrir el terminal y acceder a la carpeta SGEM del proyecto.
- Ejecutar el siguiente comando en el terminal para instalar las librerías de terceros:

  pod install

- Abrir el fichero SGEM.xcworkspace con XCode.
- Compilar y ejecutar el proyecto.