UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería en Tecnologías de la Información

Aplicación de Gestión de Turnos en el Marco Sanitario

Autor: Marcos Polo Ayuso

Dirigido por: Juan Manuel Cigarrán Recuero y Fernando López Ostenero

Curso: 2022-2023, convocatoria septiembre

Aplicación de Gestión de Turnos en el Marco Sanitario

Proyecto de Fin de Grado en Ingeniería en Tecnologías de la Información de modalidad específica

Autor: Marcos Polo Ayuso

Dirigido por: Juan Manuel Cigarrán Recuero y Fernando López Ostenero

Curso: 2022-2023

Agradecimientos

Quiero dedicar este TFG a las dos mujeres más importantes en mi vida. Mi mujer y mi madre, sin las cuales todo esto no habría sido posible. Por todo vuestro apoyo incondicional durante estos años solo puedo deciros esto: GRACIAS

También quiero dedicar este TFG a los muchos compañeros de grado con los que he compartido alegrías, risas, sufrimientos y un sinfín de emociones en estos años. Nunca pensé que en una universidad a distancia iba a tener tantos compañeros con los que compartir momentos. En especial a Óscar, que ha sido mucho más que un apoyo durante este trabajo, mil gracias. A Sukil, que tanto me ha ayudado con los idiomas. A Laura, mi "patito" durante estos largos años. A Bárbara y a Aurelio por sus ánimos y empujes constantes cuando los necesitaba.

Y por supuesto a Sofía, que ha sido mucho más que mi fuente de conocimiento. Ha sido mi guía y la iluminación cálida que este proyecto necesitaba. Te mereces lo mejor.

Gracias a todos vosotros

Resumen

La gestión de turnos es hoy en día una obligación en cualquier ámbito empresarial. Mediante este trabajo se pretende facilitar y simplificar un problema concreto dentro de un escenario concreto, la organización de los turnos de trabajo en un centro sanitario.

El sistema sanitario actual presenta, en un elevado número de casos, claras deficiencias tecnológicas en cuanto a su gestión. El mercado, sin embargo, ofrece soluciones que en la mayoría de los casos exceden las necesidades de organización de un servicio sanitario provocando el resultado totalmente opuesto que es, en muchos casos, intentar solucionar el problema de una forma casi analógica.

Dado que este problema tiene muchas particularidades, este proyecto ejemplifica un prototipo de software a medida que dará respuesta a múltiples situaciones reales ayudando al personal administrativo sanitario. Por supuesto, también proporcionará herramientas al personal médico que ayudarán a gestionar sus calendarios.

Este proyecto cubrirá las necesidades generales de un servicio hospitalario tales como alta y baja de usuarios, generación de grupos, creación de turnos, bajas, vacaciones e intercambios. De esta forma, intentará aportar una solución sencilla, eficaz y viable para este problema de gestión tan común.

Abstract

Shift management is nowadays an obligation at corporate level. The aim of this project is to facilitate and simplify a specific problem in a specific scenario, the organisation of work shifts in a healthcare centre.

The current healthcare system presents, in most cases, clear technological deficiencies in terms of management. The market, however, offers solutions that in most cases exceed the organisational needs of a health service, leading to the complete opposite result, which is to try to solve the problem with an analogical way.

Given that this problem has many particularities, this project exemplifies a prototype of customized software that will respond to multiple real-life situations by helping health administrative staff. Of course, it will provide tools for medical staff to help manage their schedules as well.

This project will cover the general needs of a hospital service such registration and removal of users, group generation, shift creation, sick leave, holidays and shift exchanges. In this way, it will attempt to provide a simple, effective and viable solution to this common management problem.

Palabras clave

Aplicación personalizada, centro sanitario, turno de trabajo, calendario, agenda, grupos de trabajo, java, hibernate, swing, DAO

Keywords

Customized application, healthcare facility, work shift, calendar, schedule, workgroup, java, hibernate, swing, DAO

Índice general

Índice de figuras

Capítulo 1 - Presentación

El trabajo es tan antiguo como la propia humanidad, la necesidad de dividir y organizar las tareas en una comunidad es algo necesario para la propia supervivencia. Con el tiempo, el ser humano empieza a ser consciente de la necesidad de establecer turnos de trabajo de forma rotatoria para poder cubrir una serie de conceptos, estos van desde la producción hasta la propia presencialidad continua de un servicio.

Tenemos constancia por escritos antiguos de la época romana sobre la división y organización de un turno laboral en múltiples situaciones. Guardias de los soldados, construcción de estructuras, relevos de los galeotes en los barcos …

El ser humano no puede desempeñar una misma función de forma continuada durante un tiempo ilimitado, por este motivo nace la necesidad de establecer turnos de trabajo que cubran toda la jornada.

Si bien en un primer momento la distribución de los turnos se hace de una forma puramente dictatorial en la que el director o empleador lo regula sin ninguna supervisión, con el tiempo, se empieza a regular la distribución de estos turnos con leyes y normativas.

La primera norma legislativa sobre los turnos de trabajo nace en el Reino Unido y viene inspirada por el lema acuñado por Robert Owen en 1810 *"ocho horas de trabajo, ocho horas de recreo, ocho horas de descanso"*.

Con el tiempo este lema llevo a la estandarización de las jornadas de ocho horas que tenemos en la actualidad.

La evolución del lema de Owen desembocará en los tres turnos de trabajo que conocemos en nuestros días: mañana, tarde y noche. La mayoría de los paises legislan y promueven turnos basados en este sistema en nuestros días.

En la actualidad todos conocemos situaciones en las que puede darse un trabajo de doble o incluso triple turno (con un merecido descanso posterior). El teletrabajo y las nuevas formas de "turno libre" o el "trabajo a demanda" abren nuevos escenarios que en el momento de la redacción de este trabajo están en proceso de regulación.

El mundo cambia, la sociedad cambia, el trabajo cambia … Por ello, como bien dijo Darwin, debemos adaptarnos para poder sobrevivir y ser más eficientes. Debemos aceptar que la organización del turno laboral que conocemos actualmente y basada en el lema de Owen, puede ser muy diferente ya no en el próximo siglo si no en la próxima década.

1.1 Motivación

Mi relación con el mundo sanitario es muy cercana por motivos personales prácticamente desde mucho tiempo atrás. En los últimos años he escuchado en muchas conversaciones de muchos médicos y enfermeras de muchos centros hospitalarios quejándose sobre su sistema de organización de turnos.

La organización de los turnos de trabajo se convirtió en un tema tan recurrente que, cuando tuve que decidir sobre qué iba a enfocar este TFG[1], fue una de las primeras ideas que me vino a la cabeza.

Un trabajo fin de grado debería ser el remate de una etapa formativa que deberías haber disfrutado recorriendo. En ese sentido, este proyecto además me ha permitido cubrir muchos aspectos que son importantes para mi tanto a nivel personal como profesional.

A nivel personal: La necesidad de asesoramiento y recopilación de información me ha permitido mantener contacto con personas a las que tengo mucho aprecio. Ha sido una parte enormemente gratificante en ese sentido.

A nivel profesional: Me ha servido para afianzar conocimientos adquiridos durante el grado y ampliar otros que por desgracia no da tiempo a profundizar. Mi principal motivación en este TFG era desarrollar el proceso “Full Stack” al completo. Desde la historia de usuario hasta la documentación final y entrega del trabajo.

[1] TFG – Trabajo fin de grado

1.2 Objetivos

El objetivo de este proyecto es crear una aplicación que sirva para ayudar al personal administrativo y a los médicos de un servicio sanitario a gestionar sus calendarios y turnos de trabajo.

A los médicos les vendría muy bien una aplicación que les permitiese gestionar una serie de tareas cotidianas relacionadas con su calendario laboral. Muchas veces, buscar un compañero con el que realizar una operación tan simple como intentar intercambiar un turno conlleva búsquedas anárquicas que suceden en sitios como la cafetería del centro sanitario al no poder ser consultadas.

También a veces el notificar que te vas a poner de baja o que quieres solicitar días libres puede convertirse en una tediosa comunciación entre médicos y personal administrativo, la aplicación debería ayudar a, por lo menos, colaborar en estas comunicaciones.

Al personal administrativo le sería tremendamente útil poder consultar en una misma aplicación cosas tan evidentes como el turno de trabajo de los médicos y que, en el caso de tener que realizar algún cambio, se haga de una forma equitativa y relativamente automatizada. Este último punto es interesante ya que en muchos servicios existen problemas que llegan incluso al plano personal debido a que, en caso de haber cambios, haya personal que tenga la sensación de que si la modificación ha sido realizada por un humano puede haber un trato desigual. Si existe una aplicación que lo haga en base a unos parámetros constantes nadie podrá poner pegas a ellos ya que le ha tocado porque un sistema a priori imparcial lo ha decidido.

Si en el caso de unos cambios de calendario que conlleven una gran complejidad y el sistema no fuese capaz de resolverlo por si mismo, el sistema debería permitir que se hiciesen los ajustes de forma manual bajo la responsabilidad del personal administrativo.

En resumen, la aplicación deber enfocarse en ayudar a la gestión de los turnos de un sistema sanitario adaptándose a sus particularidades, pero hacerlo de una forma rápida y sencilla para todo el personal que lo vaya a utilizar.

Capítulo 2 – Análisis de competidores

Dentro de la inmensidad del mundo informático existen muchas aplicaciones que cumplen funciones muy similares a las de este proyecto. El mercado nos ofrece desde soluciones muy generalistas que no están diseñadas de forma propiamente dicha para nuestro objetivo hasta soluciones que lo exceden.

Si pudiesemos hacer una división de estos competidores podríamos dividirlos en dos grandes grupos, aquellos que únicamente permiten la creación de calendarios estáticos que requieren que sea el humano el que los elabora y por otra parte aquellos que si permiten un mayor o menor grado de autonomía de creación del calendario.

2.1 Calendarios estáticos limitados

Son programas que sin estar realmente diseñados específicamente para ello incluyen la posiblidad de trabajar parcialmente con agendas, calendarios y turnos.

Encontramos en primer lugar opciones relacionadas con los gestores de correo electrónico. Así por ejemplo podemos ver que Google calendar / Outlook / Thunderbird permiten crear calendarios de forma estática pero no permiten de forma directa crear roles y usuarios. La Figura 1 nos muestra como compartir un calendario estático dentro de Goole Calendar, a su vez la Figura 2 nos muestra la misma función, pero dentro de Microsoft Outlook.

Es cierto no obstante que outlook se ve impulsado por su ecosistema empresarial teams que permite la invitación a tareas e incluso la creación de tareas para usuarios dentro de un equipo. Pero como decimos no permite una generación ni asignación directa de usuarios a un calendario propiamente dicho.

Figura 1 Imagen de google calendar y función de compartir calendario

Fuente: quehowto.com

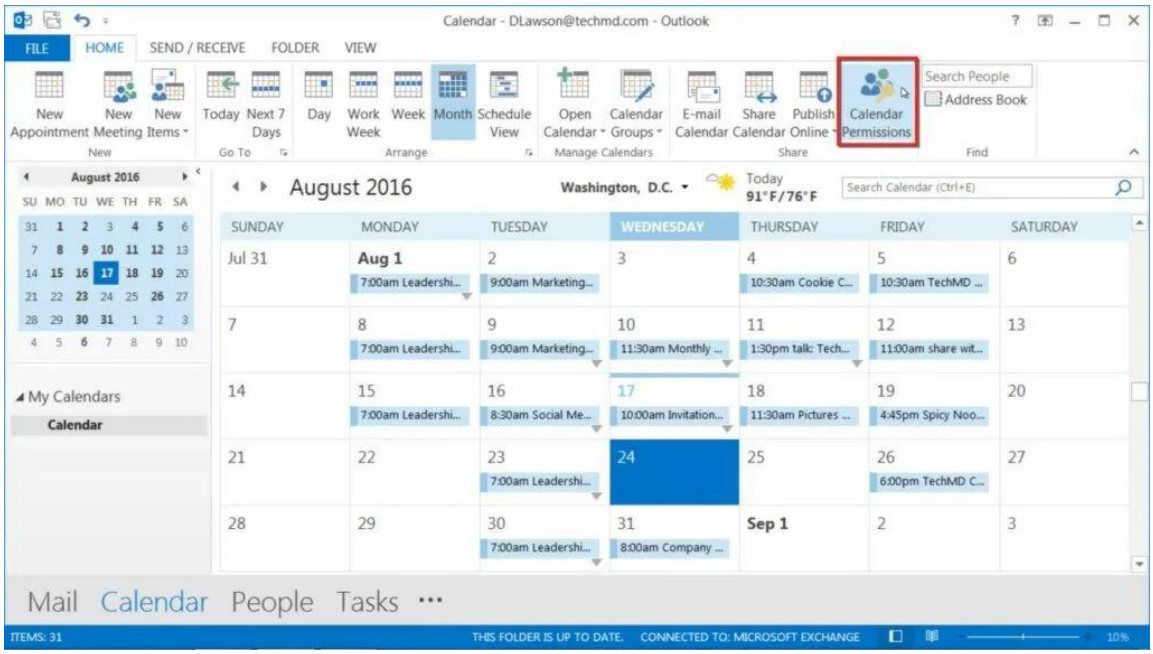

Figura 2 Imagen de outlook y la función compartir calendario

Fuente: techmd.com

En segundo lugar, existen también softwares dedicados a la gestión de proyectos empresariales, incluso específicos para el ámbito informático como pueden ser Trello, Jira, Wrike o el propio Microsoft Project. No obstante, estos programas están más enfocados a la gestión de tareas, funciones y partes de un proyecto, no a la gestión de un calendario laboral automatizado que es nuestro objetivo.

En la Figura 3 podemos ver una imagen de Jira donde se muestran las tareas pendientes para distintos usuarios con formato similar a un Scrum Board[2]

[2] SCRUM Board, Sistema que sirve para ver en qué estado se encuentra una tarea concreta, admitiendo TO DO / In progress / Done. Ejemplo en la Figura 49

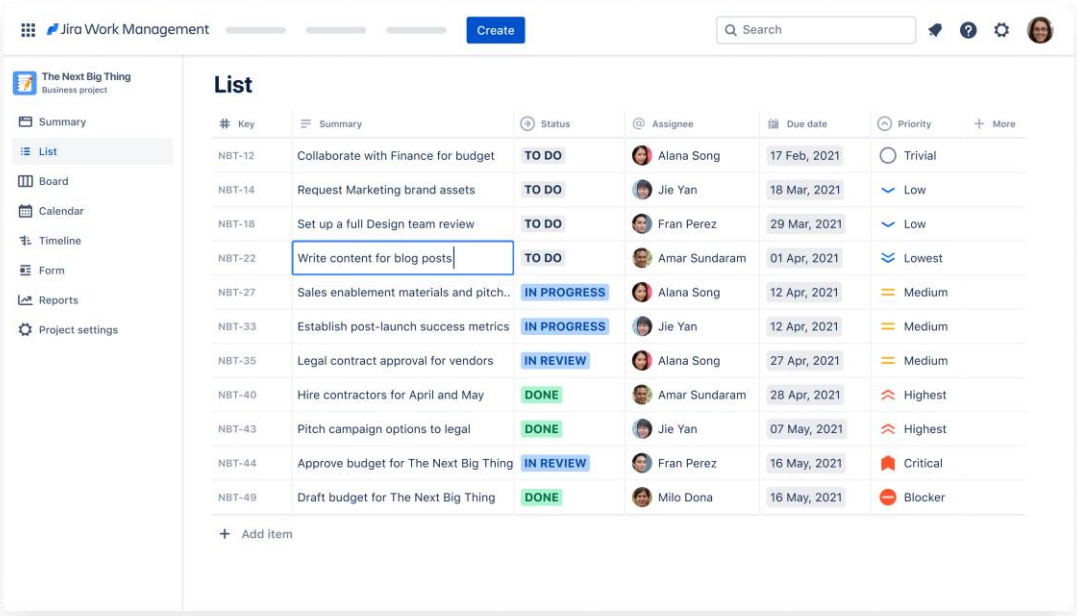

Figura 3 Imagen software JIRA

Fuente: blog.deiser.com

2.2 Softwares dedicados a la gestión laboral

Este tipo de programas están pensados para muchos sectores laborales y no están limitados al ámbito sanitario. Normalmente son ERP en los cuales la gestión de agendas y calendarios es una parte importante.

El número de programas encontrados en el mercado es muy extenso y se desarrolla a lo largo y ancho de todo el planeta, pero particularmente en Estados Unidos y en los países del norte de Europa es donde podemos encontrar un mayor número de empresas que se dedican al diseño y soporte de este tipo de software.

Hemos decidido centrarnos en tres: ShiftBoard, WhenIWork y Deputy los cuales paso a comentar

- ShiftBoard

  Es uno de los softwares más populares a nivel industrial contando con clientes tan nombrados como la petrolera Shell. Podemos ver un ejemplo de este software en la Figura 4

  Como ventajas de este software destacamos:

  + Crear grupos y perfiles de trabajadores
  + Automatización de calendario en base a criterios personalizables
  + Calculador de tiempo por tarea
  + Redistribución automática del personal a otras tareas
  + Cálculo pormenorizado de costes y rendimientos
  + Multiplataforma
  + Notificaciones sms
  + Open api disponible para personalizaciones y addons del software
  + Soporte telefónico, tickets y live chat

A cambio tiene los siguientes inconvenientes:

- Pensado principalmente para el sector industrial basado en líneas de producción
- Funcionamiento y configuración complejos
- No tiene comunicación entre empleados de forma directa, solo mensajes grupales
- No permite videollamadas
- El soporte está exclusivamente en inglés

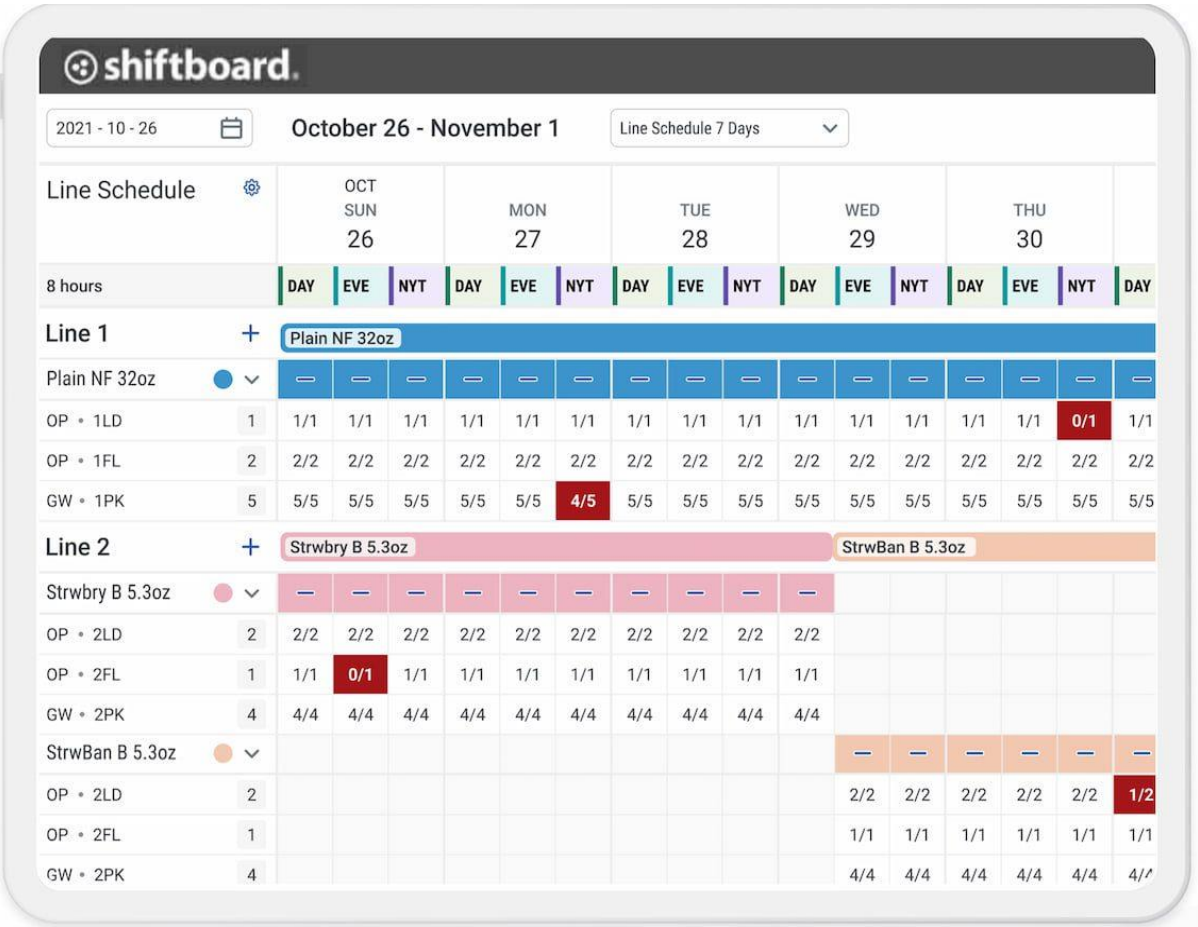

Figura 4 Imagen software shiftboard

Fuente: shiftboard.com

El coste de este software, explicado en la Figura 5 para esta aplicación, viene dado como es habitual en función del número de usuarios que van a manejarlo.

Versión	Número de usuarios	Precio
Lite	<35	45 $ / mes
Standar	<70	120 $ / mes
Plus	<125	275 $ / mes
Custom	> 125	Consultar

Figura 5 Tabla costes software Shiftboard

Fuente: shiftboard.com

❖ WhenIWork

Es posiblemente el software más famoso de gestión de plantillas para empresas. Está enfocado a cualquier tipo de sector y es muy configurable. Vemos una pantalla demostrativa en la Figura 6.

Podemos destacar como virtudes:

+ Calendario automático basado en plantillas personalizables
+ Multiplataforma
+ Plantillas incluidas para distintos sectores (paquetería, producción, sector sanitario …)
+ Interfaz sencilla y amigable
+ Organización de tiempo de tareas
+ Modo “ayuda si el empleado está disponible”
+ Geolocalización y cálculo de distancias para tareas que requieren desplazamientos
+ Geolocalización de empleado más cercano a una incidencia
+ Múltiples modos de comunicación entre empleados: individual, grupal, departamental …
+ Soporte de tickets y livechat
+ Open api (solo en versión completa)

Como inconvenientes se han encontrado

- No tiene soporte SMS
- No permite videollamadas
- No tiene soporte telefónico
- El soporte es exclusivamente en inglés

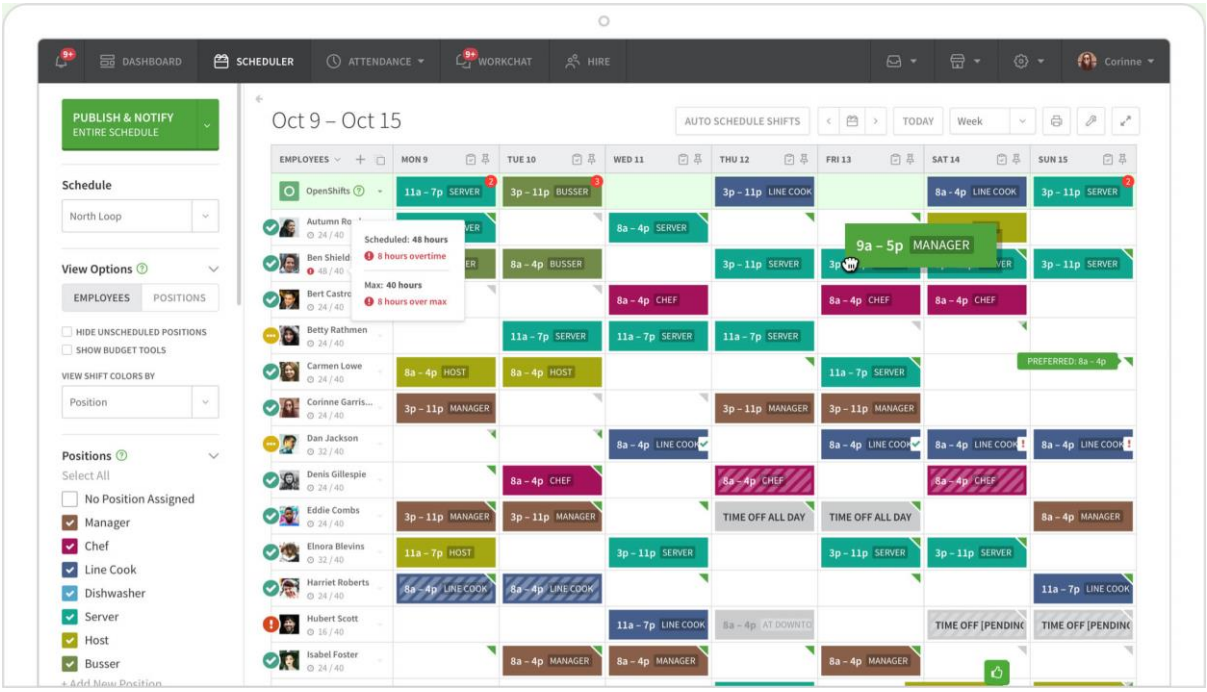

Figura 6 Imagen software WhenIWork

Fuente: wheniwork.com

En esta ocasión el precio viene dado por el tipo de versión teniendo dos tipos distintos, tenemos un resumen de precios en la Figura 7:

Versión	Funcionalidad	Coste
Standar	Todo menos: - Api - Funciones asociadas al tiempo y redistribución de tareas - Reglas personalizadas para configurar calendario	4 $ / usuario / mes
Full	Completa	8 $ / usuario / mes

Figura 7 Precio y características software WhenIWork

Fuente: wheniwork.com

❖ Deputy

De los softwares citados hasta ahora es posible que este sea el que más se adapta tanto al enunciado de este proyecto como al sector sanitario. En su web presumen de trabajar tanto para muchas clínicas privadas, servicios de emergencia con ambulancias, visitadores médicos a demanda … La Figura 8 nos muestra el funcionamiento general de su agenda.

De esta aplicación podemos destacar:

+ Diseñado pensando en el marco sanitario
+ Calculo automático de calendario en función de personal y tareas
+ Facilidad de configuración para movilidad, geolocalización y cálculo de tiempo / distancia
+ Configuraciones de costes y salarios muy elaborados
+ Sistema de comunicación avanzado a todos los niveles incluyendo videollamadas múltiples cifradas dentro del propio sistema
+ Multiplataforma
+ Interfaz intuitiva y amigable
+ Notificaciones SMS
+ Open api

Como inconvenientes destacamos:

– Necesidad de formación para manejar la aplicación debido a extensión
– Soporte únicamente vía mail
– Soporte exclusivamente en inglés

Podríamos decir que el trabajo de este fin de grado llevado a ERP de una forma profesional, con un equipo de trabajo amplio y con muchos recursos desembocaría en una aplicación como esta.

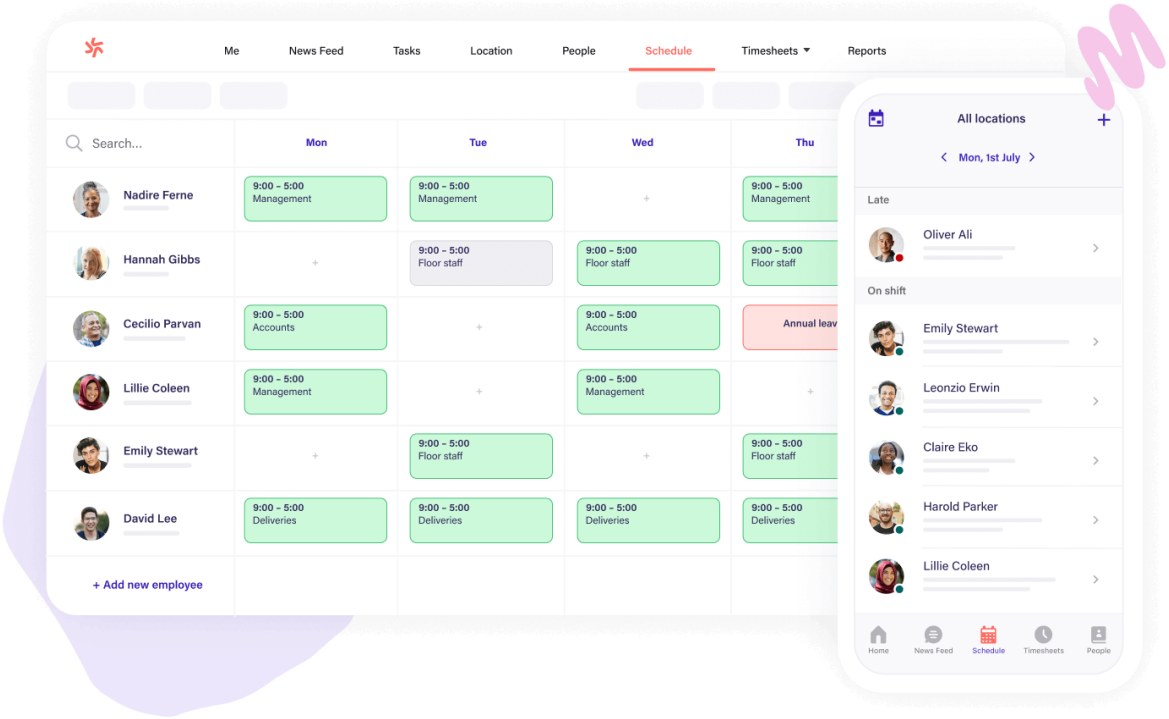

Figura 8 Imagen software Deputy

Fuente: deputy.com

Esta aplicación tiene precio fijo sin contar el número de usuarios fijándose este en 4,9 $ / mes / usuario

2.3 Software a medida

Finalmente, existen las soluciones a medida en las que una empresa informática desarrolla un software adaptado al contexto sanitario, que es el que nos compete, aunque luego haga modificaciones particulares para cada cliente.

Ejemplo de esto es el software chaman desarrollado por la empresa Costaisa situada en Barcelona. Este software es un ERP que incluye funciones organizativas de agenda y calendario automáticos. También incluye addons de compatibilidad con hardware clínico utilizable en las consultas. Este tipo de software como vemos se queda muy lejos del desarrollado en este proyecto suponiendo un desarrollo completo y final.

2.4 Software utilizado en centros sanitarios españoles

Tras un largo proceso de investigación entre muchos hospitales y servicios sanitarios públicos del ámbito nacional hemos encontrado que no existe una contratación directa entre las comunidades autonónomas a nivel de consejerías de salud con ninguna empresa de software.

En la mayoría de los casos me he encontrado situaciones muy "analógicas" (papel y boli) o digital a un nivel básico (tablas de excel rellenadas a mano). Existen servicios intrahospitalarios que han contratado ellos mismos alguno de los softwares citados anteriormente, contrándonos un caso en el que un servicio de urgencias tenía contratado uno y el servicio de UVI tenía contratado otro y los utilizan a nivel interno de forma independiente.

Otro caso es el del convenio entre un hospital universitario y la universidad de esa comunidad autonónoma la cual, les desarrolla un software como un proyecto de colaboración renovable.

En resumen, se ha observado que no existe una coordinación ni una cultura de contratar este tipo de servicios desde las comunidades autónomas si no que se deja en manos de los propios hospitales en general y de los propios servicios en particular el utilizar este tipo de herramientas.

2.5 Conclusiones

Tras analizar las tres posibles vertientes de este tipo de programas en el mercado (calendarios estáticos, sofwares generalista, softwares a medida), concluimos que el mercado está muy desarrollado con muchas soluciones a un precio asequible. No obstante, la gran mayoría de los productos pecan de extremistas en su funcionalidad ya sea por exceso o por defecto.

Nuestro proyecto va a tener la ventaja de estar perfectamente adaptado al sitio donde se va a utilizar y por este motivo supera a todas las soluciones encontradas el mercado debido a su personalización. Para poder llegar a un nivel de personalización similar sería necesario un software a medida con todo lo que ello conlleva tanto en precio como en otros recursos.

Existen muchas funciones además que exceden nuestro objetivo y que, aunque son realmente interesantes, nunca iban a ser utilizadas convirtiéndose en un sobrecoste de recursos. Nos referimos a funcionalidades como live chats, aviso por sms, geolocalización … ninguna de ellas tiene sentido en nuestra aplicación.

Cabe destacar que la mayoría de estos productos vienen del mundo anglosajón y es difícil encontrar programas que tengan soporte en castellano. Esto evidencia también otro punto a favor de este proyecto, una interfaz sencilla y en castellano que todos los usuarios puedan utilizar con facilidad.

Capitulo 3 - Propuesta

En este capítulo y tras haber analizado el mercado en el capítulo anterior, se hace evidente que lo que existe excede nuestra idea inicial y otras muchas se quedan muy cortas. Es momento de preparar nuestra idea y comenzar a desarrollarla.

Para poder dar forma a nuestra idea se recurre a los interesados de esta aplicación, es decir personal médico y administrativo de varios centros sanitarios que tanto me sugirieron este enunciado. Gracias a ellos podremos recolectar las ideas iniciales y sus deseos de funcionalidad. Esto en informática se llama “historias de usuario” y son una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos. Cada historia de usuario debe ser limitada, ésta debería poderse escribir sobre una nota adhesiva pequeña. (Cohn, 2006)

Tras recopilar mucha información, muchas historias de usuario y muchas peticiones ya nos encontramos en disposición de poder hacer una lista de requisitos funcionales.

3.1 Funcionalidades deseables

Tras varias entrevistas y recopilar varias historias de usuario las espectativas de funcionamiento se fijaron en que el software fuese capaz de realizar las siguientes tareas:

- Diferentes versiones del programa en función del perfil médico y administrativo
- Crear grupos de trabajo en función del número total del personal del servicio, así como de las instalaciones disponibles.
- Creación de turnos de trabajo con tres turnos diferentes: Mañana, tarde y guardia
- Los turnos de trabajo deberán tener unos descansos obligatorios asociados a las guardias
- Una cadencia de guardias fija en función del número de grupos de trabajo
- Sugerir compañeros con los que poder cambiar un turno de mañana/tarde o un turno de guardia de forma clara y rápida sin tener que consultar un calendario general del servicio.
- En caso de Vacaciones o bajas de compañeros las sustituciones han de realizarse de un modo justo y entendible
- Debe tener una interfaz de usuario mínima y sin subventanas.
- Visibilizar cuando una petición de cambio de turno es autorizada por dirección de forma sencilla.

La recomendación de la mayoría de las personas con las que colaboré es que no fuese un programa web ya que no es popular entre ellos debido a los problemas de compatibilidad entre navegadores que tienen en sus aplicaciones de uso diario en el trabajo.

3.2 Perfiles de usuario

La aplicación consta de dos perfiles de usuario bien diferenciados, el relativo a los trabajadores o médicos y el relativo al personal administrativo

3.2.1 Médico / empleado

Es el centro de la aplicación ya que esta se basa en el calendario que le afecta. Un médico ha de poder realizar en la aplicación las siguientes tareas:

- Consultar su calendario de trabajo
- Solicitar vacaciones
- Notificar cuando se encuentra de baja
- Obtener una lista de compañeros para poder cambiar un turno de trabajo concreto
- Notificar a la dirección sobre peticiones de cambio de turno

Los requisitos del médico es que solamente podrá pertenecer a un centro sanitario y a priori se considera que todos los empleados tienen la misma capacidad laboral.

3.2.2 Personal administrativo

Son los encargados de la gestión del centro sanitario teniendo potestad para un gran número de decisiones. Por simplicidad en la aplicación se ha decidido agrupar todas estas capacidades en un solo perfil, aunque en la realidad la cadena de mando está jerarquizada en muchos más niveles.

Las tareas que podrá realizar este perfil son las siguientes:

- Dar de alta nuevos médicos
- Modificar los datos del personal
- Contratar médicos para el centro sanitario de entre una lista de disponibles
- Crear y modificar grupos de trabajo
- Crear calendarios laborales mensuales
- Consultar calendarios
- Modificar de forma manual el calendario para realizar ajustes personalizados
- Autorizar o denegar las peticiones de cambio de turno de los empleados

El perfil administrativo solo será capaz de administrar su centro sanitario y es dado de alta por el administrador del sistema.

3.3 Conceptos clave

Una vez que tenemos claro lo que vamos a hacer y quién lo va a usar es necesario concretar y precisar los tres conceptos clave sobre los que pivotará toda la aplicación. De esta forma, teniendo estos tres conceptos claros podemos construir entorno a ellos.

Cuando hablamos de organización laboral atendiendo a su calendario hemos de distinguir varios conceptos:

- Agenda:
  - Resumen de eventos de un individuo en un turno de trabajo
- Turno de trabajo:
  - Franja horaria de trabajo en una jornada determinada. Aquí hemos considerado turnos de mañana, tarde y guardia
- Calendario:
  - Conjunto de turnos de trabajo expresados en periodos temporales. Mensual, semanal, quincenal …

3.3.1 Turnos de trabajo y "malus"

Los turnos de trabajo de este calendario laboral se crean en función de un parámetro clave que es el personal disponible. El sistema está preparado para poder diseñar calendarios con un sistema de cadencia de las guardias, es decir, el día importante sobre los que están diseñados los calendarios son los días de guardia.

Las guardias llevan asociado un descanso de las dos siguientes jornadas de trabajo. Por ejemplo, si se ha tenido guardia el lunes, no se trabaja martes y miércoles.

En función del número de personal disponible lo habitual es que un centro sanitario establezca una cadencia de guardias cada 6 u 8 días. Lo ideal es tener cuantas menos guardias al mes mejor y es el caballo de batalla siempre de las luchas laborales de este sector, si tenemos guardias cada 8 días tendremos menos guardias al mes por lo que se intenta tener personal para poder hacerlo así.

De esta forma, si tuviésemos una cadencia de guardia cada 6 días el turno sería G, L, L, T, M, M donde **G = Guardia, L = Libre, T = Tarde, M = Mañana**. A esto hemos de añadir la consideración de que, si los días que estamos de M o T son festivos, se convierten automáticamente en L.

El número de grupos de trabajo va directamente relacionado con la cadencia ya que si esta es cada 6 días el personal se agrupará en 6 distintos grupos de trabajo, análogamente si tenemos cadencias de 8 días tendremos 8 grupos de trabajo.

En la Figura 9 vemos el ejemplo de como se distribuirían los grupos con una cadencia de 6

Día	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
Grupo 1	G	L	L	T	M	L	G
Grupo 2	L	L	T	M	M	G	L
Grupo 3	L	T	M	M	G	L	L
Grupo 4	T	M	M	G	L	L	L
Grupo 5	M	M	G	L	L	L	L
Grupo 6	M	G	L	L	T	L	L

Figura 9 Tabla ejemplo turno semanal

Las guardias de un sanitario pueden ser de dos tipos:

- Guardias de festivo

  Entendiendo además de festividad, los fines de semana. Estas guardias dan comienzo a las 8:00 del día inicial y terminan a las 8:00 del día siguiente, es decir 24 horas.

- Guardias de días no festivo

  Habitualmente de lunes a viernes salvo que uno de esos días sea festivo. Duran desde las 15:00 hasta las 8:00 del día siguiente, es decir 17 horas.

Sin embargo, pese a que podemos clasificar las guardias en estos dos tipos, para poder puntuar el **malus** que cuesta cada guardia o incluso otro tipo de días hay que afinar más. La clasificación de los costes del malus vienen especificada en la Figura 10.

Tipo de turno	Puntuación
Día normal	10 puntos
Día normal víspera de festivo	15 puntos
Guardia día normal	25 puntos
Guardia víspera de festivo	30 puntos
Guardia fin de semana	50 puntos
Guardia festivo	150 puntos

Figura 10 Coste malus de turno por tipo

Si elegimos para este ejemplo el miércoles de la Figura 9, vemos que hay dos grupos trabajando de mañana, dos descansarían y otros dos estarían de tarde (el grupo de guardía entra a las 15:00 y sale a las 22) y un grupo estaría de noche que es el grupo de guardia.

El caso más favorable para tener una guardia sería el miércoles ya que tras las libranzas de jueves y viernes tampoco se trabajaría el sábado y el domingo. El caso más desfavorable sería el tener una guardia el viernes ya que las libranzas postguardia coinciden con el fin de semana por lo que “se pierden”.

Las cadencias se 6 y 8 se denominan también cadencias “hacia atrás” y “hacia adelante” respectivamente. Esto es así porque en una cadencia de 6 días si al médico le toca guardia un martes, la siguiente la tendrá un lunes. En una cadencia de 8 si le toca guardia un martes, la siguiente la tendrá un miércoles.

Decimos que la cadencia se establece en función del tamaño de la plantilla del centro sanitario ya que para establecer una cadencia cada 8 días, que es la deseada, hace falta más médicos que para una cadencia de 6.

El último aspecto que nos toca detallar es ¿cómo se sustituyen esas guardias en caso de incidencia?, es decir en caso de que haya una baja o un contratiempo. Para ello tenemos un concepto superimportante para el funcionamiento de este proyecto que es el **grupo antípoda**.

El grupo antípoda es el grupo candidato para sustituir la guardia de otro grupo, cada grupo tiene su propio grupo antípoda. La idea es que el grupo de médicos que están de guardia ese día tienen como grupo antípoda el grupo que está de turno de tarde. De esta forma, en caso de improvisto, la persona que está de tarde está prevenida de que le puede tocar quedarse a hacer también la noche. Elegir el sistema por el que un miembro del grupo es el candidato a ser sustituto es muy variopinto. Es posible hacerlo de forma aleatoria, rotatoria o como hemos hecho en este proyecto utilizando un sistema de puntuación llamado ***malus***[3]. Este sistema añade una puntuación al médico cada vez has tenido que hacer un sobreesfuerzo por cubrir a un compañero, de esta forma se garantiza la equidad en las sustituciones.

3.3.2 Tamaño de los grupos

Para poder decidir el tamaño de los grupos de trabajo antes citados hemos de conocer las instalaciones del centro sanitario para poder optimizarlo. En la Figura 11 vemos las posibles opciones de instalaciones que puede tener un hospital, pudiendo tener más de una de cada.

```
public enum AreaType {
    1 usage
    BOX, OBSERVACION, UVI, TRIAJE, MULTIPLE
}
```

Figura 11 Enumerado con las instalaciones de un centro sanitario

Para poder explicar cuántos médicos son necesarios para cada instalación muestro en la Figura 12 la función del código que utiliza el programa para calcularlo. Espero que el lector sea capaz de traducirla sin dificultad.

```
/** Calcula el número de personas óptimo para un grupo de trabajo en función de las instalaciones del centro sanitario. ...*/
2 usages  Marcos-PolAy
private int getSizeOfGroupForHealthCenter(HealthCenter healthCenter){

    int multiPurpose = crudManager.getNumberOfWorkAreas(AreaType.MULTIPLE, healthCenter);
    int triage = crudManager.getNumberOfWorkAreas(AreaType.TRIAJE, healthCenter);
    int icu = crudManager.getNumberOfWorkAreas(AreaType.UVI, healthCenter);
    int observationUnit = crudManager.getNumberOfWorkAreas(AreaType.OBSERVACION,healthCenter) * 2;
    int boxes = Math.round((float)(crudManager.getNumberOfWorkAreas(AreaType.BOX, healthCenter)/2));

    return Math.round((float)((multiPurpose+triage+icu+observationUnit+boxes)* employeesToFacilitiesRatio));

}
```

Figura 12 Función para la obtención del número óptimo de médicos por grupo

[3] Malus – Nombre que le damos al parámetro que contabiliza el estado de penalización del usuario en función del número de cambios forzados de calendario que ha tenido.

3.3.3 Bajas

Las bajas bajas laborales, sea cual sea su motivo, es algo que siempre trastoca los calendarios ya que no pueden posponerse o rechazarse. El tratamiento de las bajas por parte del sistema tiene tres posibles soluciones:

- No sustituir

  La baja no se cubre, los compañeros tienen que asumir que la carga de trabajo aumenta. Si bien no es lo deseable, es una situación real del mundo laboral y como tal, se contempla.

- Cubrir con un empleado externo

  Entre los médicos que no pertenecen al centro sanitario de forma regular, el personal administrativo elige uno. Se asigna al sustituto los mismos turnos que tenía el médico titular.

- Reasignar al personal

  Se utiliza al personal regular para cubrir al que está de baja. Tenemos tres posibles casos en función del turno asignado:

  - Turno de mañana

    Dado que el turno de mañana es el que cuenta con mayor personal, no se hace nada, es asumible pensar que entre los médicos presentes pueden repartirse el trabajo de forma temporal.

  - Turno de tarde

    El médico con el malus más bajo es reasignado de la mañana a la tarde y de esta forma queda cubierto.

  - Guardia

    Para sustituirla se utiliza al miembro del grupo antípoda con el malus más bajo. De forma automática se le asigna al sustituto descanso las dos jornadas siguientes a la guardia.

3.3.4 Vacaciones

Cuentan con las tres mismas opciones para solucionarla que una baja y se tratan de idéntica forma, pero tiene la diferencia de que el cuerpo directivo puede negarlas. Aplicado a terminos legales del mundo real, cada trabajador tiene derecho a un número determinado de días al año. La distribución de estos descansos viene fijada por el convenio y el contrato que tengan empresa y trabajador.

En resumen, unas vacaciones son administrativamente hablando igual que una baja, pero con la posibilidad de ser denegadas.

3.3.5 Intercambio

Es más que habitual que entre los compañeros haya intercambios en los turnos y como no podía ser menos, este programa ha de contemplarlo.

Tenemos ahora las siguientes posiblidades:

- Cambio de mañana por tarde y viceversa

  Ocurre durante el mismo día, el sistema permite solicitar el cambio de una persona que está de tarde por otra que está de mañana y viceversa.

- Cambio de guardia

  Para poder intercambiar una guardia por otra los médicos han de ser miembros de grupo antípoda entre sí. Al intercambiarse la guardia, obviamente también se arrastran en el cambio los días de descanso.

3.4 Requisitos tecnológicos

Ahora que ya disponemos de los requisitos deseables y de los perfiles que van a conformar nuestra aplicación es el momento de plentearnos que tecnologías y servicios vamos a tener que utilizar. La dedisión tomada es crear una aplicación sencilla cercana al concepto de standalone[4].

La primera decisión que tomar es que lenguaje de programación vamos a utilizar. Posteriormente deberemos elegir una forma en la que el usuario se va a comunicar con la aplicación, lo que informáticamente hablando llamamos “frontend” y finalmente deberemos decidir que hacemos con los datos que el software debe manejar, lo que informáticamente llamamos “backend”. Analicemos ahora ambos frontend y backend

3.4.1 Interfaz de escritorio (frontend)

La interfaz, aunque similar y con un diseño reconocible tendrá obviamente funcionalidades diferentes en función de cada perfil de usuario.

Elegiremos un sistema de interfaz que permita mediante menús, tablas, desplegables y elementos similares la comunicación con el usuario que lo está utilizando.

Dado que no queremos dependencias externas debemos utilizar algo que funcione directamente con el lenguaje de programación que elijamos.

3.4.2 Capa de acceso a datos (backend)

Dado que la aplicación va a ser utilizada por múltiples médicos y personal administrativo de forma simultánea los datos que vayamos a utilizar deberán estar disponibles para todos cumpliendo el concepto de consistencia, es decir, que los datos sean los mismos para todos los usuarios.

El parrafo anterior nos obliga a que los datos han de poder ser almacenados y consultados de forma persistente 24/7, deseando obviamente que todo esto sea transparente para los usuarios.

Para que la capa de acceso a datos funcione el programa ha de tener acceso a la red ya que los datos estarán centralizados externamente. Sin este acceso el programa no funcionará.

[4] Standalone – Aplicación ejecutable que no necesita de ningún servicio ni colaborador externo para funcionar

3.5 Fuentes de información

Un software informático que se precie de ser medianamente complejo no sale exclusivamente del cerebro del que lo desarrolla, inevitablemente vamos a necesitar una serie de fuentes de información que nos sirvan para poder abarcar desde una consulta puntual hasta la obtención de un conocimiento completo sobre algo hasta ese momento desconocido.

En la Sociedad de la Información como bien dijo Bangemann[5] en su informe allá en el año 1994. La información es el pilar de todo trabajo hoy en día. Por ello, hemos de conocer las fuentes de información que podemos consultar y si están actualizadas.

Un buen punto de partida de lista de fuentes sería la siguiente:

- Libros de texto base de la universidad
- Libros técnicos relacionados con cada una de las áreas del proyecto. Consultar la bibliografía del trabajo para verlos.
- Páginas web técnicas de referencia. Imprescindibles y habituales serán:
    - W3schools[6] para consultas generalistas de varios ámbitos
    - Baeldung[7] recomendado para Hibernate y todo lo relacionado con JPA
- Foros de consulta, destacando el famoso stackoverflow.com
- Canales de cursos de youtube
    - Píldoras informáticas by @pildorasinformaticas
    - Makigas: Aprender a programar by @makigas
- Inteligencia artificial
    - ChatGPT[8]

[5] Europa y la Sociedad global de la informacion. Recomendaciones al Consejo Europeo. Bruselas, 1994.
[6] W3schools: https://www.w3schools.com
[7] Baeldung: https://www.baeldung.com
[8] Bot de consulta generado por la empresa OpenAI disponible en https://openai.com/chatgpt

3.6 Conclusiones

El objetivo de este proyecto va a ser el de crear una aplicación dividida por capas totalmente independientes con un nivel mínimo de acoplamiento.

La estructura de la aplicación debería contar con una capa de presentación, una capa de lógica de negocio y una capa de acceso a datos. Estas capas deberán relacionarse mediante un patrón de diseño que guíe la estructura de todo el trabajo junto con otros patrones de diseño auxiliares que permitan estructurar niveles inferiores de abstracción. El código de la aplicación ha de seguir unas buenas prácticas facilitando la futura mantenibilidad y reutilización.

La aplicación deberá poder cumplir el concepto de multiusuario y multiperfil adaptándose así a las necesidades de un servicio sanitario estándar.

La interfaz de usuario deberá premiar siempre lo funcional frente a lo estético ya que esto debería favorecer su funcionamiento. También es deseable que la propia interfaz no tenga dependencias de servicios externos y sea capaz de ser lanzada en cualquier ordenador sin necesidad de instalaciones adicionales.

También es deseable que este proyecto no sea finalista si no escalable pudiendo tener futuras ampliaciones con funcionalidades que pudiesen facilitar servicio más global.

Una vez que ya tenemos acotada la propuesta debemos responder a la siguiente pregúnta que es ¿qué tecnologías voy a necesitar para poder desarrollarla?

Capítulo 4 -Tecnología utilizada

En este capítulo describiremos el paso de la idea a las necesidades de esta. Analizaremos que dispositivos, servicios, lenguajes de programación, qué tecnologías son necesarias para implementar las distintas capas del software, redes de comunicaciones …

Estamos en el siglo XXI y la forma de trabajar que tenemos ahora no tiene nada que ver a la que teníamos en el siglo pasado. A diferencia de hace no muchos años que únicamente teníamso equipos fijos en una ubicación determinada, actualmente existen los equipos portátiles que nos dan la opción de poder trabajar en cualquier lugar.

Otro aspecto importante es que al contar esta aplicación con un servidor de base de datos externos será necesaria también una red de comunicaciones para poder trabajar sobre él. Todo esto y otros pormenores se tratarán a continuación en este capítulo.

4.1 Hardware y servicios

En mi caso opté por montar una infraestructura que me permitiese trabajar en cualquier sitio sin importal cual fuese. Para ello necesite los siguientes elementos hardware.

- Ordenador portátil

  El cual me permitió poder programar en cualquier parte a cualquier hora de una forma eficiente y silenciosa accediendo a los datos de forma remota.

- Ordenador fijo

  Para el trabajo doméstico utilicé mi ordenador de sobremesa debido al tamaño de la pantalla y a un teclado de mejor calidad que hacía el trabajo más cómodo.

- Servidor para la BBDD

  Como la BBDD es independiente de la ubicación del programa al estar este pensado para multiusuario mi decisión fue reutilizar un ordenador de sobremesa antiguo y conectarlo a la red de mi casa. De esta forma daría igual desde ubicación estuviese programando que tendría siempre acceso a los mismos datos.

  Obviamente el servidor no tiene que estar encendido 24/7 si no se va a usar, para ello realicé una configuración de WOL [9]que garantizaba su arranque. De la misma forma me aseguraba de apagarlo al terminar el trabajo para ahorrar electricidad.

[9] WOL – Wake On Lan, servicio que permite despertar a un ordenador apagado dentro de una misma red local

4.2 Lenguaje de programación principal

Por coherencia ya que es el lenguaje de programación utilizado mayoritariamente durante el grado en la UNED decidí utilizar Java.

Java es un lenguaje de programación muy utilizado en la actualidad Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems y adquirido posteriormente por Oracle[10] en 2009. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta super computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con millones de usuarios reportados. (Java, s.f.)

Java cuenta con versión libre llamada openjdk por lo que no es necesario el pago de licencias para poder utilizarlo con fines educativos.

El lenguaje se adaptaba perfectamente a mis propósitos ya que lo podia utilizar para toda la capa de lógica, datos y presentación.

4.3 Capa de presentación / SWING

Este punto fue sorprendente para mí ya que cuando decidí que la aplicación tenía que ser de escritorio para no tener dependencia de más servicios como un servidor web, descubrí que no existían tantas formas nativas para poder hacerlo.

Finalmente encontré dos posiblidades, utilizar JavaFX o utilizar Swing. Si bien ambas opciones eran viables me decanté por Swing ya que lo había utilizado con anterioridad.

La primera capa de presentación nativa diseñada para java fue AWT[11] nacida con la necesidad de poder desarrollar GUI[12] pero rápidamente quedó claro que hacía falta una nueva versión con más funcionalidad.

En 1997 nace Swing extendiendo AWT y proporcionándole mayor funcionalidad y seriedad, fue la principal novedad de aquel año en la actualización de java que fue la 1.1.

Sus fortalezas se basan en estos puntos de vista:

+ Diseño por capas basado en capas
    - Podemos añadir elementos dentro de contenedores y de esta forma obtener un diseño a capas independientes dentro de una misma vista.
+ Muchos elementos diferentes
    - Se pueden crear diseños con muchos tipos de elementos. Menus desplegables, tablas, campos de texto, botones, etiquetas …

[10] Oracle - https://www.oracle.com/es
[11] AWT – Abstract Window Toolkit
[12] GUI – Graphic User Interface

+ Eventos asociados a elementos

    - Swing crea eventos asociados a cada elemento de forma que podemos controlar que sucede en la ventana en todo momento.

    - Son de todo tipo, desde el típico clic sobre un elemento hasta eventos de foco, ratón.

    - Suelen centrarse en un controlador que aglutina los eventos generados por una serie de ventanas asociadas a un mismo perfil, esto ayuda mucho al desarrollo en el patrón MVC.

En esta memoria tenemos multiples imágenes de la interfaz gráfica construida con Swing en el Anexo 3 Manual de usuario

4.4 Base de datos

Este paso de la arquitectura fue claro para mi ya que tenía claro desde el primer momento que la BBDD que iba a utilizar sería MySQL[13].

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web. (MySQL, s.f.)

Los motivos que me llevaron a decantarme por este servicio y no por otro fueron: su sencillez, su alto grado de implantación y documentación, su fácil integración con Java y su buena relación con casi la totalidad de ORM del mercado.

Como ya se ha explicado anteriormente para poder trabajar en cualquier ubicación el servidor MySQL se montó en un ordenador independiente, para que este servidor permitiese acceso desde otro punto hubo que hacer configuraciones adicionales que no tuvieron una gran trascendencia.

Existen otras alternativas mucho más completas y profesionales que tienen una mayor funcionalidad, pero para este trabajo no era necesario más.

Para poder crear las tablas, acceder de una forma rápida y general a los datos, mantenimiento etc suele utilizarse aplicaciones gráficas de terceros que son compatibles con muchos servidores diferentes.

Yo empleé HeidiSQL[14] que es una aplicación gratuita multiplataforma que satisface todas las necesidades para este tipo de operaciones. HeidiSQL, inicialmente conocido como MySQL-Front, es un software libre y de código abierto que permite conectarse a servidores MySQL, así como Microsoft SQL Server y PostgreSQL. (HeidiSQL, s.f.)

En la Figura 13 podemos ver una captura de pantalla de heidiSQL, concretamente la tabla Empleados de nuestra base de datos. Vemos como el programa nos muestra las tablas en la parte izquierda, las claves de referencia en la parte superior derecha y los campos de la tabla en la inferior derecha.

[13] MySQL - https://www.mysql.com
[14] HeidiSQL - https://www.heidisql.com

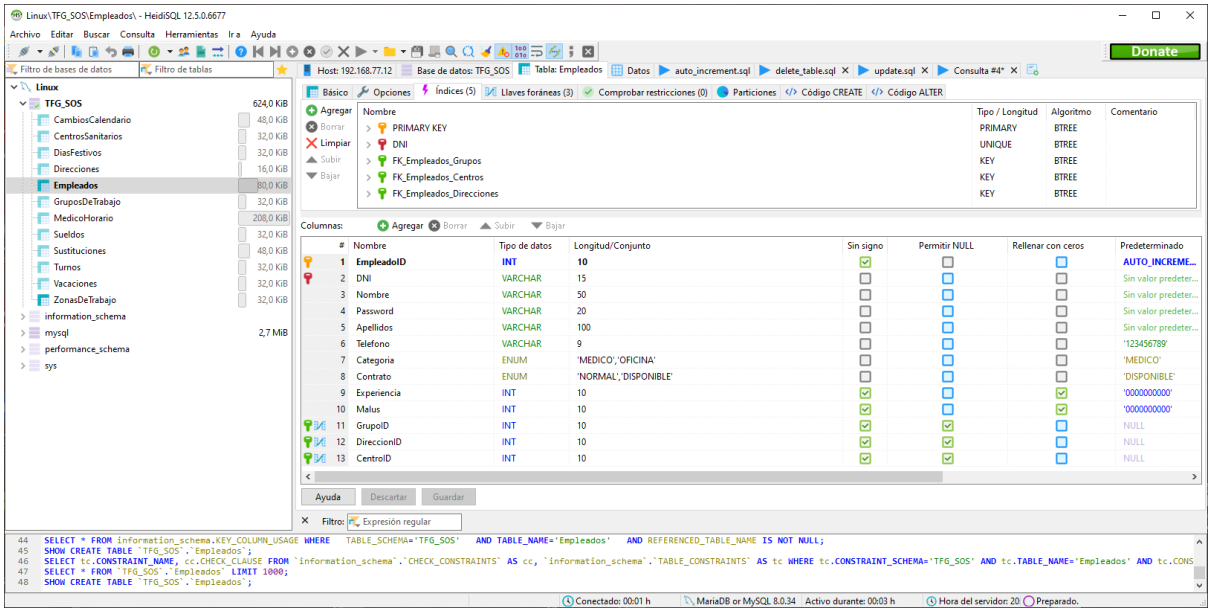

Figura 13 Imagen de la tabla Empleados en HeidiSQL

4.5 ORM / Capa de persistencia

Un ORM [15] es una herramienta que sirve para hacer de intermediario entre una base de datos, generalmente relacional y un lenguaje de programación, en este caso Java.

El ORM elegido de entre las opciones disponibles fue Hibernate[16] el cual es uno de los más famosos por su fácil integración con prácticamente todos los lenguajes y frameworks actuales.

Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos o anotaciones en los beans de las entidades que permiten establecer estas relaciones. (Hibernate, s.f.)

Las relaciones pueden ser unidireccionales o bidireccionales entre los objetos mapeados. En el caso de relaciones 1 a muchos la parte “muchos” es una estructura de datos (normalmente una lista) que se encuentra dentro del objeto de cardinalidad 1. De esta forma se puede acceder a esos datos de forma directa sin la necesidad de realizar operaciones adicionales.

Hibernate se relaciona con la base de datos utilizando un objeto de sesión. De esta forma la conexión entre el programa en ejecución y la BBDD está perfectamente gestionado pudiendo abrirse y cerrarse la conexión cuando sea necesario. De esta forma se ahorran los problemas típicos de conexiones que quedan abiertas sin cerrarse al final las operaciones necesarias.

Este tipo de acceso tiene sus pros y sus contras obligando a que el programador deba tener claro qué datos relacionados entre tablas va a necesitar de forma directa e indirecta, lo que en informática se llama acceso Eager o acceso Lazy. Explicar esto excede los objetivos de este documento por lo que animamos al lector a buscar por su cuenta información al respecto.

[15] ORM – Object Relational Mapping
[16] Hibernate - https://hibernate.org

Hibernate tiene principalmente tres formas de poder acceder a los registros de una tabla:

- Acceso mediante ID / primary key

  Es una buena práctica que cada registro en una tabla tenga una ID única para poder acceder a él. Habitualmente este campo se establece además de forma autoincremental en la base de datos. Hibernate es capaz de proporcionándole el tipo de objeto y su id recogerlo directamente de la BBDD.

- Consultas HQL

  HQL[17] es un tipo de lenguaje de consultas derivado del SQL original creado específicamente para Hibernate. La similitud es tal que cualquier persona con conocimientos medios de SQL puede entender perfectamente una consulta escrita en este lenguaje.

  La particularidad de estas consultas es la integración y traducción del objeto a la tabla que se encuentra en la base de datos. Si un buen IDE nos proporciona navegación entre las propiedades de un objeto utilizando el ".", como vemos en la Figura 14 Hibernate nos proporciona lo mismo asociándolo diréctamente con la tabla a la que apunta.

```
String hql = "SELECT DR FROM DoctorAndSchedule DR INNER JOIN WorkShift WR ON " +
        "WR.workShift_Id = DR.workShift.workShift_Id INNER JOIN Employee E ON " +
        "E.dni = DR.doctor.dni WHERE E.dni =:DNI AND WR.dateOfShift=:Date";
```

Figura 14 Consulta HQL

  En la figura podemos ver como los alias permiten una navegación dentro de las propiedades de los objetos relacionados. Vemos como por ejemplo El objeto DoctorAndSchedule tiene en su interior un objeto doctor que tiene dentro una propiedad llamada dni. Esta sintaxis no es posible en SQL original. HQL se encargará de traducir y mapear automáticamente el ejemplo del que hablamos incluso si ese registro estuviese en una tabla diferente a la consultada, esto es gracias a las anotaciones puestas en el código fuente que mapean los objetos.

  Sin duda es una de los grandes avances y utilidades que proporciiona este ORM, la capacidad de hacer consultas personalizadas de una formá más sencilla y rápida que con SQL estándar.

- Consultas criteria

  Hibernate también permite realizar consultas mediante el objeto Criteria. Mediante la creación de unos criterios determinados se puede acceder a todos los datos utilizando el mapeo relacional con las anotaciones antes citadas.

[17] HQL – Hibernate Query Language

4.6 Redes de comunicaciones

La necesidad de una base de datos centralizada para que la aplicación pueda cumplir con la función multiusuario y multiperfil obliga a que exista una red de comunicación entre la app y este servicio de datos.

Para poder establecer la conexión eran necesarias dos cosas. Una red de comunicación propiamente dicha y también una metodología de conexión las cuales explicamos a continuación.

- Redes de comunicación

  Al estar en localizaciones de trabajo diferentes necesitamos, obviamente, disponer de redes que comuniquen nuestros dispositivos. Si bien en mi domicilio contaba con una red FTTH estándar como en la gran mayoría de las residencias del país, cuando estaba fuera de mi domicilio y por seguridad, siempre utilizaba la conexión 4g compartida de mi smartphone con mi portátil. De esta forma y en conjunto con la VPN la comunicación era viable en cualquier ubicación sin restricciones.

- VPN

  Para poder conectar con la BBDD de forma remota y fuera de la red local había dos opciones:

  Una primera opción era abrir el puerto correspondiente en el router y gestionarlo directamente, pero esto necesitaría cifrado adicional en las comunicaciones y configuración adicional en el servidor de la BBDD para permitir acceso desde fuera de la red local por lo que se descartó esta opción.

  Una segunda opción era montar una VPN[18]. De esta forma podríamos conectarnos a la exterior desde el exterior sin necesidad de exponer nuestra red doméstica. Para ello necesitamos obviamente un servidor de VPN, pero afortunadamente mi router doméstico lo provee de forma automática, así como un servicio de DDNS[19]. De esta forma cubrimos todos los posibles aspectos de conectividad remota y se garantiza el acceso seguro a los mismos datos del servidor de datos.

[18] VPN – Virtual Private Network
[19] DDNS – Dynamic Domain Name System o símplemente Dynamic DNS

4.7 Control de versiones

Un sistema de control de versiones es una herramienta, local o externa, que permite crear una copia de seguridad de un momento concreto del desarrollo de forma manual. Si el alojamiento de estas copias se hace en la nube nos veremos beneficiados de las ventajas de ello y no tener que transportar copias "a la antigua usanza" con dispositivos de almacenamiento.

Permite también una segmentación del trabajo entre muchas personas permitiendo crear "ramas" de desarrollo paralelas que posteriormente se fusionarán con la rama de trabajo principal.

Como valoración personal afirmo que la existencia de este tipo de herramientas me ha cambiado la vida a la hora de trabajar. Como una imagen vale más que mil palabras, la Figura 15 nos muestra cómo era la vida y las copias de seguridad antes de conocer herramientas de control de versiones

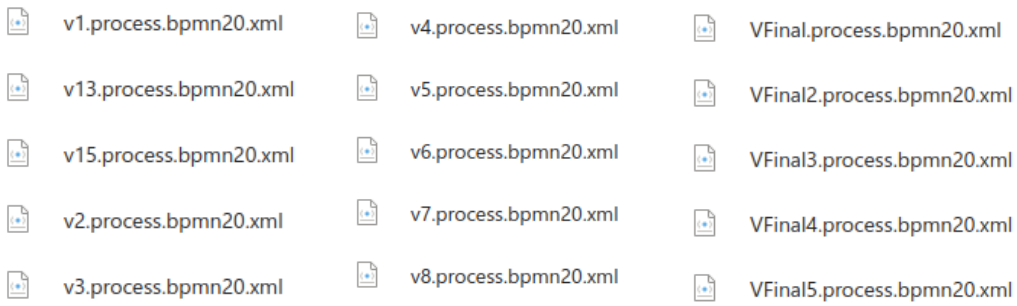

Figura 15 Imágen de carpetas, ejemplo sin control de versiones

En la Figura 15 vemos una carpeta real dedicada a un trabajo sobre una asignatura en el inicio del grado universitario. Tras descubrir los sistemas de control de versiones puedo decir que su enseñanza no debería limitarse al campo informático o al desarrollo de proyectos, debería enseñarse en los institutos a una tempranada edad.

Existen varias herramientas de este tipo en el mercado, aunque la que yo utilicé, que es sin duda también la más popular es GIT. Para poder tener acceso a este control de versiones en cualquier parte utilicé GitHub que es un repositorio online que permite vincular GIT directamente.

4.8 Conclusiones

En este capítulo hemos hablado sobre las tecnologías, hardare y servicios que vamos a necesitar para poder desarrollar la aplicación. Una vez que tenemos esto claro, al menos en una gran parte ya que todo está abierto a posibles modificaciones, podemos centrarnos ahora planear nuestro sistema y rutina de trabajo para no caer en la anarquía y la improvisación.

Capítulo 5 - Metodología de desarrollo

En este capítulo vamos a describir que sistema de trabajo vamos a utilizar para diseñar y posteriormente implementar el proyecto. Si bien el refrán "cada maestrillo tiene su librillo" es algo extendido, en el mundo informático y sobre todo en la arquitectura de software es un error demasiado típico. El seguir una metología de desarrollo y un conjunto de buenas prácticas no ayudará en multiples aspectos como:

- Rutina de trabajo
- Estandarización
- Orden y clasificación de componentes jerárquico y relacionado
- Utilización de buenas prácticas
- Fácil mantenibiliad y reutilización
- Uso eficiente del tiempo
- Control de errores
- Documentación organizada
- Gestión de recursos

Estas son solo una breve lista de los beneficios que tiene usar una buena metodología y un buen sistema de trabajo, hay muchísimas más, pero creo que ha quedado patente que su uso es casi obligatorio.

Existen muchas metodologías para el desarrollo de proyectos por lo que encontrar un sistema que se adapte a la perfección a un TFG universitario como el mío ha de considerarse como una guía y no como un dogma.

Dentro de los distintos tipos de TFG, este es uno de los autodenominados *"proyectos propios",* en el cual el alumno lo presenta a un tutor consensuándolo para un desarrollo del proyecto en base a la

Idea inicial del alumno.

Para la elección de una metodología (o aproximación a la misma) ha de tenerse en cuenta varios factores:

- Conocimientos previos sobre metodologías de desarrollo
- Acceso formativo a metodologías de desarrollo
- Adecuación de una metodología de desarrollo al proyecto valorando:
    - Dimensión y tamaño
    - Módulos y arquitectura del sistema

Si bien como se ha dicho antes no hay que tomar las metodologías como un dogma, me he inspirado principalmente en una, aunque haya podido coger elementos de otras. La metodología de software en la que me he inspirado ha sido el "Proceso Unificado de Desarrollo de Software" (Jacobson, Booch, & Rumbaugh, 1999)

5.1 Pilares de la metodología

Esta metodología se basa en tres pilares que se adecuan perfectamente a este proyecto, que son:

- Apoyado principalmente en los casos de uso
- Centrado en la arquitectura
- Iterativo e incremental

5.1.1 Casos de uso

Una práctica muy común a la vez que eficiente es desarrollar teniendo en cuenta los requisitos funcionales del producto. Estos requisitos funcionales vienen documentados en los casos de uso.

Los casos de uso describen en forma de lista de acciones e interacciones el comportamiento del sistema estudiado desde el punto de vista de los actores. Esta definición concuerda plenamente con el objetivo de este TFG ya que es un sistema que pretende automatizar un sistema laboral, en el cual, el centro son precisamente personas, es decir, los actores representados en los casos de uso mediante sus distintos roles.

De esta forma los casos de uso serán una herramienta imprescindible para guiar todo el proceso de desarrollo (análisis, diseño, implementación, pruebas …) constituyendo un hilo conductor mediante el cual se avanza en el desarrollo de este producto software.

Los casos de uso atravesarán un ciclo de vida en el cual, como no podía ser de otra manera, cada nueva fase se apoyará en la anterior. En el caso de este TFG nuestro caso de uso viajará por las siguientes fases siguiendo las recomendaciones del PUD[20]

5.3.1.1 Fase de Requisitos

Se programan múltiples entrevistas y reuniones con personas que podrían ser usuarios del sistema de una u otra forma. Tras recopilar esa información se pueden considerar los requisitos del sistema y los casos de uso que vamos a tener, descritos en el apartado anterior. También quedan definidos los usuarios del sistema.

[20] Proceso Unificado de Desarrollo de Software

5.3.1.2 Fase de Análisis

En esta fase definimos conceptualmente las tres *"clases de análisis"*:

- *Boundary Classes*
  - Representa un elemento de la interfaz del usuario, es decir, una clase que interactúa con el mundo exterior del sistema.
  - Se encargan de manejar las entradas y salidas del sistema, y a menudo se utilizan para validar la entrada del usuario y presentar información al usuario de manera comprensible.
  - En otras metodologías son llamadas *"view classes"*
- *Control Classes*
  - Representan clases que manejan la lógica del sistema.
  - Coordinan la interacción entre las boundary classes y las entity classes.
  - Utilizadas para implementar la funcionalidad del sistema.
- Entity Classes
  - Representan objetos y entidades del mundo real
  - Habitualmente son almacenados en el sistema
  - Son utilizadas para hacer operaciones con los valores de sus propiedades

La Figura 16 muestra las tres clases existentes que tienen la siguiente representación en el diagrama.

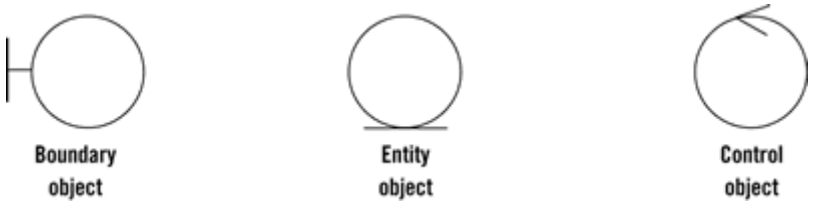

Figura 16 Tipos de clases en el sistema

Una vez identificadas estas clases se esboza de forma general el flujo secuencial del caso de uso. Para ello utilizamos un diagrama de colaboración entre entidades.

5.3.1.3 Fase de diseño

Una vez que hemos esbozado en el diagrama de colaboración de forma preliminar el flujo que ha de seguir ha llegado el momento de detallar ese flujo.

Para llevar a cabo una trazabilidad completa del flujo de proceso utilizamos un diagrama de secuencia, el cual especifica todas las interacciones entre el actor y los diferentes artefactos del sistema.

5.3.1.4 Fase de implementación

Ahora nos toca codificar el diagrama de secuencia. Si el diagrama está bien elaborado el paso del diagrama al IDE será casi lineal. En este paso es cuando queda patente la importancia del uso de una metodología de diseño adecuada.

5.3.1.5 Fase de documentación

Esta fase será opcional y no viene como tal en el manual general del PUD, sin embargo, considero que es imprescindible.

En el caso de que tengamos que haber investigado sobre situaciones acerca de cómo realizar una funcionalidad, utilizar una herramienta que desconocíamos, configurar algo relacionado con el punto en el que estamos … Esto debería documentarse.

El proceso de documentación debería ya no ser parte de este proyecto si no de la vida de cualquier ingeniero. En nuestra vida vamos a tener, con absoluta seguridad, que enfrentarnos a situaciones que se verán espaciadas en el tiempo y hayamos olvidado cómo resolver.

5.3.1.6 Fase de pruebas

No es posible verificar que un sistema funciona si no se prueba. Existen principalmente dos formas de realizar pruebas de cada nueva funcionalidad implementada en un software.

- La primera alternativa es dejar la fase de pruebas para el final y agrupar todas las pruebas del programa, de esa forma la fase de pruebas tiene un carácter independiente debiendo planear el tiempo de una forma global para esta fase.
  Es recomendable cuando el software es desarrollado por muchos participantes diferentes y también se recomienda que no sean las mismas personas que desarrollan una parte las que la prueban evitando sesgos debido a que los testers “externos” se les pueden ocurrir situaciones que no se les ha ocurrido a los desarrolladores.
  La principal desventaja es que, si se detectan problemas, dado que el personal que lo detecta es ajeno al desarrollo, tiene que notificarlo a los desarrolladores que tendrán que reajustar el problema con la pertinente pérdida de tiempo en los procesos de comunicación.

- La segunda alternativa, y la empleada en este proyecto es realizar las pruebas una vez que se termina esa parte. De esta forma nos aseguramos de que, si una futura funcionalidad se apoya en una anterior esta funcionará, aunque sea posible que requiera ajustes posteriores. Esta alternativa se ajusta mejor a un desarrollo individual como este proyecto y proporciona una mejor sensación de trabajo enfocado a la funcionalidad ya que con las pruebas concluimos una y pasamos a la siguiente.

Una prueba se dividirá en los siguientes conceptos:

- Funcionalidades que probar
- Entradas al sistema
- Resultados esperados
- Resultados obtenidos
- Confirmación de funcionamiento

5.1.2 Centrado en la arquitectura

La arquitectura permite representar los aspectos estáticos y dinámicos del sistema a desarrollar, debiendo tener en cuenta que requisitos tecnológicos existen para el sistema tales como: Plataforma de despliegue, ámbito de utilización, multiplataforma, escalabilidad, portabilidad, disponibilidad …

La arquitectura está íntimamente relacionada, como no puede ser de otra forma, con los casos de uso avanzando ambos de forma conjunta e integrada ya que un caso de uso ha de encajar y ser compatible forzosamente con la arquitectura planteada. A su vez la arquitectura ha de permitir desarrollar los casos de uso de la forma más sencilla y eficiente posible.

Recordamos, como dijimos en el capítulo anterior que la arquitectura elegida para este TFG estará dividida en tres partes que servirán para desarrollar tres capas colaborando en el funcionamiento del patrón genérico MVC[21] apoyado en un patrón DAO[22]:

- Back-end
  - El SGBD[23] utilizado ha sido MySQL por su simplicidad e idoneidad para este proyecto. Se utiliza como un sevicio externo y deslocalizado del resto del sistema.
  - El ORM utilizado en este caso será Hibenate que es uno de los más empleados en la actualidad.
- Lógica de negocio

  Implementada en Java como lenguaje de programación, elegido este por ser el lenguaje predominante en el Grado Universitario de la UNED

- Front-end

  El sistema visual de presentación elegido ha sido Swing que es el descendiente del primitivo AWT[24] que fue el primer sistema de representación gráfica a modo de ventanas que tuvo java.

[21] MVC, patrón de desarrollo basado en la terna Modelo – Vista – Controlador
[22] DAO, Patrón de diseño “Data Access Object”
[23] SGBD, Sistema Gestor de Bases de Datos
[24] AWT, Abstract Window Toolkit

5.1.3 Iterativo e incremental

Esta metodología de desarrollo propone dividir el proyecto en fracciones más pequeñas del mismo. Cada una de estas fracciones se denomina iteración. Pudiendo establecerse una jerarquía de iteraciones habiendo otras de un nivel inferior que serán sub-iteraciones de las primeras.

Estas iteraciones serán planificadas al principio del proyecto y sobre todo responderán al desarrollo de las tres grandes capas del proyecto BackEnd, lógica y FrontEnd.

Cada iteración identificará sus propios riesgos y deberá resolverlos antes de pasar a lo próxima iteración.

5.2 Fases de desarrollo de la metodología

Investigando sobre diferentes metodologías de desarrollo fue la división por fases de esta la que hizo que fuese la elegida. Al leer las fases de esta metodología vi que encajaba en un porcentaje muy elevado. Describamos ahora las fases de desarrollo

5.2.1 Inicio

- Se establece una arquitectura base para el proyecto.
- Se conforman los distintos perfiles de usuario que participarán en el sistema
- Aparecen las historias de usuario que esbozan lo que serán los casos de uso en la próxima fase.
- Creación de un plan de proyecto que deberá seguirse durante todo el desarrollo.

5.2.2 Elaboración

- Se especifican todos los casos de uso que conformarán los requisitos funcionales del sistema
- En este momento la elaboración de un calendario de hitos empieza a ser importante para establecer tiempos

5.2.3 Construcción

- Se procede a crear el producto a partir de las especificaciones de la fase de elaboración.
- Se añaden los requisitos de arquitectura que hayan podido surgir
- Al final de esta fase ha de ejecutarse una batería de pruebas que garantice la estabilidad del sistema

5.2.4 Transición

- En esta fase el producto pasa a la "fase beta" y está lista para pasar el proceso de pruebas por un conjunto de usuarios
- Los usuarios han de rellenar informes y sobre todo sugerir posibles mejoras. Es muy importante estudiar y valorar las posibles dificultades trasmitidas por ellos.
- Se resuelven las últimas incidencias y se incluyen las posibles mejoras sugeridas

5.2.5 Lanzamiento

- Finalmente se crea la documentación necesaria para la presentación del producto final.
- El producto está listo para su puesta en producción

5.3 Flujos de trabajo

Cada una de las fases del del PUD se divide en iteraciones. En la Figura 17 se muestra como cada una de estas iteraciones se subdivide en flujos de trabajo

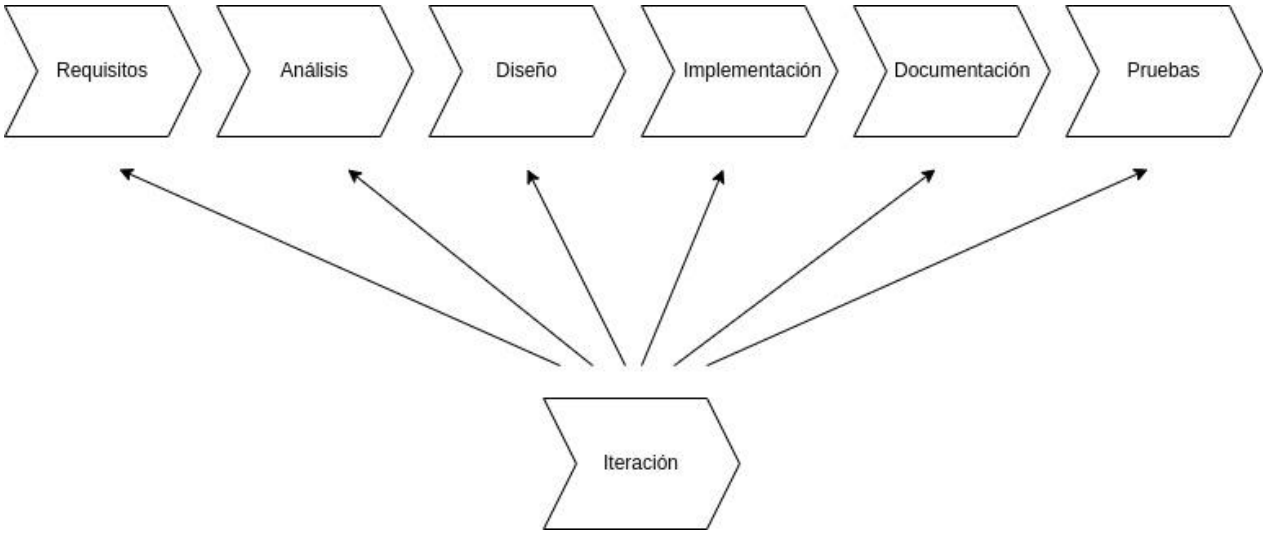

Figura 17 Flujo de trabajo de una iteración

Fuente: (Piattini Velthuis, García Rubio, García Rodríguez de Guzmán, & Pino, 2018). *Figura 10.8 (Gráfico) En Libro. Adaptación propia*

5.3.1 Requisitos

- Comenzando en un punto determinado se identifican que elementos de entrada ya existentes o no son necesarios para empezar la tarea.
- En caso de que un requisito no exista desencadenaría otro flujo de trabajo que tendría prioridad sobre esta tarea en la que nos encontramos.
- Se capturan los requisitos funcionales y no funcionales

5.3.2 Análisis

- El objetivo ha de ser describir los requisitos con mayor nivel de detalle.
- El punto principal que debe determinarse aquí es la interacción entre los posibles objetos colaboradores.
- Es importante seguir convenciones comunes en el análisis que ayuden a estandarizarlos.
- Necesitaremos tantos análisis como requisitos tengamos si bien es cierto que colaborarán entre ellos

5.3.3 Diseño

- Llega el momento de modelar los requisitos
- Se centra en establecer una estructura estable que será implementada posteriormente
- Debe ser clara y comprensible dando poco margen para la interpretación ya que al pasar a implementación debe poder seguirse idealmente de forma casi lineal.
- La documentación generada en este punto es con diferencia la más importante ya que es el guión a seguir posteriormente
- Esta metodología utiliza UML[25] como estándar para la documentación

5.3.4 Implementación

- Partiendo de los diagramas y documentos generados en la fase de diseño se procede a codificar.
- El sistema se va implementando gradualmente de forma incremental.
- En este punto estamos a caballo entre las fases de “elaboración” y “construcción” aunque también habrá que volver a ella en la fase de transición para solucionar problemas o añadir funcionalidades.

[25] UML – Unified Modeling Languaje. Lenguaje que permite especificar y construir un software mediante diagramas y elementos gráficos.

5.3.5 Documentación

- Este flujo de trabajo no existe como tal dentro de la metodología primigenia, es un añadido personal.
- Es un flujo opcional el cual será necesario cuando se realice algo por primera vez y haya requerido formación previa.
- La documentación no tiene por qué pertenecer siquiera al propio proyecto si no que quedará disponible para el personal implicado para un uso posterior.

5.3.6 Pruebas

- Sirve para verificar el resultado de la implementación
- Se prueba cada elemento individualmente y en conjunto
- Se diseñan, implementan y documentan los casos de prueba

5.4 Conclusiones

En este capítulo hemos decidido metodología de trabajo, PUD (Proceso Unificado de Software). Hemos explicados sus componentes principales, la interacción entre ellos, las distintas fases en las que se divide y los flujos de trabajo que contienen.

Una vez decidido ya lo que queremos hacer y cómo vamos a trabajarlo es el momento de que prosigamos metiéndonos de lleno en la siguiente fase, el diseño, el cual abordaremos en el próximo capítulo.

Capítulo 6 - Diseño de la aplicación

En este capítulo vamos a dedicarnos a diseñar y explicar las partes que van a conformar nuestra aplicación. La metodología de trabajo elegida y explicada en el capítulo anterior nos servirá como guía comenzando por un modelado del dominio mediente un diagrama E/R que nos servirá, para construir posteriormente las tablas de nuestra BBDD las cuales se han mostrado anteriormente en la Figura 18.

Una vez tengamos el dominio modelado describirémos los casos de uso que describirán las distintas funcionalidades de nuestra aplicación con mayor precisión. Y tras esto el sistema estará diseñado y listo para pasar a la siguiente fase.

6.1 Modelo del dominio

El modelado del dominio es la primera parte y quizás la más importante de toda la fase de diseño. Para modelarlo vamos a basarnos en un diagrama entidad relación que, como su propio nombre indica representa una serie de entidades que son cojetos o conceptos del mundo real con existencia propia y la relación que estas tienen entre si.

Diseñar correctamente este diagrama es algo fundamental debido a que hay dos componentes que se basan directamente en él que son las tablas de nuestra futura base de datos y los POJOS[26] de nuestro futuro código fuente. Cada una de nuestras tablas / pojos estarán representadas por una entidad y las relaciones entre estas generarán respuestas de distintos tipos.

El proceso de asociación y transformación de las tablas y sus relaciones de la BBDD al código fuente se llama **mapear** y para ello Java (que es el lenguaje utilizado en este proyecto) utiliza JPA[27]. Nosotros, como ya explicó en el capitulo anterior hemos utiliza Hibernate como implementación de JPA.

[26] POJO – Objeto de una clase simple que no tienen dependencias de otras más complejas exceptuando la herencia

[27] JPA – Java Persistence Api

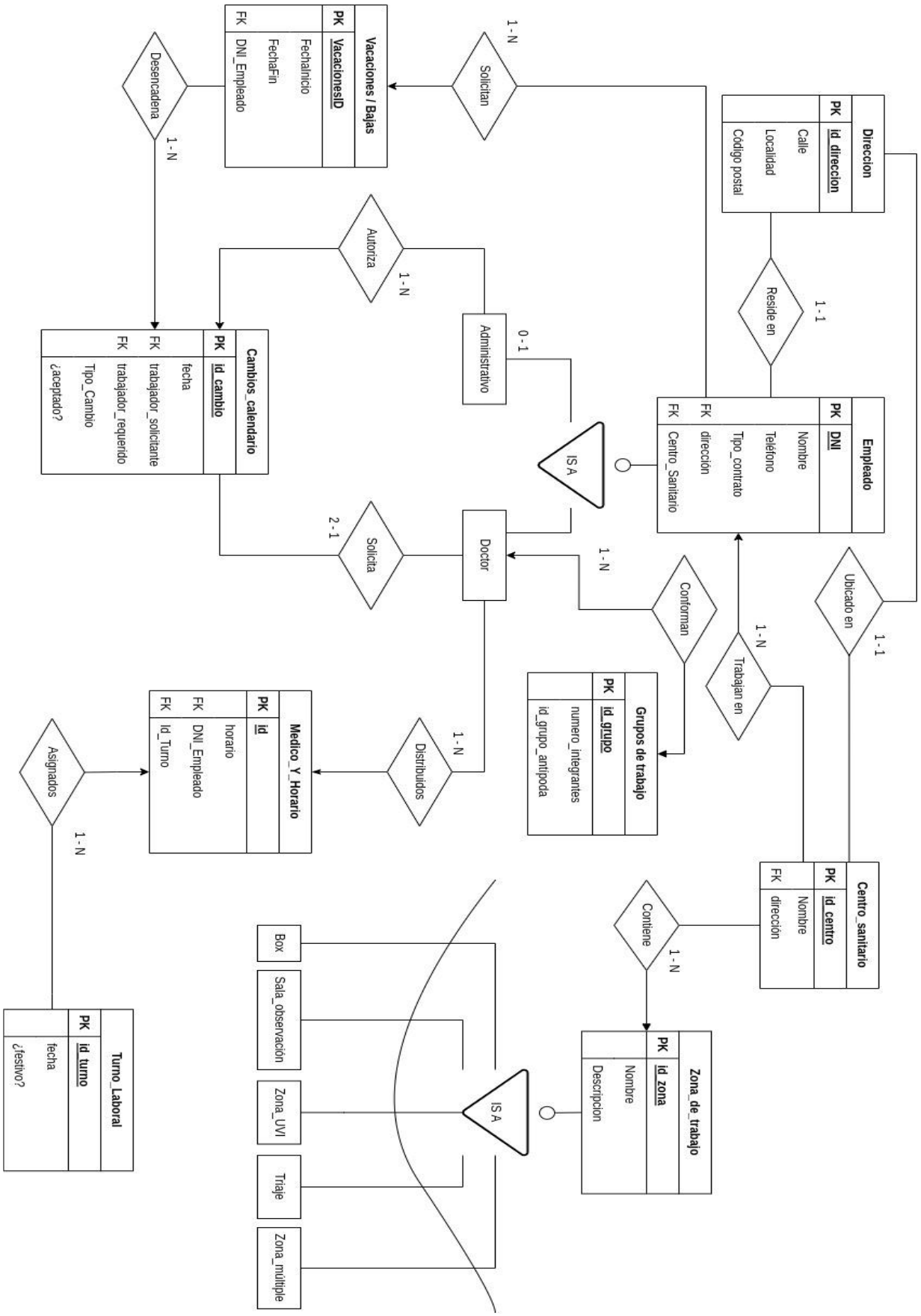

Figura 18 Diagrama entidad / relación del dominio del sistema

En la Figura 18 podemos ver el diagrama entidad relación del proyecto. No hay mucho que explicar ya que las entidades son lo suficientemente descriptivas, pero me voy a detener a explicar algunas de ellas.

La entidad Turno_Laboral es la representación general de un día de trabajo con fecha concreta, por ejemplo, el 14 de agosto del año 2024. Sin embargo, un día laborable en nuestro contexto se divide en tres posibles turnos: Mañana, tarde y guardia.

La entidad Medico_Horario cubre la asociación de los turnos al médico en cuestión. De esta forma y en función si el día es festivo, habrá un número determinado de médicos que estén de mañana, otros tantos de tarde y otros tantos de guardia.

La entidad cambios_calendario es una notificación en el sistema generada por los médicos y que será posteriormente aceptada o rechazada por el personal de gestión.

Podemos generar estos cambios en el calendario desde tres posibles escenarios con las siguientes particularidades:

- Intercambio entre compañeros
  - Intercambio guardia de días diferentes. Conlleva el cambio de los días de libranza post guardia
  - Intercambio de mañana por tarde. Cambio que sucede en el mismo día. No conlleva más acciones que el cambio directo
- Vacaciones
  - Se contrata a un externo para sustituirlas trasladándole a él todos los días de vacaciones directamente.
  - No se sustituye el turno. Se acepta que hay un miembro menos del personal. Esta situación es aceptable bajo ciertas circunstancias
  - Se sustituyen los días de trabajo con compañeros haciéndoles cambiar los turnos de la siguiente manera:
    - Las mañanas del empleado en vacaciones no se sustituyen, habitualmente hay personal suficiente por lo que no es necesario.
    - Las tardes son sustituidas moviendo a este turno uno de los que están de mañana ese día. El cambio se realiza en función de su *malus.*
    - Las guardias son sustituidas por un miembro de su grupo antípoda en función del indicador *malus*.
  - Se deniegan las vacaciones por decisión del servicio. Obviamente no hay cambios y el empleado debe acudir a trabajar según su turno original.

- Baja laboral

  Este escenario es igual que el anterior en el que tratamos las vacaciones con una única particularidad, el personal administrativo no puede denegar una baja laboral. Por lo demás se puede aplicar lo especificado en el apartado anterior.

6.2 Casos de uso

Una vez que tenemos definidos los usuarios y la arquitectura general del sistema hemos de pasar a los requisitos funcionales de la misma. Estos requisitos se ven representado por los casos de uso.

6.2.1 Elección de los casos de uso

A partir de las historias de usuario y mis propias ideas sobre cómo debía ser la aplicación, se desarrolla una lista de qué casos de uso deberíamos construir.

Cabe destacar la colaboración de múltiples sanitarios que han dado forma a esta aplicación al enriquecerla con sus experiencias personales de cómo se lleva a cabo de forma manual o semiautomatizada en sus centros de trabajo. Sin estas colaboraciones para recoger información, este TFG hubiese sido mucho más complicado de desarrollar.

Los casos de uso deberán representar, además de las típicas CRUD, las siguientes funcionalidades:

- Identificarse en el sistema (login)
- Elaboración de turno de trabajo
- Constituir grupos de trabajo
- Alterar grupos de trabajo
- Cambio de turno entre compañeros
- Gestión de vacaciones de usuario
- Gestión de bajas de usuario
- Creación y modificación de notificaciones en el sistema

Estas funcionalidades representan el motivo de la existencia de esta aplicación, los dos primeros establecen un primer calendario y los demás representan cambios y modificaciones en él.

6.2.2 Listado de Casos de Uso

Comenzamos con el caso de uso de crear usuario expuesto en la Figura 19, este caso de uso da de alta el personal dentro del sistema. Aclaramos que en este caso de uso lo que sucede es que un administrativo da de alta un médico, los aministrativos solo pueden ser dados de alta por el Sysadmin

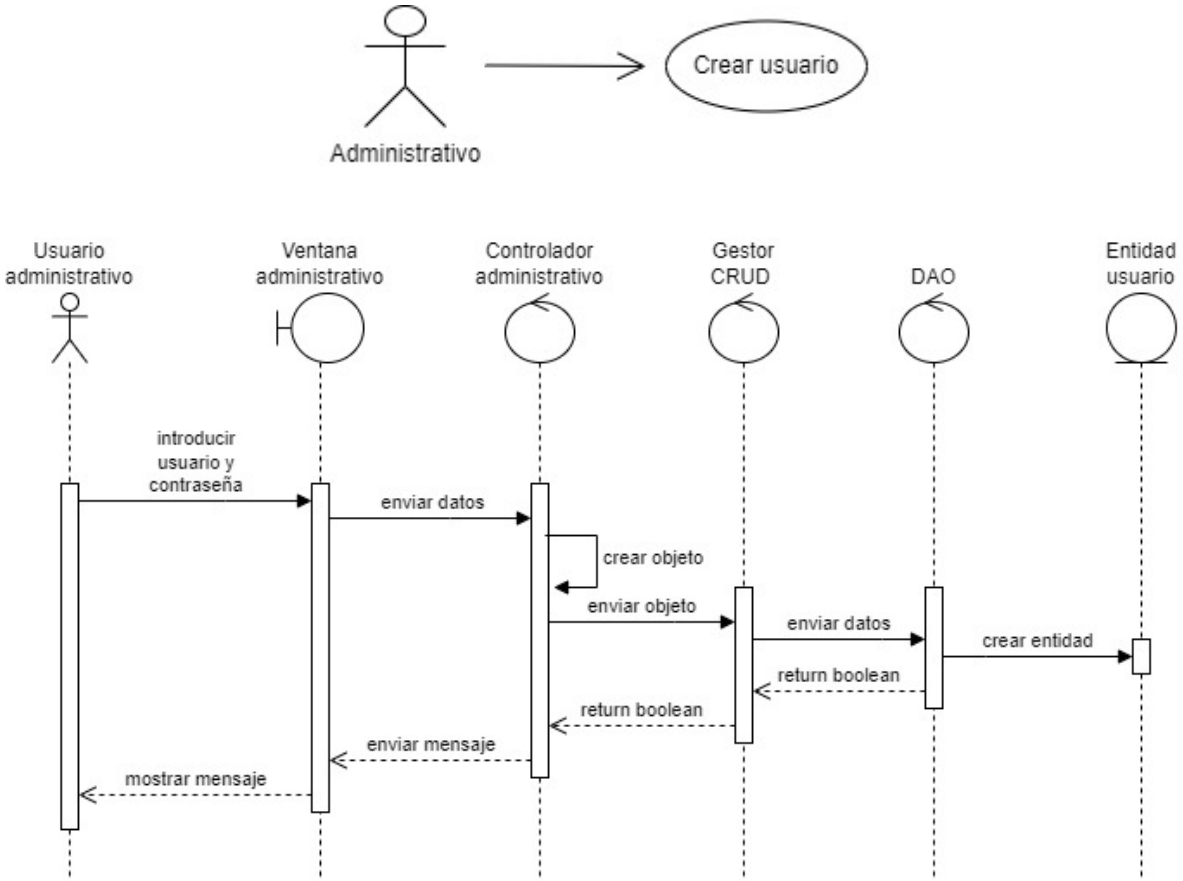

Figura 19 Diagrama de secuencia, crear usuario

Nombre	*Crear usuario*
Descripción	Crea un usuario y lo añade a la BBDD
Participantes	Usuario
Precondiciones	1. El usuario ha de estar logueado 2. La BBDD ha de funcionar correctamente 3. El usuario pertenece a un centro sanitario 4. No puede existir una instancia igual previamente
Flujo Normal	
1	Insertar los datos en el formulario de datos
2	Enviar los datos al gestor de datos
3	Recibir resultado de la operación mediante un boolean
Postcondiciones	
1	El usuario ha sido informado del resultado de la operación
2	El sistema ha de seguir funcionando en cualquier caso

Siguiente caso de uso, leer usuario. La descripción de la Figura 20 nos muestra los pasos para poder leer un usuario del sistema.

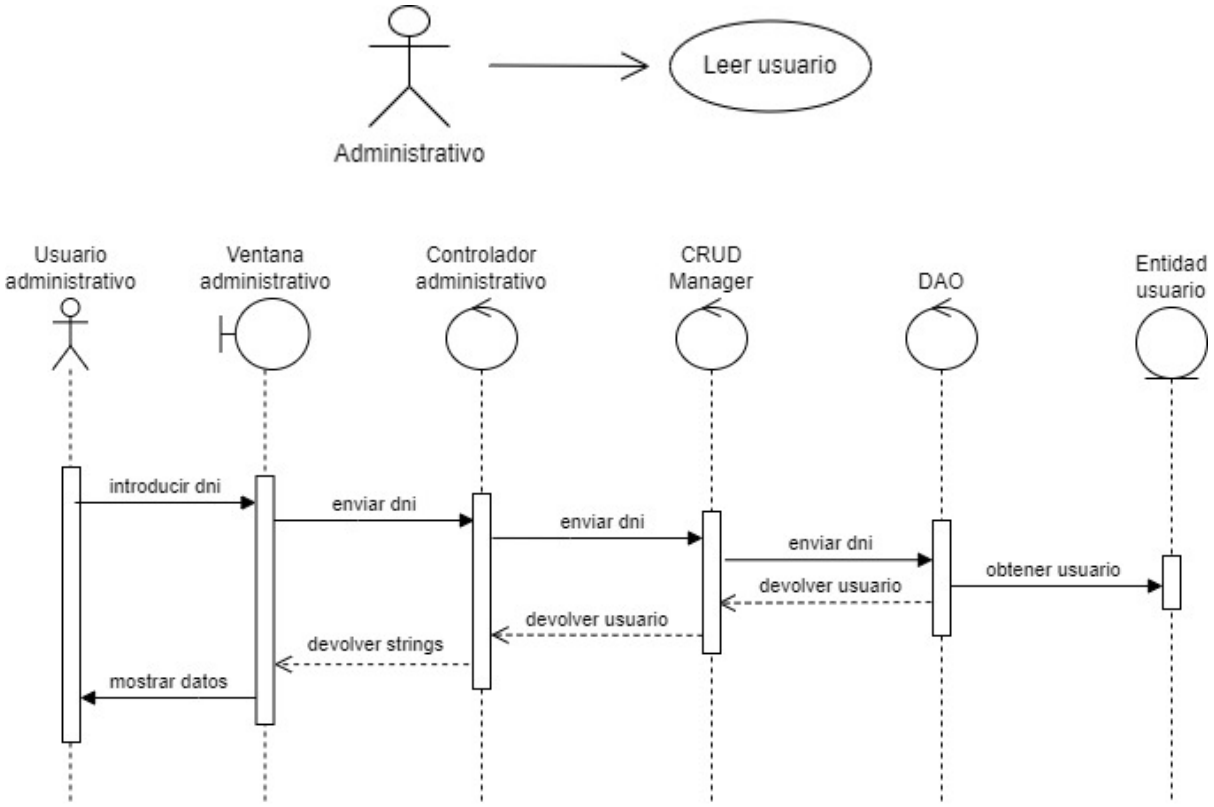

Figura 20 Diagrama de secuencia, READ

Nombre	*Leer usuario*
Descripción	Pide un usuario y lo obtiene de la BBDD
Participantes	Usuario
Precondiciones	1. El usuario ha de estar logeado 2. La BBDD ha de funcionar correctamente 3. El usuario pertenece a un centro sanitario
Flujo Normal	
1	Insertar los valores en los campos de búsqueda
2	Envíar los datos al gestor
3	Recibir el elemento o null
Postcondiciones	
1	El usuario ha sido informado del resultado de la operación
2	El sistema ha de seguir funcionando en cualquier caso

Le toca el turno a actualizar usuario, en la Figura 21 podemos ver como tras obtener el usuario previamente se modifican los valores del usuario y se almacenan nuevamente.

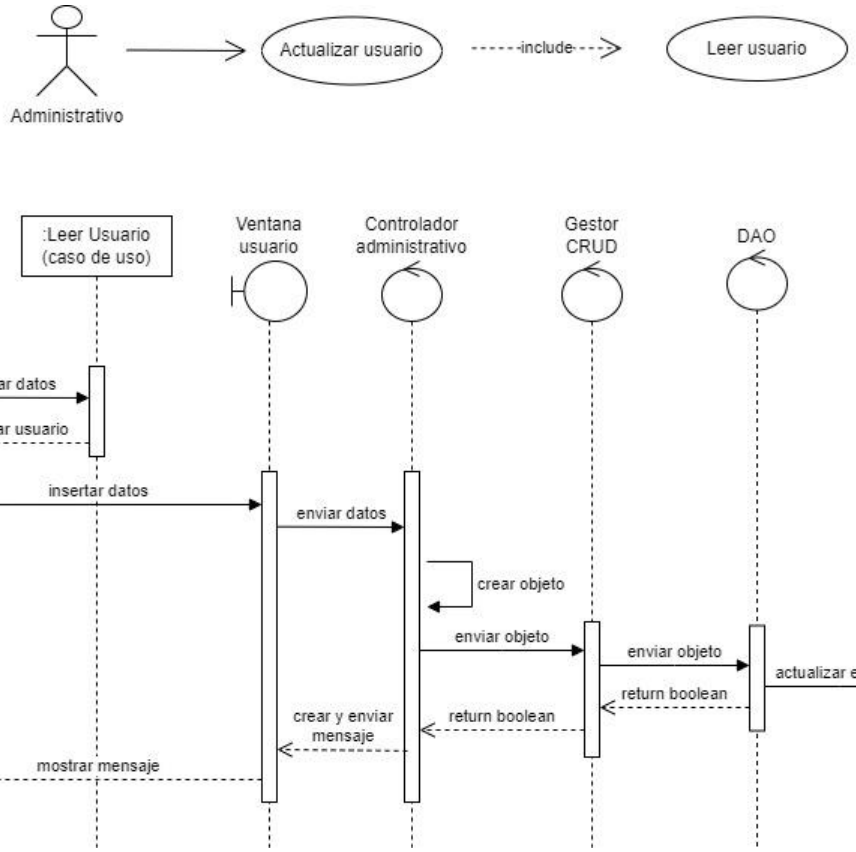

Figura 21 Diagrama de secuencia, UPDATE

Nombre	*Actualizar usuario*
Descripción	Actualiza los datos de un usuario existente en la BBDD
Participantes	Usuario
Casos de uso externos	Leer usuario
Precondiciones	1. El usuario ha de estar logeado 2. La BBDD ha de funcionar correctamente 3. El usuario ha de pertenecer a un centro sanitario 4. El objeto por actualizar ha de existir en la BBDD
Flujo Normal	
1	Insertar valor de búsqueda
2	Se envían los datos al gestor
3	Se muestran los datos de la entidad completa al usuario
4	El usuario modifica la entidad
5	Se envían los datos actualizados al gestor
6	Recibir confirmación de la actualización
Postcondiciones	
1	El usuario ha sido informado del resultado de la operación
2	El sistema ha de seguir funcionando en todo caso

El siguiente caso de uso es el de eliminar un usuario del sistema. La Figura 22 ilustra como con el envío del dni es suficiente para realizar las todas las operaciones necesarias.

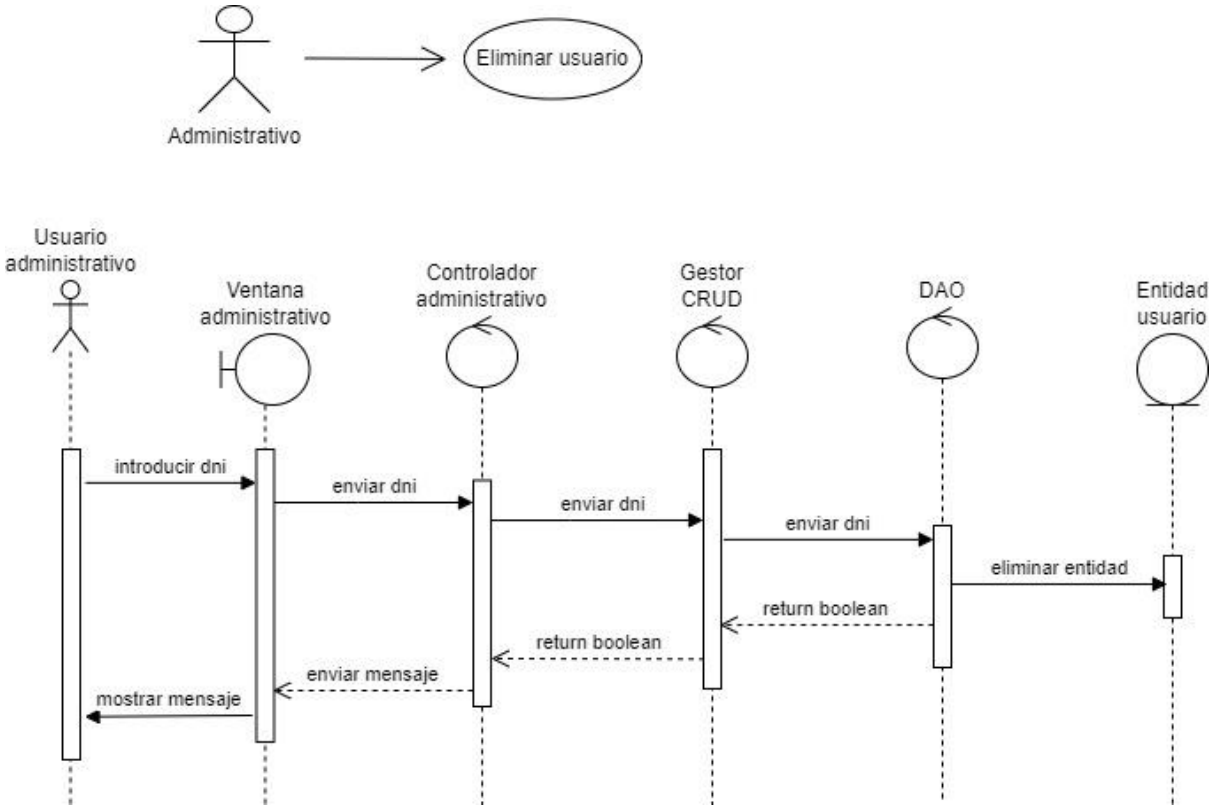

Figura 22 Diagrama de secuencia, DELETE

Nombre	*Eliminar usuario*
Descripción	Elimina un usuario existente en la BBDD
Participantes	Usuario
Precondiciones	1. El usuario ha de estar logeado 2. La BBDD ha de funcionar correctamente 3. El usuario ha de pertenecer a un centro sanitario
Flujo Normal	
1	Insertar el valor de búsqueda
2	Enviar los datos al gestor
3	Recibir resultado de la operación
Postcondiciones	
1	El usuario ha sido informado del resultado de la operación
2	El sistema ha de seguir funcionando en todo caso

El caso de uso de Login permite identificar a los usuarios en el sistema. Es un caso de uso complejo que tiene la particularidad de generar ventanas y controladores dinámicamente en función del perfil de usuario registrado en el sistema. La Figura 23 describe todo el proceso

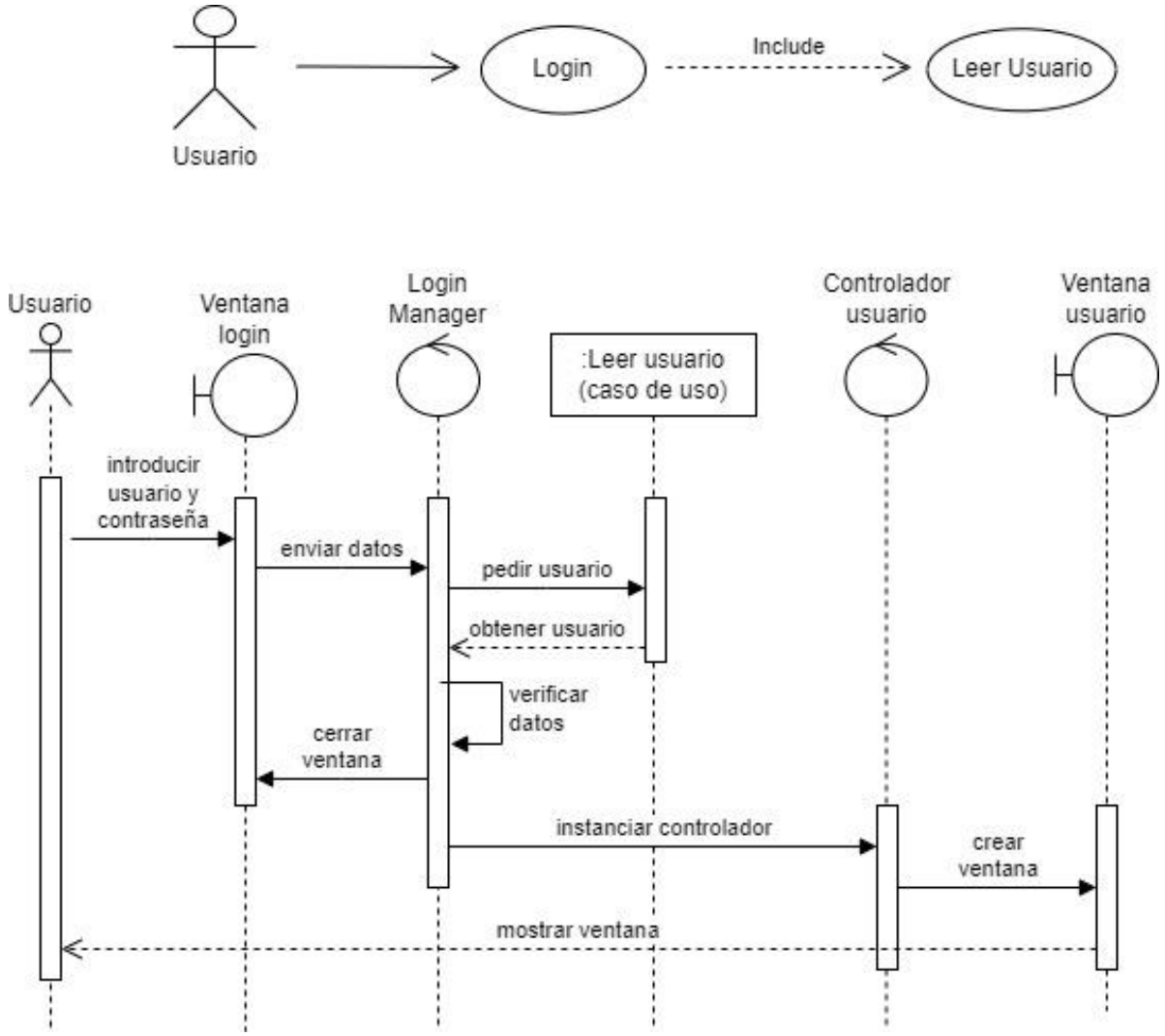

Figura 23 Diagrama de secuencia, LOGIN

En la próxima página continuaremos con la descripción pormenorizada del caso de uso.

Nombre	*Login*
Descripción	Registra al usuario dentro del sistema
Participantes	Usuario
Casos de uso externos	Leer Usuario
Precondiciones	1. La BBDD ha de funcionar correctamente
Flujo Normal	
1	El usuario inserta su id y contraseña en la ventana
2	La ventana envía los datos al login mánager
3	El login mánager obtiene el usuario y verifica su identidad
4	El login mánager destruye la ventana de login
5	El login mánager crea el controlador concreto del usuario en función de su perfil
6	El controlador crea la ventana de usuario
7	El usuario visualiza su ventana principal
Flujos Alternos	
3.A	¿El usuario y contraseña son correctos?
3.A.1	SI. Se continúa el flujo normal
3.A.2	NO. Se muestra mensaje de error y se vuelve a la pantalla de login
Postcondiciones	El sistema ha de seguir funcionando en todo caso

El siguiente caso de uso nos ilustra a grandes rasgos como crear grupos de trabajo. Podemos discernir en la Figura 24 como debemos obtener los usuarios y consultar con usuario administrativo varias veces para conseguir el objetivo.

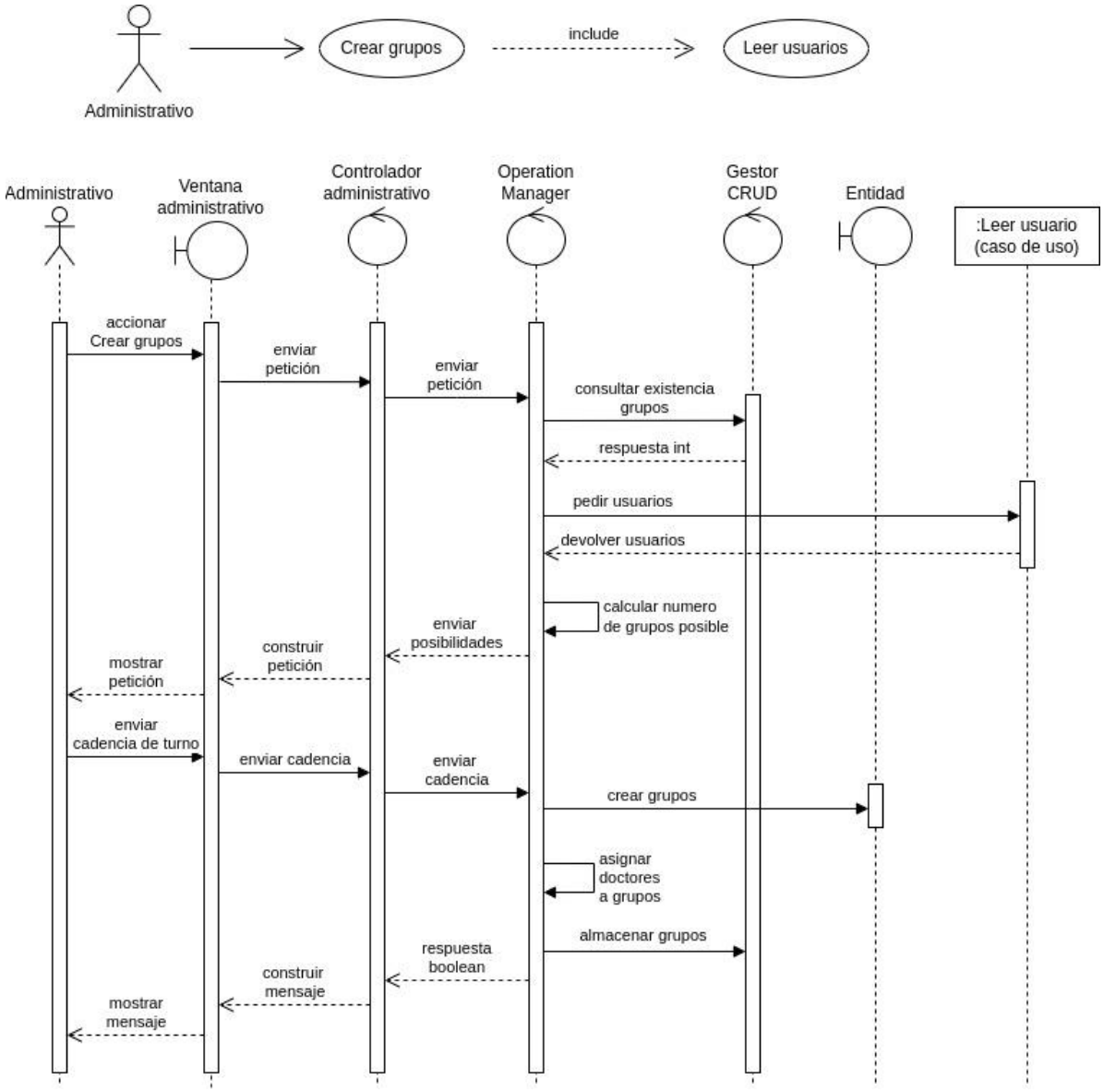

Figura 24 Diagrama de secuencia, Crear grupos

Nombre	*Crear grupos*
Descripción	Crea los grupos de trabajo necesarios para el funcionamiento del calendario. Cada grupo representa un "grupo de guardia" por lo que, si tenemos 6 grupos, tendremos guardias cada 6 días.
Participantes	Usuario administrativo
Casos de uso externos	Leer Usuario
Precondiciones	1. La BBDD ha de funcionar correctamente 2. El usuario ha de estar logeado 3. El usuario ha de pertenecer a un centro sanitario
Flujo Normal	
1	El usuario administrativo entra en la zona del sistema destinada a los grupos
2	El usuario activa la función de crear grupos en la ventana
3	El operation mánager recibe la petición de crear los grupos y calcula el número de grupos posible en función de los datos del centro sanitario y de los médicos disponibles
4	El operation mánager hace llegar la petición de número de grupos a crear al usuario
5	El usuario responde la petición de número de grupos a crear
6	El operation mánager crea los grupos
7	El operation mánager asigna los doctores a cada grupo
8	El operation mánager le envía al CRUD mánager los grupos para que los guarde en el sistema
9	Se envía un mensaje al usuario confirmándole el resultado
Flujos Alternos	
1.A	¿Existen grupos en el sistema?
1.A.1	Si existen, comunicar y finalizar
1.A.2	No existen, el flujo continúa
3.A	¿Existen médicos suficientes para las cadencias de 6 u 8?
3.A.1	Solo existen médicos para cadencias de 6, se le comunica al usuario y si acepta se prosigue
3.A.2	Existe la posibilidad de ambas cadencias, se pregunta y se prosigue el flujo en función de la respuesta.
3.A.2	No hay médicos para ninguna cadencia. Comunicar y finalizar
8.A	¿Los grupso se guardan correctamente?
8.A.1	Si, se prosigue el flujo
8.A.2	No, se comunica al usuario el fallo y se finaliza la operación
Postcondiciones	
1	El sistema ha de seguir funcionando en todo caso
2	El usuario es consciente del resultado de la operación

Con el siguiente caso de uso, crear calendario, representado en la Figura 25 se desarolla la funcionalidad más famosa de la aplicación y la que da realmente sentido a su existencia. Nota destacable de este caso de uso es que debe encajar un turno con el del mes anterior.

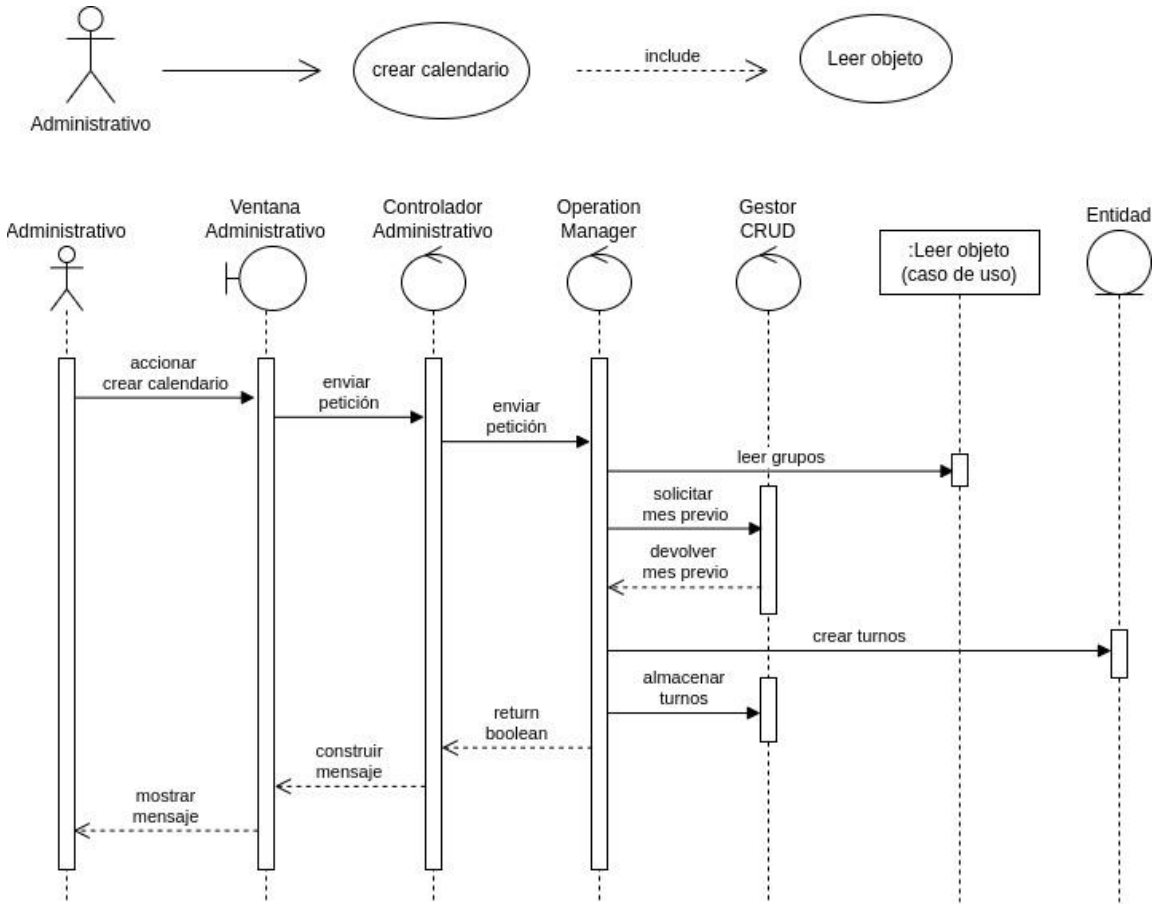

Figura 25 Diagrama de secuencia, Crear calendario

Nombre	*Crear calendario*
Descripción	Crea meses de turnos para nuestro calendario
Participantes	Usuario administrativo
Casos de uso externos	Leer objeto
Precondiciones	1. La BBDD ha de funcionar correctamente 2. El usuario administrativo ha de estar logeado 3. El usuario ha de pertenecer a un centro sanitario 4. No ha de existir turno existente en el mes a crear 5. Han de existir grupos de trabajo ya definidos en el sistema
Flujo Normal	
1	El usuario administrativo entra en la zona de crear calendario
2	El usuario administrativo selecciona mes y año a crear
3	El operation mánager recibe la petición de creación de turno
4	EL operation mánager recoge los grupos de trabajo de ese centro sanitario
5	El operation mánager comprueba que no exista turno para ese mes
6	El operation mánager va elaborando los turnos de trabajo día por día
7	El operation mánager guarda los turnos en la BBDD
8	Se notifica el resultado al usuario
Flujos Alternos	
4.A	¿Existen grupos de trabajo para el centro sanitario?
4.A.1	No, se notifica al usuario y finaliza la función
4.A.2	Si, se comprueba que el número sea 6 u 8
4.A.2.A	¿El número es correcto?
4.A.2.A.1	No, se notifica al usuario y finaliza la función
4.A.2.A.2	Si, se continúa el flujo normal
5.A	¿Existe turno para ese mes?
5.A.1	Si, se notifica al usuario y finaliza la función
5.A.2	No, se continúa el flujo normal
Postcondiciones	
1	El sistema ha de seguir funcionando en todo caso
2	El usuario es consciente del resultado de la operación
3	El sistema ha de garantizar la finalización y almacenamiento de todo el calendario o de ninguno.

El caso de uso mostrado sirve para resolver bajas y vajacaciones, es equivalente para ambos supuestos con la única diferencia de que las vacaciones pueden ser rechazadas. Como vemos en la Figura 26 este caso de uso deriva en otras dos posibles soluciones, el uso de compañeros y la sustitución con externos para la sustitución. Todo ello gestionado en la etapa “resolver”

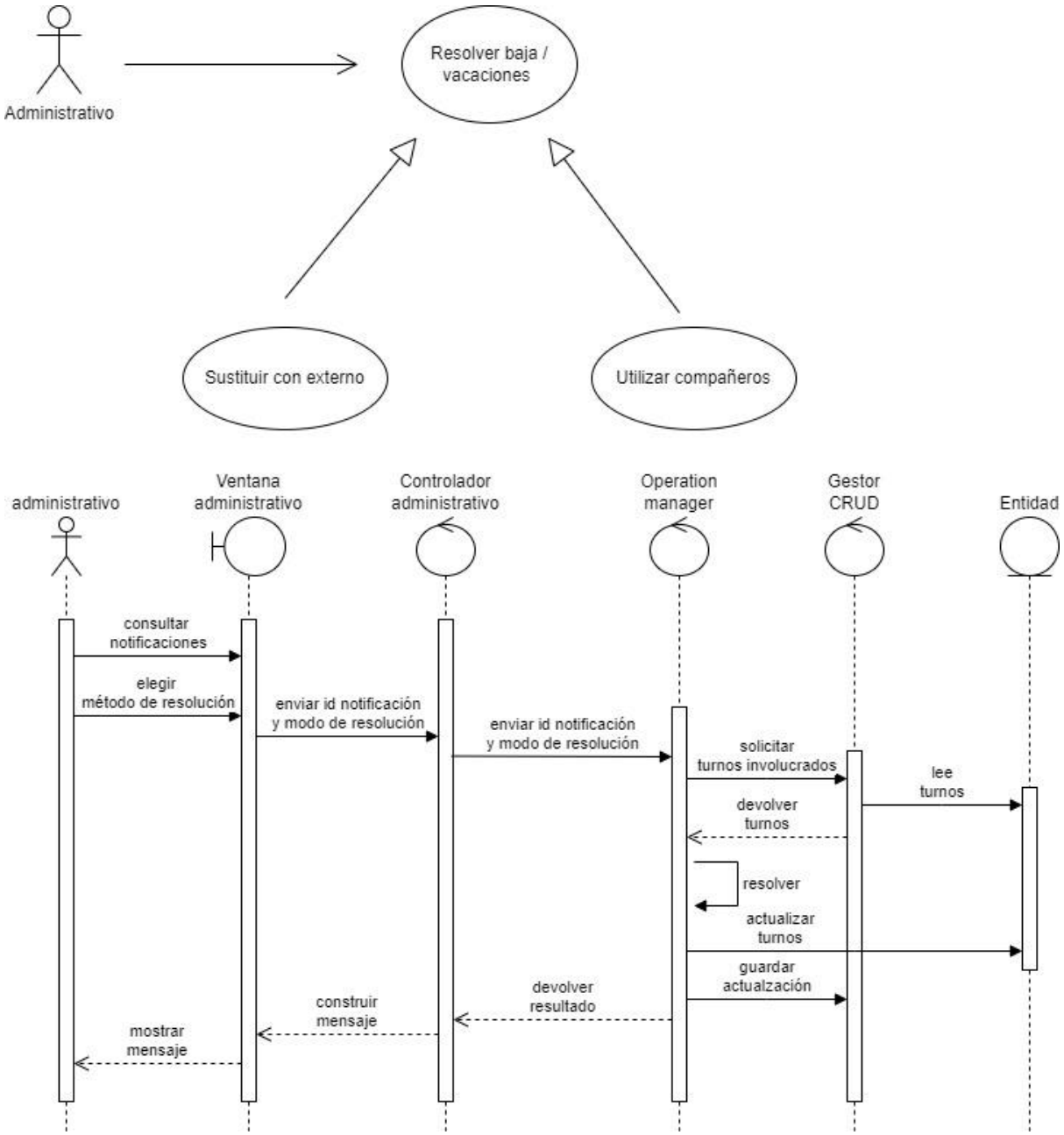

Figura 26 Diagrama de secuencia, resolver baja / vacaciones

Nombre	*Resolver baja / vacaciones*
Descripción	Gestionar una baja o unas vacaciones de un empleado
Participantes	Usuario administrativo
Precondiciones	1. La BBDD ha de funcionar correctamente 2. Existe una notificación de vacaciones / baja en el sistema 3. El usuario está logueado en el sistema 4. El usuario administrativo pertenece al mismo centro sanitario que el solicitante 5. Ha de existir un turno de trabajo establecido para los días solicitados
Flujo Normal	
1	El administrativo consulta las notificaciones registradas en el sistema
2	El usuario selecciona la notificación el método de resolución
3	El operation mánager recibe el id de notificación y el método de resolución
4	El operation mánager recoge los turnos involucrados en los posibles cambios de la BBDD
5	El operation mánager realiza los cambios necesarios en los turnos en función del método de resolución
6	Se actualizan los turnos en la BBDD
7	Se modifica la notificación inicial
8	Se muestra al usuario administrativo el resultado de la operación
Flujos Alternos	
2.A	¿La notificación es de vacaciones?
2.A.1	Si, se puede seleccionar rechazar
2.A.2	No, solo existen las opciones estándar
3.A	¿Notificación rechazada? (solo caso vacaciones)
3.A.1	Si, pasa directamente al paso 7 y se establece la notficación en “rechazado”, no se hacen más cambios
3.A.2	No, se prosigue con el flujo
5.A	¿Se contrata sustituto?
5.A.1	Si, Se asignan los turnos al empleado sustituto
5.B	¿Serán los compañeros los que sustituyen?
5.B.1	Si, Los compañeros sustituyen en base a unos criteros al empleado ausente
Postcondiciones	
1	El sistema ha de seguir funcionando en cualquier caso
2	La baja / vacaciones ha de gestionarse de forma íntegra, en caso contrario se vuelve al estado original

El caso de uso siguiente e ilustrado en la Figura 27 nos explica como crear notificaciones dentro del sistema. Las notificaciones son disparadas por las peticiones de cambios de turno de un médico sea cual sea su naturaleza

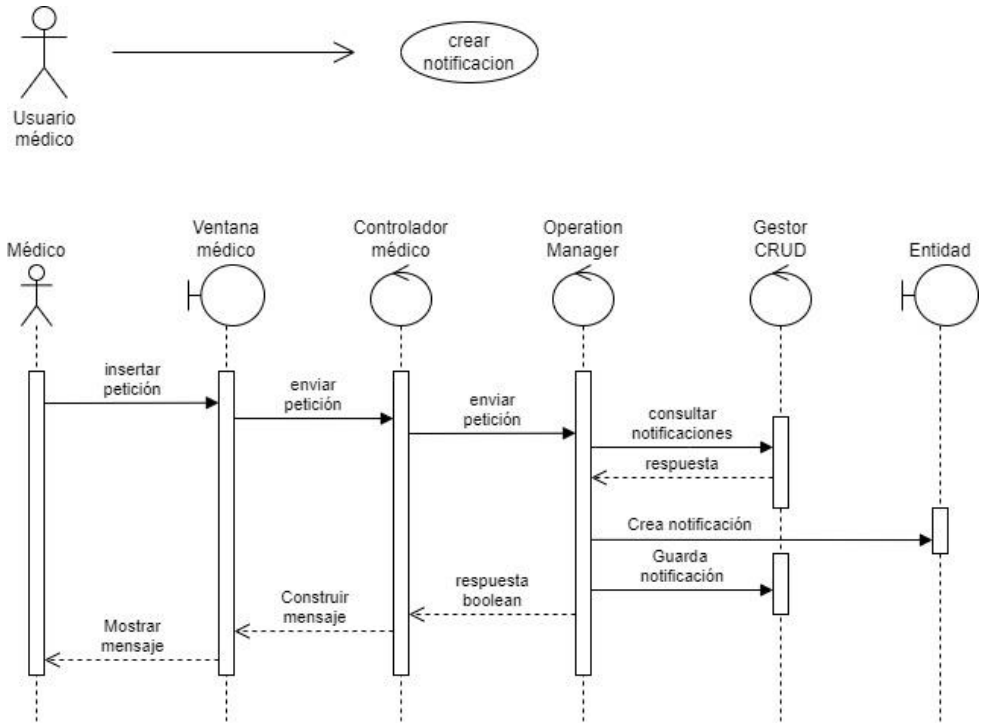

Figura 27 Diagrama de secuencia, Crear notificación

Nombre	*Crear notificación*
Descripción	Crea una notificación en el sistema para una posterior resolución por parte de los usuarios administrativos
Participantes	Usuario médico
Precondiciones	1. La BBDD ha de funcionar correctamente 2. El usuario está logueado en el sistema 3. El médico tiene contrato con un centro sanitario 4. Debe existir un turno de trabajo para las fechas incluidas en la notificación
Flujo Normal	
1	El médico inserta la petición en el sistema
2	El sistema consulta si existen otras notificaciones para esas fechas
3	El sistema crea y guarda la nueva notificación
4	El sistema contesta al médico con el resultado de la creación de la notificación
Flujos Alternos	
2.A	¿Existen otras notificaciones?
2.A.1	Si, se notifica la existencia de otra notificación y se finaliza la función
2.A.2	No, se prosigue el flujo normal
Postcondiciones	
1	El sistema ha de seguir funcionando en cualquier caso

El siguiente caso de uso, buscar compañeros, es un caso de uso auxiliar ya que será invocado por otros casos de uso y por otras funciones de la aplicación. Su explicación se detalla en la Figura 28

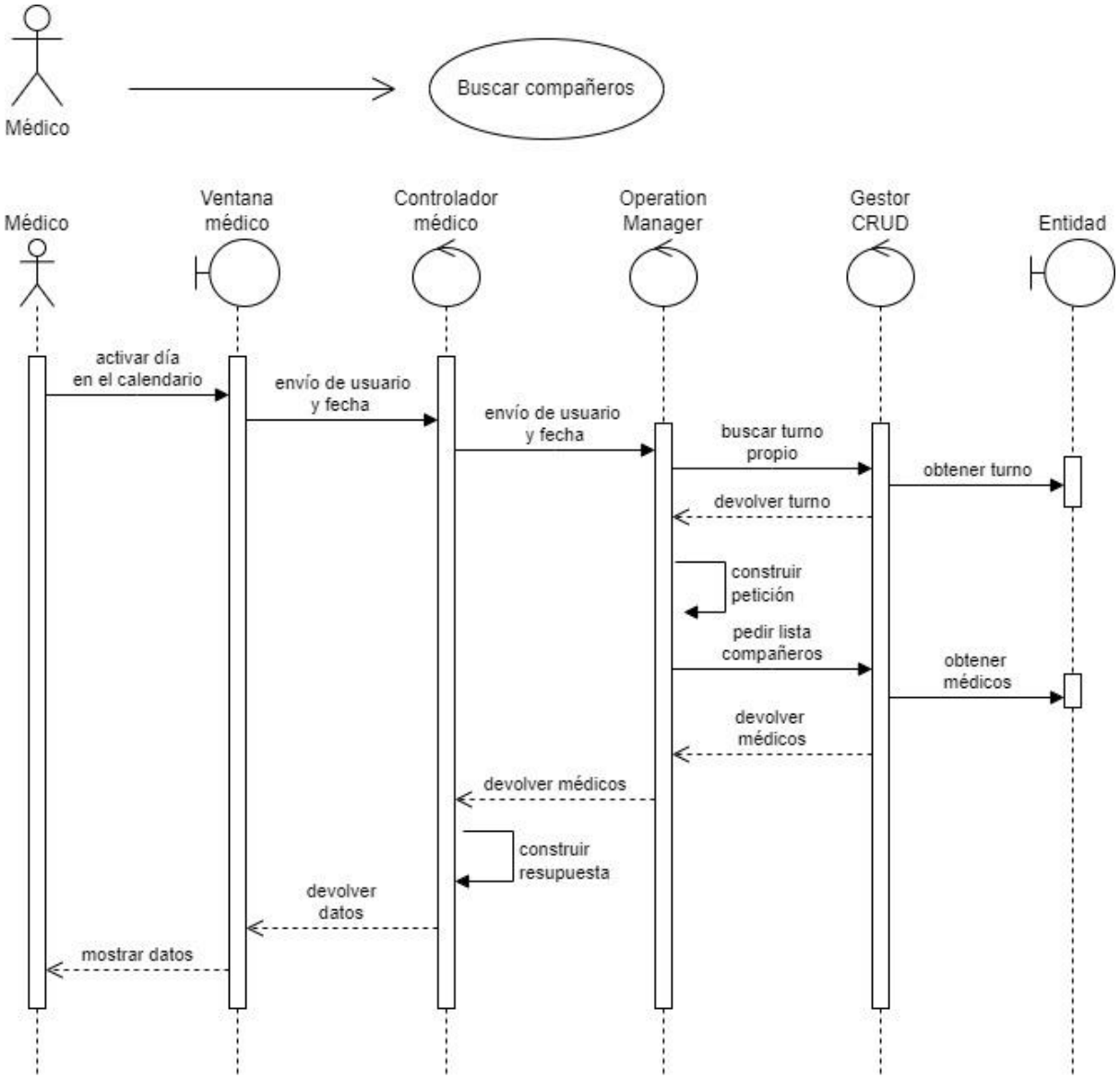

Figura 28 Diagrama de secuencia, Buscar compañeros

Nombre	*Buscar compañeros*
Descripción	Devuelve una lista de compañeros compatibles para un cambio
Participantes	Médico
Precondiciones	1. La BBDD ha de funcionar correctamente 2. El médico está logueado en el sistema 3. El médico pertenece a un centro sanitario 4. Ha de existir un turno para las fechas a cambiar
Flujo Normal	
1	El médico entrra en su página de calendario y activa el día que quiere cambiar
2	Se envía el día y el usuario al operation mánager
3	El operation mánager busca el turno que tiene el médico el día que quiere cambiar
4	El operation mánager construye una petición para la BBDD
5	Se realiza la consulta a la BBDD
6	Se devuelven los médicos al controlador
7	El controlador construye el mensaje necesario para la vista
8	La vista muestra los datos al usuario
Flujos Alternos	
4.A	El médico tiene turno de mañana o tarde
4.A.1	Se construye la petición con el turno inverso al que tiene le médico, es decir, si el médico está de mañana, se piden los médicos que están de tarde ese día y viceversa
4.B	El médico tiene turno de guardia
4.B.1	Se busca a los miembros del grupo antípoda que tienen su turno original (no ha sufrido cambios) ese día
Postcondiciones	
1	El sistema ha de seguir funcionando en todo caso

Si antes teníamos un caso de uso que creaba calendarios es obvio pensar que posteriormente tendremos que ser capaces de consultarlos. Si bien su proceso explicado en la Figura 29 puede parecer trivial ha de cumplir obviamente una serie de requisitos detallados a continuación.

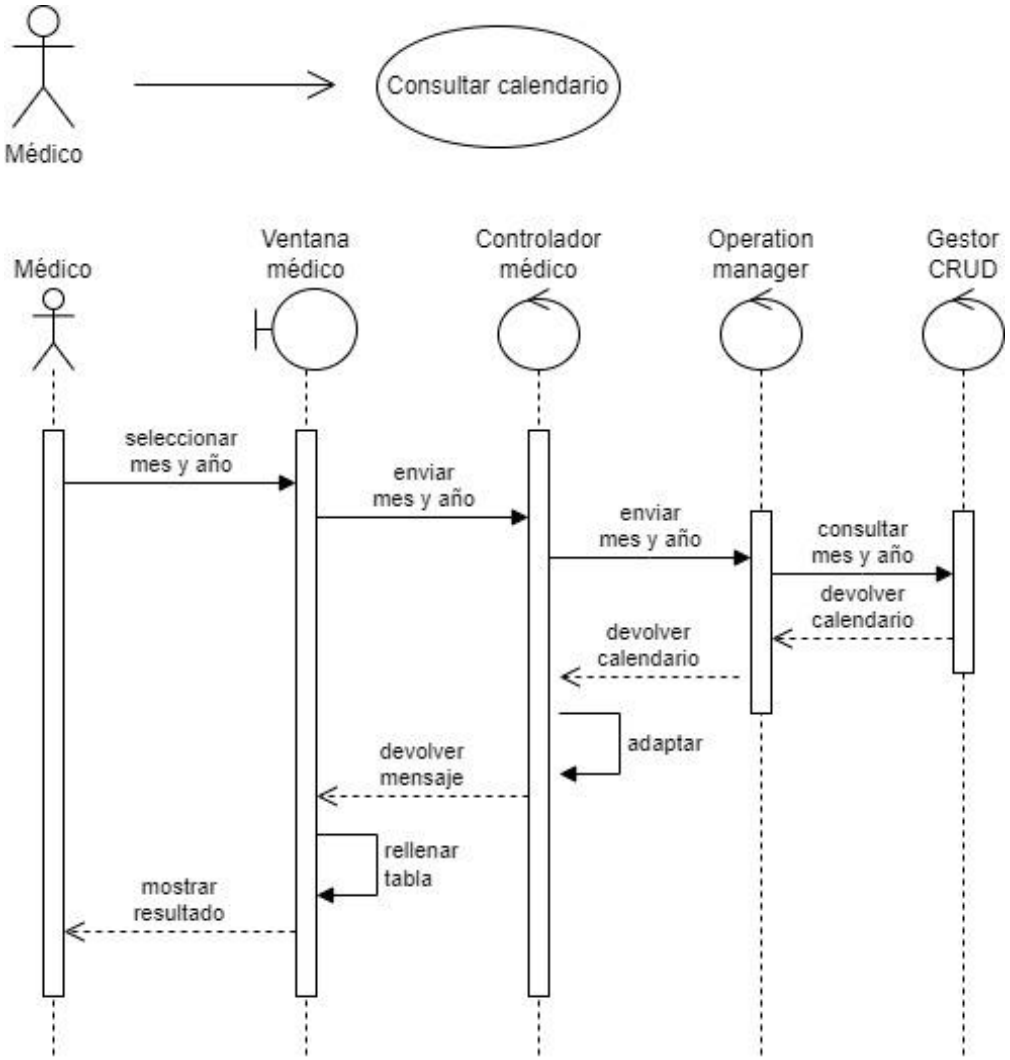

Figura 29 Diagrama de secuencia, Consultar calendario

Nombre	*Consultar calendario*
Descripción	El médico consulta su turno de trabajo para un mes concreto
Participantes	Médico
Precondiciones	1. La BBDD ha de funcionar correctamente 2. Ha de existir un turno de trabajo para esa fecha 3. El médico está logeado en el sistema
Flujo Normal	
1	El médico introduce el mes y el año que quiere consultar
2	La petición de mes y año llega al operation manager
3	El operation manager hace la petición correspondiente al gestor crud
4	El gestor CRUD busca en la BBDD
5	El gestor CRUD devuelve el calendario
6	El operation mánager devuelve el calendario
7	El controlador adapta la respuesta a la vista y se le envía
8	La vista rellena la tabla
9	Se muestra el resultado al usuario
Flujos Alternos	
	No existen, se devuelve el calendario o se muestra un mensaje de error si no existe calendario en el sistema en el último paso
Postcondiciones	
1	El sistema ha de seguir funcionando en todo caso

El caso de uso mostrado a continuación es el más complejo de todo el programa. Intercambiar turnos requiere llamar u otros muchos casos de uso como muestra la Figura 30

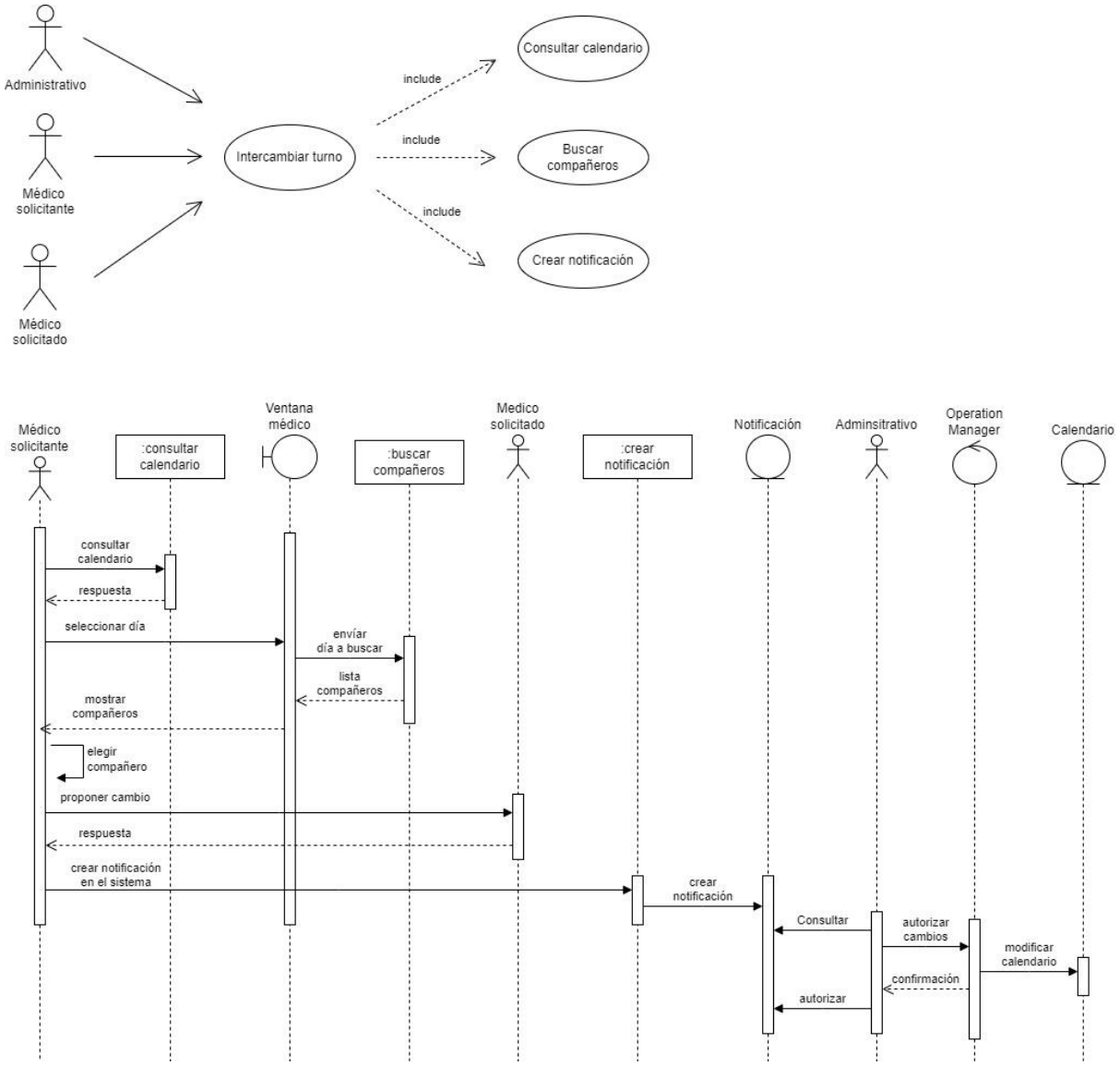

Figura 30 Diagrama de secuencia, Intercambiar turno

Nombre	*Intercambiar turno*
Descripción	Dos médicos intercambian entre si un turno compatible. Es decir, una mañana por una tarde o una guardia por otra
Participantes	Médico solicitante, médico solicitado, administrativo
Precondiciones	1. La BBDD ha de funcionar correctamente 2. Los actores pertenecen todos al mismo centro sanitario 3. Existe un turno establecido para el turno a cambiar 4. Los actores han de logearse en el sistema 5. El turno para cambiarv debe ser un turno original del sistema

Flujo Normal	
1	El médico solicitante consulta el calendario para conocer su turno el día que quiere cambiar
2	El médico solicitante recibe la lista de compañeros compatible para el cambio.
3	El médico solicitante contacta con los posibles compañeros
4	El médico solicitado contesta a la solicitud de cambio
5	El médico solicitante crea la notificación
6	El administrativo atiende la notificación
7	El administrativo le dice al operation mánager que autoriza el cambio
8	El operation mánager hace las modificaciones necesarias en el calendario
9	El usuario administrativo cambia el estado de la notificación a aceptado
Flujos Alternos	
3.A	¿Hay médicos compatibles para el cambio?
3.A.1	Si, el solicitante tiene compañeros compatibles para el cambio. Prosigue el flujo
3.A.2	No hay médicos compatibles, finaliza la función.
4.A	Contestaciones de los compañeros
4.A.1	Todas son negativas, finaliza el proceso al no tener compañero con el que intercambiar
4.A.2	Al menos un compañero contesta afirmativamente, continua el flujo
7.A	Autorizar el cambio
7.A.1	Si, prosigue el flujo
7.A.2	No, se establece la notificación en rechazado y finaliza la función
8	Tipo de turno a intercambiar
8.A	El cambio es mañana por tarde o tarde por mañana
8.A.1	¿La fecha es la misma?
8.A.1.A	Si, se intercambian los turnos y prosigue el flujo
8.A.1.B	No, se notifica del error y finaliza la función
8.B	El cambio es de guardia por guardia
8.B.1	¿Es grupo antípoda?
8.B.1.A	No, se notifica el error y finaliza la función
8.B.1.B	Si, se intercambia el día de guardia y los dos días siguientes
Postcondiciones	
1	El sistema ha de seguir funcionando en todo caso
2	Los turnos no se modifican si no es de forma completa

Capítulo 7 - Desarrollo del software

En este capítulo llegamos al momento que todo desarrollador quiere, que es pasar a codificar. Apoyándonos en la metodología anteriormente descrita y en los casos de uso que acabamos de recibir, ahora es el momento de elegir que patrones de diseño podemos utilizar para implementar los casos de uso y el resto de los requisitos funcionales y no funcionales de la aplicación.

Hablaremos del estándar Modelo / Vista / Controlador (MVC) como la principal guía de diseño y de los patrones que utilizaremos para partes más concretas.

Describiremos también como no puede ser otro modo las clases participantes y de como se estructurarán en paquetes para facilitar su gestión.

Finalmente, abordaremos como usando conceptos aprendidos durante el grado podemos hacer un código de mejor calidad.

7.1 Patrones de diseño. MVC original, uso en la actualidad y adaptaciones al proyecto

El uso de patrones de diseño para el desarrollo de software es considerado algo más una buena práctica, se considera algo indispensable.

Un patrón de diseño no es más que una guía o plantilla reutilizable para resolver una situación concreta de una forma generalizada. De esta forma tenemos siempre la misma estructura para resolver un mismo conjunto de problemas ahorrándonos el tiempo de tener que plantearnos como afrontarlo cada vez que nos lo encontramos.

Tenemos patrones de diseño asociados al código fuente y otros que tienen que ver más con la arquitectura (como el MVC). Existe mucha documentación sobre esto en las redes por lo que no nos vamos a detener en aumentar el detalle de esta explicación.

MVC[28] es el más famoso de los patrones de diseño de software. Su principal característica es que está pensado para separar un programa en tres partes:

- Modelo de datos, que contendrá los datos subyacentes del sistema
- Vistas que mostrarán a los usuarios información almacenada en el modelo.
- Controlador que orquestará las operaciones necesarias entre ambos componentes

Esto es un esbozo de la idea general de este patrón de diseño. Sin embargo, no es más que la punta del iceberg de lo que realmente pasa en una aplicación y no debe tomarse como un dogma si no como una estrategia a seguir. Entendamos que esta estrategia implica que la presentación e interacción con el usuario van cada una por un lado y que finalmente las interacciones entre todos los componentes también deben tener su independencia.

[28] MVC – Modelo / Vista / Controlador

Una forma básica de este patrón puede representarse con el ejemplo del avión y el piloto representados en la Figura 31 la cual explica a grandes rasgos como interaccionan los tres componentes principales entre si.

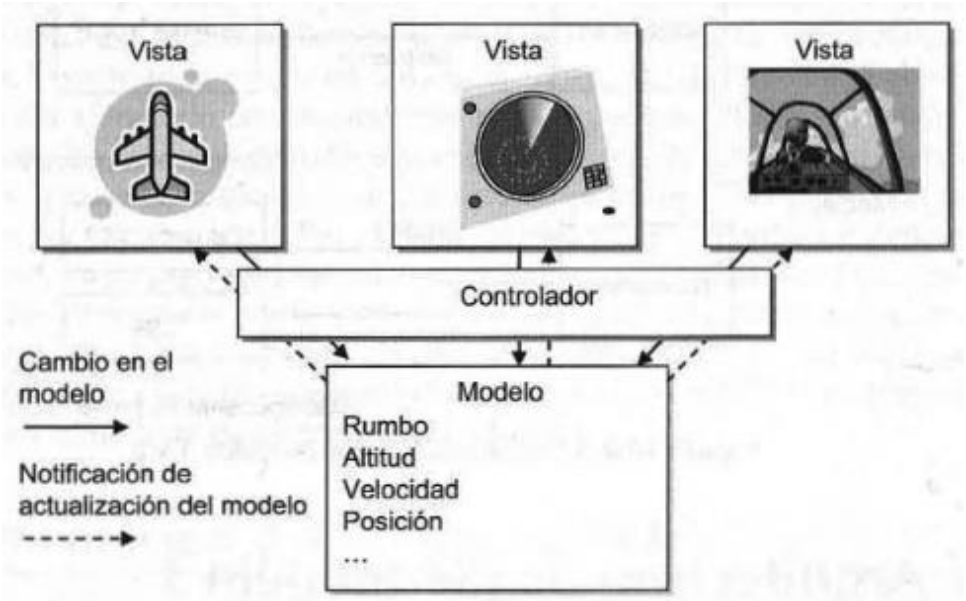

Figura 31 Ejemplo de representación del patrón MVC

Fuente: (Parsons, 2009) *Figura 10.2 en libro*

Este proyecto se ha inspirado en este patrón de diseño ya que una aplicación de escritorio se adapta muy bien a él. No obstante, como ya he dicho antes no debe tomarse este patrón como un dogma si no como una idea general.

En la actualidad los proyectos han evolucionado de tal forma que el controlador ha sufrido la mayor evolución. Actualmente no se tiene un solo controlador central que orqueste todas las operaciones siendo más habitual la existencia de muchos controladores para tareas más específicas, aunque tengan uno central superior que los coordine o no.

Podemos decir que ahora el diseño general está dividido en dos partes más generales que son el front-end y el back-end, entendiendo que el primero se encarga de la presentación e interacción con el usuario y el segundo de todo lo que sucede aparte de la presentación. En este contexto el controlador queda un poco en medio de ambas quedando a criterio del desarrollador cuan visible pueda o no ser y cuanta independencia pueda o no tener además de cuanto está más cerca de una parte o de otra.

El back-end se subdivide de forma general en las siguientes capas:

- Lógica de negocio
    - Contiene las reglas de negocio de la aplicación
    - Conjunto de clases donde se procesan, actualizan y modifican los datos
    - Realiza todas las operaciones aritméticas que sean necesarias
    - Recibe y envía datos entre la capa de presentación (vistas) y la capa de datos
- Capa de acceso a datos
    - Contiene la dirección de los distintos orígenes de datos a los que tendrá acceso el sistema
    - Establece la configuración con los distintos orígenes de datos
    - Realiza las operaciones de lectura y escritura que le ordena la capa de lógica de negocio
    - Proporciona una abstracción de las operaciones CRUD[29] para la capa de lógica de negocio

En la siguiente sección explicaremos como las clases Java de nuestra aplicación se circunscriben a las capas.

[29] CRUD – Create / Read / Update / Delete

7.2 Clases participantes

Esta sección pretende hacer un análisis general de la estructura de clases de la aplicación asociándolo a las capas citadas en el apartado anterior. No entraremos en detalle del código ya que considero que la implementación propiamente dicha es algo personal de cada programador. No obstante, puedo asegurar que se ha realizado siguiendo la mayoría de las buenas prácticas actuales.

Debido al gran número de clases utilizadas he decidido también dividir los diagramas para que sean visualmente más agradables.

Ya que la aplicación consta de dos perfiles diferenciados: administrativo y médico. Los diagramas estarán también enfocados de esa forma mostrando así las clases sobre las que participará cada perfil. Del mismo modo para el back-end separaremos los diagramas de lógica de negocio y acceso a datos.

7.2.1 Operations Manager / el contrato del corazón de la aplicación

En un primer lugar y antes de entrar en materia, se expone en la Figura 32 la interfaz IOperationManager, que es la que contiene todas las funcionalidades propiamente dichas de la aplicación. Si tiramos del hilo del capítulo de esta memoria en el cual exponíamos los requisitos funcionales veremos que están todos aquí representados. Además, hay funciones auxiliares al propio funcionamiento del programa las cuales también pertenecen a la lógica de negocio aquí aglutinada.

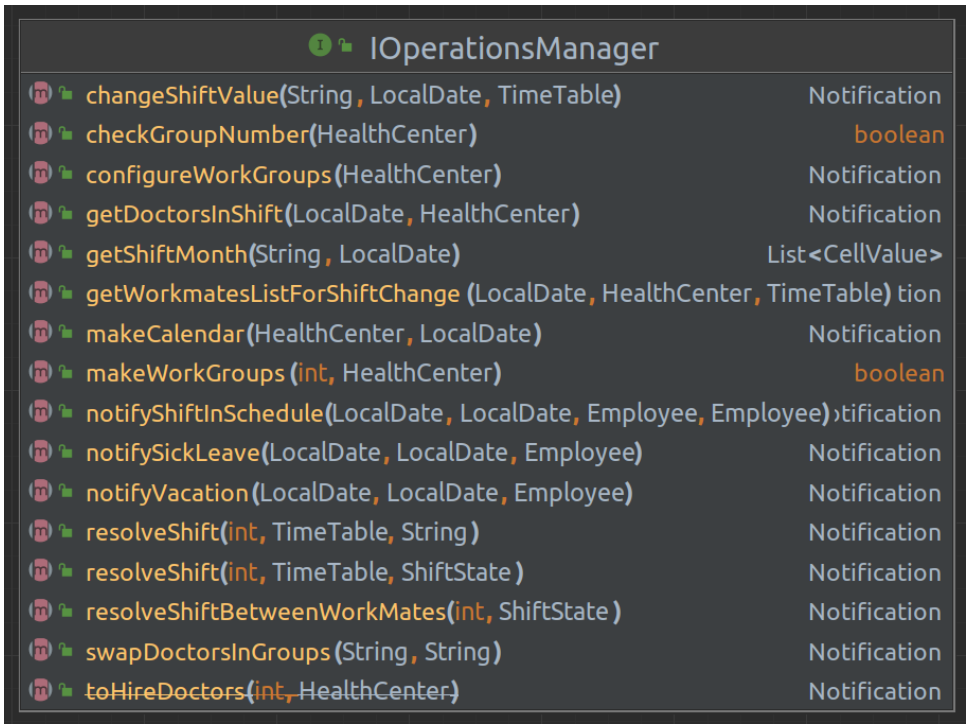

Figura 32 Interfaz IOperationsManager

Fuente: Captura de pantalla directa del IDE IntelliJ

7.2.2 Estructura de paquetes

Comienzo con la descripción del sistema de paquetes que es la siguiente

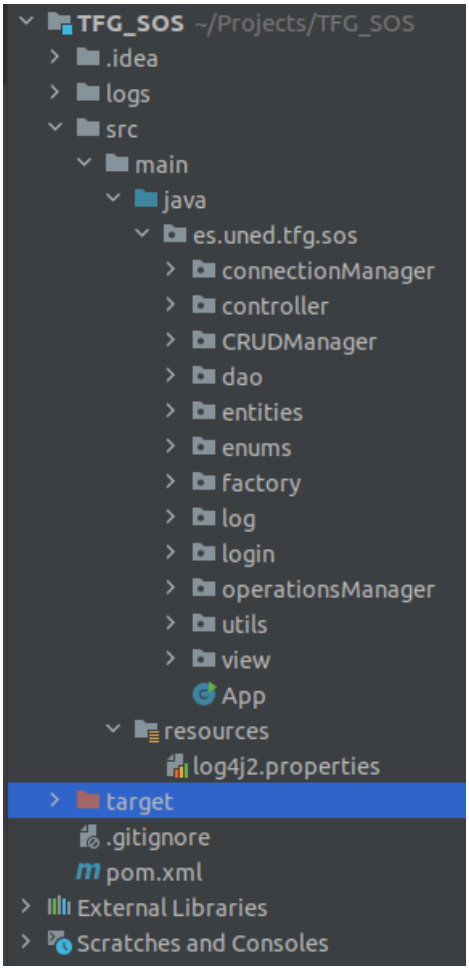

Figura 33 Estructura de paquetes

Fuente: Captura de pantalla directa del IDE IntelliJ

Podemos ver en a la izquierda en la Figura 33 la estructura completa de paquetes de la aplicación.

Si bien es posible hacer una mayor subdivisión incluyendo directamente el frontend y backend como dos paquetes que "cuelguen" del principal, se ha preferido dejar así para mantener la versatilidad y escalabilidad en caso de que fuese necesaria.

Como vemos la clase APP aparece suelta y parece que está fuera de contexto, pero no es así. Esta clase hace de lanzador de la app estableciendo las configuraciones necesarias por defecto y un correcto orden de arranque de los componentes de la aplicación.

Fuera de la estructura destacan otros dos elementos típicos que son el archivo log4j2.properties que establece la configuración del log explicado Anexo 1 Logger y el archivo pom.xml que contiene la configuración de todas las dependencias externas al proyecto.

7.2.3 Front-end

El front-end de este proyecto, como ya se ha dicho anteriormente, es una interfaz de escritorio multiplataforma diseñada en con SWING. La estructura del front-end se basa en los paquetes Views y Controller.

El controlador está asociado al perfil de usuario y a todas las ventanas que tienen relación con ese perfil. Los nombres de clase como vemos son plenamente descriptivos y no necesitan una mayor explicación.

Aclaramos que las flechas de color verde representan las relaciones de implementación y las azules la extensión. Las flechas de color blanco explican el uso y cardinalidad. Todo ello cumple el estándar habitual de UML.

En la Figura 34 que podemos ver detalla la relación de clases utilizadas en el front-end por el perfil de usuario médico. La aplicación comienza llamando al LogínController que crea la LoginView. En función del login se iniciará un perfil u otro, en este caso el perfil médico. Vemos también todas las vistas diferentes a la que este perfil tendrá acceso, así como los controladores. La clase BaseController es abstracta y sirve para aglutinar código común a ambos controladores, por eso la vista principal trabaja con ella al ser un campo protegido en su interior.

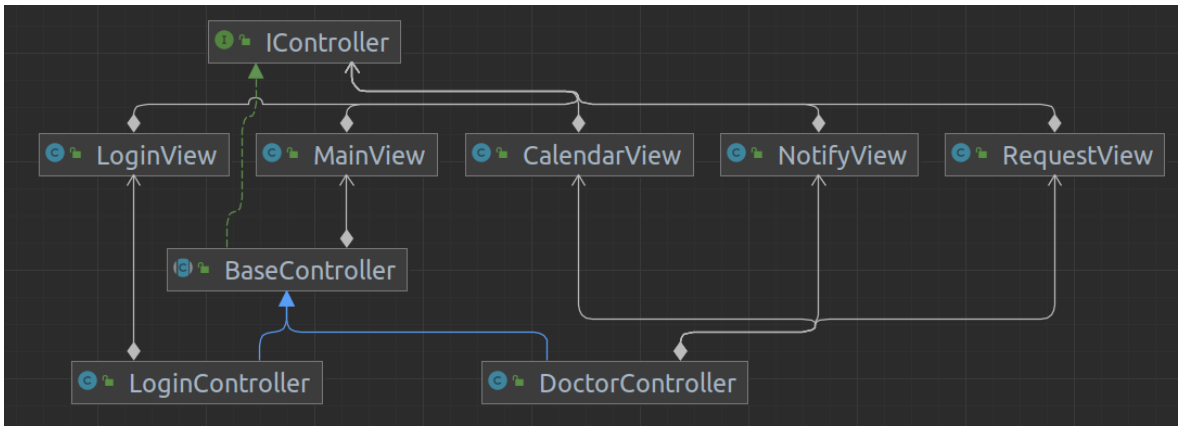

Figura 34 Diagrama de clases, perfil médico

Fuente: Captura de pantalla directa del IDE IntelliJ

Ahora, la Figura 35 nos detalla el front-end vinculado al perfil administrativo. Como vemos es casi idéntica a la figura anterior con la salvedad de que las vistas son obviamente distintas al pertenecer a otro perfil.

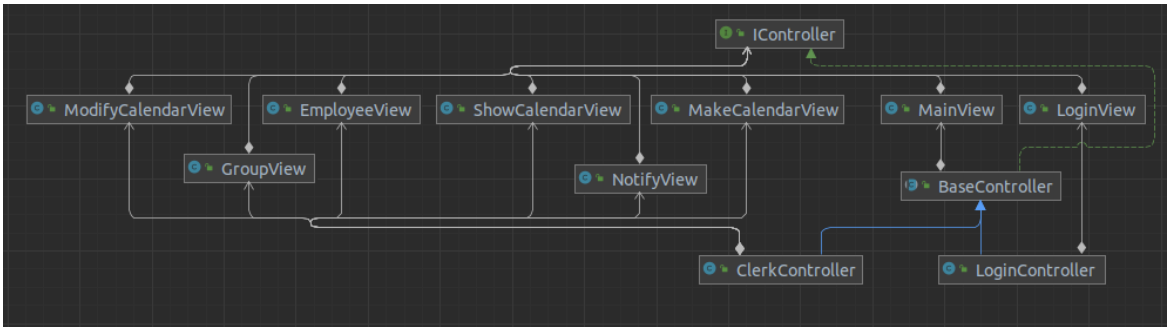

Figura 35 Diagrama de clases, perfil administrativo

Fuente: Captura de pantalla directa del IDE IntelliJ

Ilustraremos ahora el front-end con dos de las imágenes más representativas. La figura Figura 36 es uno de los mejores ejemplos de la aplicación. En ella vemos una pantalla dividida en dos secciones. La parte izqueirda es totalmente dinámica y consta de un calendario que está conectado internamente con la BBDD. Si cambiamos el mes o el año obviamente la disposición de los días de calendario cambiará, lo importante además de esto es que, cada vez que pulsemos encima de un día (en el ejemplo se ha pulsado el día 16 de septiembre de 2023) la tabla de la derecha cambiará de forma automática mostrando la situación de todos los trabajadores de la plantilla ese día. Esta pantalla forma parte del perfil administrativo y un médico concreto no tiene acceso a ella ya que no tiene por qué conocer el turno de todo el servicio.

Figura 36 Visualización de calendario

Esto nos hace plantearnos la siguiente pregunta, si un médico no conoce los turnos de trabajo de sus compañeros ¿cómo es posible saber a quién le tiene que pedir el cambio? La respuesta a esto es que el sistema ha de guiar a ese médico a realizar cambios que no rompan la estructura de todo el calendario y sea posible con un mínimo de ajustes.

La figura Figura 37 nos ilustra esa situación, pertenece al perfíl médico. En ella, la parte izquierda sirve para seleccionar el mes y año del calendario el cual se reprentará en la zona central de la imagen y cuando pinchamos encima del día que queramos el sistema nos sugerírá en la tabla de la derecha una lista de compañeros compatibles con el turno del día que queremos cambiar.

Figura 37 Visualización de calendario mensual

7.2.4 Back-end

Como se ha dicho anteriormente el back-end está dividido por definición en dos partes: Lógica de negocio y la capa de acceso a datos. Por lo que si bien y como es evidente hay partes compartidas haremos una imagen para cada una.

- Lógica de negocio

  En la Figura 38 vemos representadas las clases participantes en la lógica de negocio. El 90% de la lógica y la funcionalidad de la aplicación se gestiona desde la clase OperationsManager.

  Si bien el CRUDManager tiene que ver con la capa de acceso a datos, se añade a esta figura para que quede claro que es el nexo entre ambas partes del back-end.

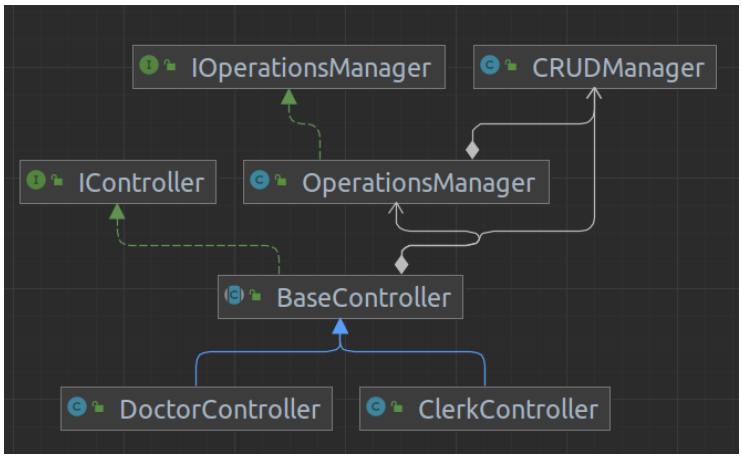

Figura 38 Diagrama de clases, lógica de negocio

Fuente: Captura de pantalla directa del IDE IntelliJ

- Acceso a datos

  En la Figura 39 y reconociendo al CRUDManager como el nexo con la lógica de la aplicación, exponemos la capa de acceso a datos.

Figura 39 Diagrama de clases, capa acceso a datos

Fuente: Captura de pantalla directa del IDE IntelliJ

Pasamos a explicar la imagen superior que es la que nos muestra de forma global toda la capa de acceso a datos. La imagen está organizada de izquierda a derecha tal y como ocurre con su funcionamiento.

En un primer momento el crudManager recibe una petición de datos desde la capa de lógica o desde la capa de presentación. Esta clase es la encargada de pedir el objeto directamente a la BBDD y enviar

lo que le piden, bien sea el objeto entero, un array con Strings de campos del objeto, o incluso mezclas de valores de varios objetos. Esto hace que el crudManager tenga dos posibles operaciones en las que a veces es un simple intermediario que recibe petición, reenvía la petición a quien debe y devuelve lo mismo que le han pedido o, por el contrario, puede recibir una petición y hacer múltiples llamadas a otros participantes para componer su respuesta.

El CRUDManager se conecta con una fábrica que ofrecerá las instancias de cada uno de los DAO asociado a una entidad. Tanto la fábrica como el DAO se explicará con detenimiento en la próxima sección (patrones de diseño).

Cada uno de estos DAO recibe una petición y utiliza el ConnectionManager para establecer conexión con el origen de los datos. La información es recogida y devuelta al CRUDManager el cual hará con ella lo que tenga que hacer.

Los DAO van asociados cada uno a una clase que en función del punto de vista obtiene un nombre diferente:

- Entidad, desde el punto de vista de una base de datos
- POJO, desde el punto de vista del código Java más básico
- BEANS, desde el punto de vista de Java EE

Independientemente de cómo les llamemos, estos objetos sirven para representar datos de una forma sencilla. Un ejemplo de estas entidades lo tenemos en la Figura 40 que representa al objeto empleado:

Figura 40 Representación UML de la clase empleado

Fuente: Captura de pantalla directa del IDE IntelliJ

7.3 Patrones de diseño

Como ya se ha explicado con anterioridad utilizar patrones de diseño es algo casi indispensable si aspiramos a realizar un proyecto serio y profesional. Utilizando el patrón MVC como algo conceptual esta aplicación cuenta con varios patrones de diseño los cuales han ayudado a estructurarlo.

Los patrones de diseño pueden agruparse en tres grandes grupos:

- Patrones de construcción
    - Encargados de las operaciones CRUD con objetos y estructuras
- Patrones estructurales
    - Sirven como plantillas y guías para poder crear arquitecturas con las clases de una forma estandarizada y coherente.
- Patrones de comportamiento
    - Definen posibles actuaciones que puede tener un objeto (o conjunto de ellos) en una determinada situación.

7.3.1 Patrón singleton

Este patrón de creación tiene una funcionalidad clara, evitar que haya más de una instancia de un objeto obligando a utilizar siempre la misma instancia en memoria de este.

Al crear un objeto como un singleton evitamos que ese objeto pueda ser instanciado con un new obligándole a pasar siempre por un método de obtención de instancia llamado habitualmente *instance()* ó *getInstance()*. Así se garantiza que, aunque se recoja esta instancia muchas veces y se asigne a diferentes variables incluso dentro de diferentes clases de operaciones vivas simultáneamente, sea la misma instancia. Podemos ver un código estándar de un patrón singleton en la Figura 41.

```
public class Comercial
{

  private static Comercial instance = null;

  private Comercial(){}

  public static Comercial getInstance()
  {
    if (instance == null)
      instance = new Comercial();
    return instance;
  }
```

Figura 41 Implementación patrón singleton

Fuente: Adaptación de (Drebauwer, Patrones de diseño en Java. Los 23 modelos de diseño: descripciones y soluciones ilustradas en UML 2 y Java, 2023)

El uso de este patrón se hace obligatorio con objetos que tienen como misión representar conexiones externas tales como conectores a BBDD, sockets, parsers … Es más que probable que si un usuario abre muchas conexiones contra un mismo servicio sin haber terminado las anteriores, el servicio colapse.

Se ha utilizado este patrón en aquellos contextos en los que se considera necesario por dos motivos principales. Bien sea porque el objeto siempre va a tener la misma función o bien porque está vinculado a conexiones externas.

Las clases declaradas como singleton en el sistema han sido

- CRUDManager
- DAO, cada una de sus implementaciones
- OperationManager
- DAOFactory
- LoginManager

7.3.2 Patrón fábrica

El llamado patrón factory es un patrón de construcción que tiene como función principal el poder ofrecer instancias de objetos a través de un intermediario. Gracias a este intermediario combinado con temas como la herencia o el uso de interfaces, en el caso de que hablemos de una entidad o de una clase de operaciones respectivamente, podemos obtener un tipo de objeto preestablecido o prototipado si fuese necesario.

Existen muchas maneras de implementar este patrón en función de las necesidades del programador. Las más famosas son abstract_factory, factory_method o simplemente factory.

Es muy común que las fábricas sean declaradas como singleton como se ha hecho en este proyecto. La Figura 42 que representa a la interfaz IDAOFactory que hemos empleado relacionada con el patrón DAO explicado en la sección 7.3.3 para obtener cada una de las implementaciones asociadas a entidades de este.

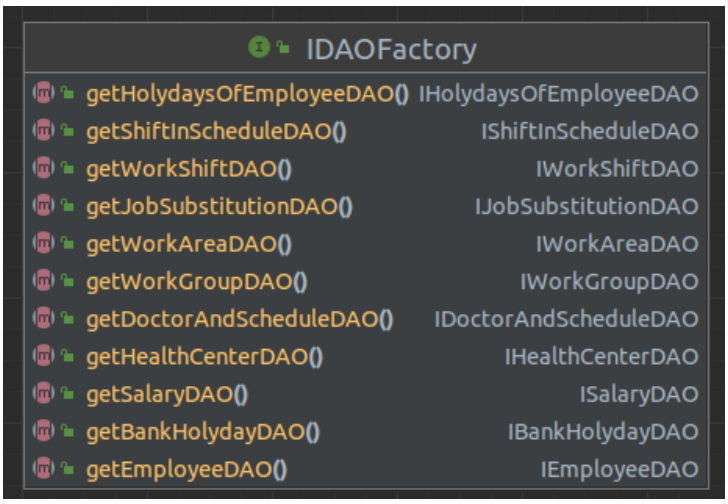

Figura 42 Interfaz IDAOFactory

Fuente: Captura de pantalla directa del IDE IntelliJ

La explicación del uso de este patrón en este proyecto viene por la necesidad de abstraer el origen de datos. Los DAO que se obtienen de la fábrica están pensados para utilizar una BBDD mysql en conjunto con Hibernate como ORM. Sin embargo, es posible que ambos cambien en un futuro y esto ha de ser transparente para el resto de la aplicación. En caso de cambio no habrá más que cambiar el objeto devuelto por el método get y con esto el cambio afectará a todo el proyecto a la vez.

7.3.3 Patrón DAO

Este patrón sirve para proporcionar transparencia en la conexión con el origen de datos al resto de la aplicación. La Figura 43 nos ilustra como el patrón relaciona una entidad person con un origen de datos JDBCPersonDAO ejecutando una serie de operaciones definidas en una interfaz. Si llevamos esta generalización a nuestro caso particular obtenemos el siguiente diagrama, el cual hemos reducido a un componente de cada tipo ya que, si los mostrásemos todos sería visualmente poco comprensible.

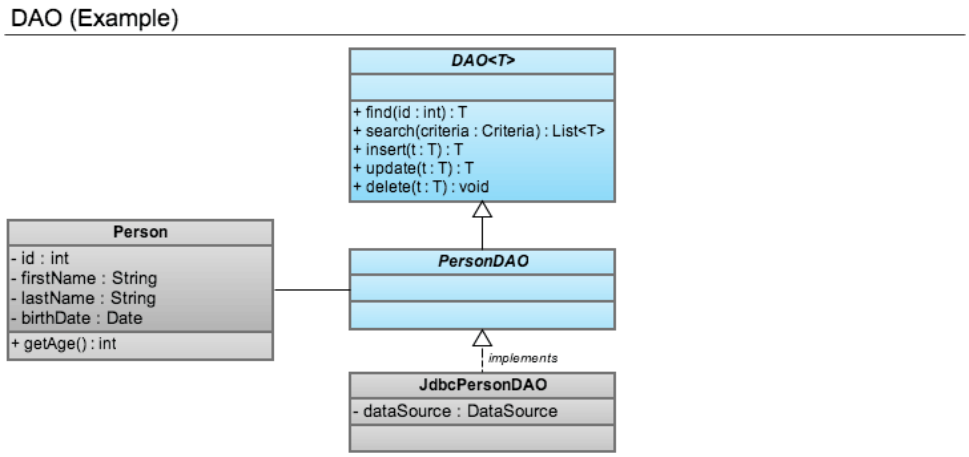

Figura 43 Estructura básica del patrón DAO

Fuente: davejoyce.github.io/patterns/2013/04/24/composite-dao-pattern

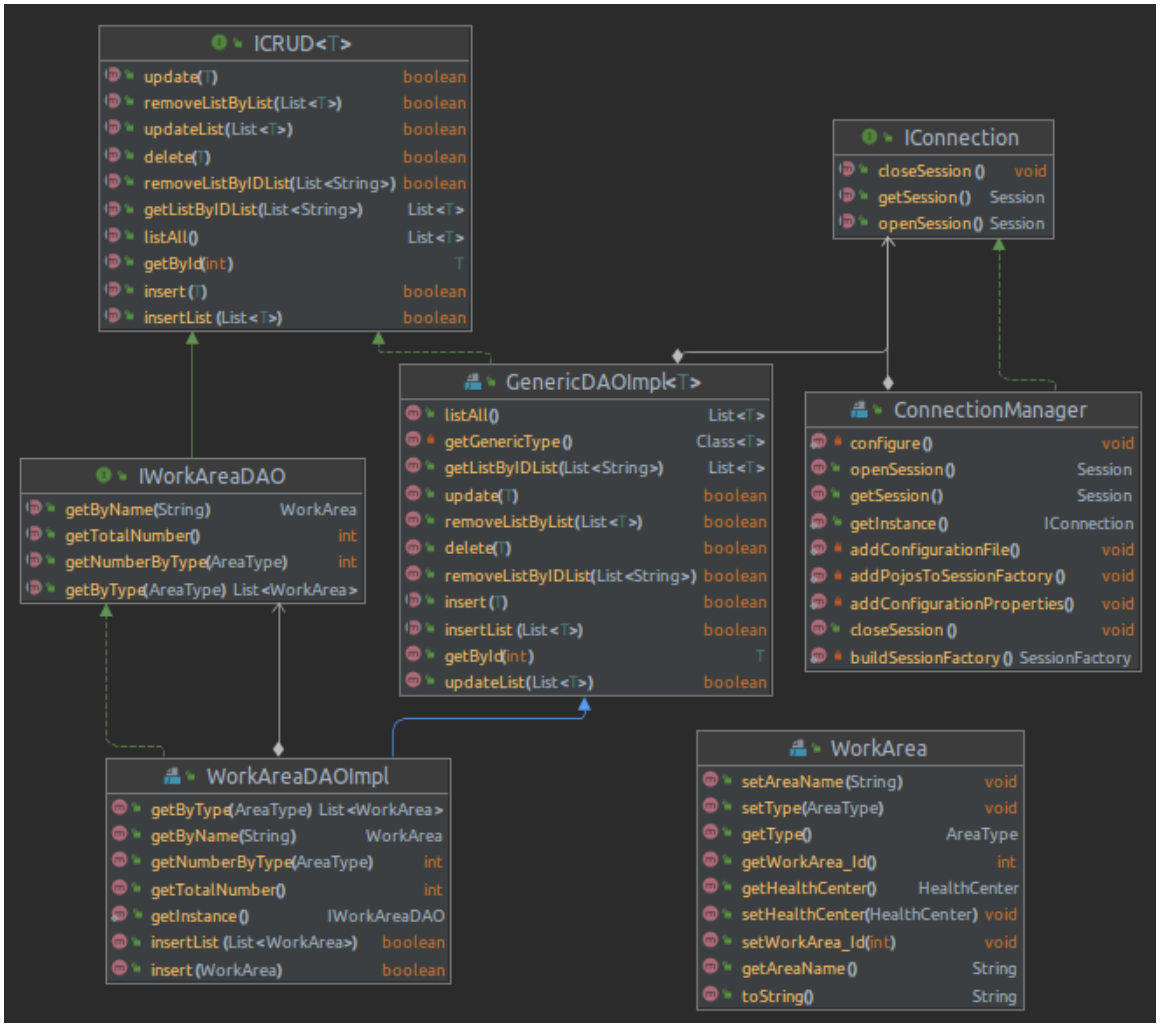

Figura 44 Diagrama de clases patrón DAO

Fuente: Captura de pantalla directa del IDE IntelliJ

En la Figura 44 y para explicar de forma concreta este patrón, hemos utilizado la entidad WorkArea y todo lo relacionado con ella. Se han descrito también las operaciones que puede realizar cada clase para una mayor comprensión del funcionamiento.

Gracias a este patrón si el origen de datos cambiase o hubiese que alternar varios orígenes no habría más que cambiar el ConnectionManager. Si hubiese que cambiar el intermediario que en este caso es Hibernate a otro deberíamos cambiar el WorkAreaDAOImpl por otra clase adaptada a ese nuevo intermediario. En resumen, con pocas modificaciones podemos realizar las adaptaciones que serán necesarias para la capa de acceso a datos facilitando enormemente el mantenimiento de la aplicación haciendo que las clases encargadas de la lógica que necesiten trabajar con las entidades no tengan que preocuparse por nada relacionado con la obtención y almacenaje de datos.

Podemos distinguir en la Figura 44 las siguientes partes:

- DataSource

  O dicho de otra forma el origen de datos propiamente dicho que viene dado por la clase ConnectionManager. Esta clase configura el acceso a MySQL utilizando un ORM, en este caso Hibernate.

- Operaciones generales

  Descritas en la clase GenericDAOImpl vemos todas las operaciones que podremos aplicar a todos los objetos que utilicen el patrón y no solo a nuestro WorkArea. Esto es posible mediante el uso de genéricos <T>.

- Operaciones particulares

  Evidentemente cada una de las entidades deben permitir interacciones con el origen de datos de una forma diferente si fuese necesario. Esto queda patente en el WorkAreaDAOImpl el cual permite operaciones como obtención por tipo o por nombre mientras que, es posible que para otras entidades estas operaciones carezcan de sentido.

- Entidad

  Es la razón de ser y de utilizar este patrón, poder realizar operaciones CRUD con un origen de datos sobre esta entidad.

7.3.4 Patrón fachada

Este patrón tiene la finalidad de agrupar una serie de funcionalidades de distinto origen en un único objeto consiguiendo que para los interesados al acceso a ellas sea mucho más simple al estar centralizado. Las fachadas permiten que el acoplamiento sea dirigido ya que en caso de necesidad de cambios y mantenibilidad está claro donde tenemos que tocar.

La otra función principal es la transparencia y abstracción de las funcionalidades que se centralizan, ya que en caso de cualquier tipo de modificación esto será transparente para el consumidor final de la función al ser necesario cambiar únicamente la llamada en la fachada.

Para nuestro proyecto la decisión fue establecer una fachada entre las clases de operaciones y el acceso a datos, esta clase fue bautizada como CRUDManager. El nombre hace clara referencia a que todas las operaciones CRUD pasarán por allí.

Todas las clases de operaciones tendrán acceso a una única instancia de CRUDManager (ya que está configurado como singleton) el cual conectará con cada uno de los DAO, de esta forma, si estos cambiasen las clases que los utilizan no serían conscientes de ello no teniendo que modificar nada de su código para poder seguir realizando la misma operación.

La Figura 45 nos muestra una función incluida en nuestra fachada que servirá para ilustrar las explicaciones anteriores. Podemos observar como todas las llamadas están delegadas a una fábrica la cual sería el objeto del cambio en caso de modificación, pero totalmente transparente para el resto del programa.

```
public boolean updateEntity(EntityName entityName, Object obj) {

    switch (entityName) {
        case BankHolyday:
            return factory.getBankHolydayDAO().update( t: (BankHolyday) obj);
        case DoctorAndSchedule:
            return factory.getDoctorAndScheduleDAO().update( t: (DoctorAndSchedule) obj);
        case Employee:
            return factory.getEmployeeDAO().update( t: (Employee) obj);
        case HealthCenter:
            return factory.getHealthCenterDAO().update( t: (HealthCenter) obj);
        case HolydaysOfEmployee:
            return factory.getHolydaysOfEmployeeDAO().update( t: (HolydaysOfEmployee) obj);
        case JobSubstitution:
            return factory.getJobSubstitutionDAO().update( t: (JobSubstitution) obj);
        case Salary:
            return factory.getSalaryDAO().update( t: (Salary) obj);
        case ShiftInSchedule:
            return factory.getShiftInScheduleDAO().update( t: (ShiftInSchedule) obj);
        case WorkArea:
            return factory.getWorkAreaDAO().update( t: (WorkArea) obj);
        case WorkGroup:
            return factory.getWorkGroupDAO().update( t: (WorkGroup) obj);
        case WorkShift:
            return factory.getWorkShiftDAO().update( t: (WorkShift) obj);
        default:
            LogHandler.logMessage( classType: this.getClass(), type: LogLevels.ERROR, message: "El tipo de entidad "
                    + entityName.name() + " no existe en getList");
            return false;
    }
}
```

Figura 45 Función de actualización de entidad. Patrón fachada

Fuente: Captura de pantalla directa del IDE IntelliJ

7.4 Conceptos aplicados al desarrollo de software

El buen desarrollo de software está bastante reñido con la casualidad. Durante su formación un ingeniero informático es instruido en muchos conceptos que posteriormente se aplican a la forma de codificar, sin ellos es posible que el código funcione igualmente pero el efecto espaguetti [30] está asegurado.

Todas las técnicas y conceptos tratados durante todo este capítulo y durante el capítulo 5 relativo a la metodología de desarrollo orbitan sobre como conseguir un código con dos propiedades descritas aquí que son la reutilización y la mantenibilidad.

7.4.1 Acoplamiento

Este término hace referencia al grado de interconexión de los objetos. El grado de acoplamiento determina lo difícil que es realizar cambios en una aplicación. El acoplamiento es inevitable en términos generales, pero debemos intentar reducirlo al máximo para realizar un código más fácil de mantener.

Una buena práctica es aceptar un cierto nivel de acoplamiento, pero dirigirlo hacia una un punto concreto como una fachada o un intermediario (Barnes & Kölling, 2013)

Durante todo el proyecto se ha intentado creo que con éxito el poder mantener un acoplamiento relativamente bajo durante todo el código a la vez que, se han utilizado fachadas o patrones de diseño que han permitido que al menos el acoplamiento estuviese dirigido.

Imaginemos que tenemos que calcular la capacidad de salto de un animal en función de tres o cuatro parámetros. No es lo mismo pasarle a esa función los parámetros directamente que pasarle la entidad entera y que esa función los trate. Si pasamos únicamente los parámetros esa función será reutilizable para más tipos distintos de animales, pero si pasamos la entidad concreta únicamente nos valdrá para esa entidad, esto representa un bajo acoplamiento y un alto acoplamiento respectivamente.

7.4.2 Cohesión

Este término hace referencia al número de tareas y la diversidad de las tareas de las que es responsable cada unidad de una aplicación. Es relevante a cualquier nivel bien sea unidades formadas por una sola clase, como para métodos individuales. (Barnes & Kölling, 2013)

De esta forma si por ejemplo trabajamos con una clase encargada del tratamiento de imágenes, es deseable que todas las operaciones encargadas con ese tratamiento estén ubicadas en él. También sería deseable que no contuviese ninguna operación que no tuviese que ver con ello.

En caso de necesitar operaciones que no tienen que ver con la funcionalidad principal debemos sacarlas de ahí teniendo cuidado de no aumentar el acoplamiento, como ya hemos visto anteriormente.

[30] Efecto espaguetti – Forma de codificación deficiente basada en la creación código lineal sin estructurar de gran longitud metafóricamente hablando igual a un espaguetti

7.4.3 Granularidad

Cada unidad de código debería ser responsable de una única tarea coherente dentro del mismo nivel de abstracción. Esto quiere decir que si una tarea necesita apoyarse en otra más pequeña (una o las veces que sea necesario), esta tarea ha de separarse de ese flujo de código y ser llamado desde él. De esta forma favorecemos la reutilización ya que estas unidades más pequeñas pueden ser utilizadas por diferentes tareas de un nivel de abstracción más alto.

También facilita el mantenimiento ya que, si quisiéramos modificar una operación concreta, será más fácil de localizar si está ubicada en un sitio correcto, haciendo una única tarea.

Decimos que cuanto más separados estén los componentes tendremos una granularidad "más fina", si por el contratrio los componentes fuesen muy extensos y sin separación diríamos que son de "grano grueso".

7.4.4 Reutilización

Más que un concepto, la reutilización es un deseo y un fin asociado a un código bien escrito por esto, tener código duplicado es algo que todo programador debería evitar. Si estamos reutilizando código con asiduidad y de forma coherente también estaremos cumpliendo, casi automáticamente, todos los conceptos de los que hemos hablado anteriormente.

Es complejo querer resumir todos estos conceptos en la cabecera de una función, pero de todo el código escrito en este proyecto, la Figura 46 quiza sea la que mejor lo exprese:

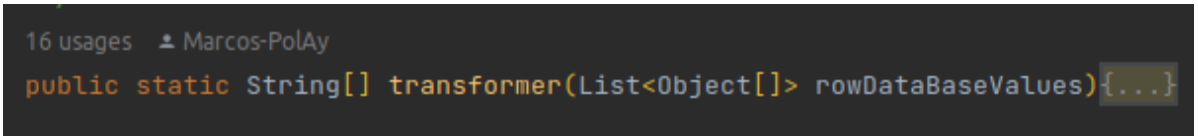

Figura 46 Cabecera de una función, ejemplo de reutilización

Fuente: Captura de pantalla directa del IDE IntelliJ

En la cabecera de la función representada en la Figura 46 detectamos los conceptos descritos con anterioridad. Gracias al IDE de desarrollo utilizado (intelliJ) vemos de forma automática que esta función se utiliza en 16 ocasiones. Si una función tiene muchos usos y sus parámetros son decriptivos sabremos que la función está cumpliendo con su cometido.

7.4.5 Mantenibilidad

He decidido dejar este concepto para el final ya que bajo mi punto de vista es el más importante de todos los que estamos tratando. Que un código tenga una buena mantenibilidad quiere decir que en caso de tener que hacer modificaciones, ampliaciones, reconfiguraciones o cualquier cosa en general que vaya a cambiar la aplicación de una u otra forma, no sea necesario cambiar muchas líneas de código.

Si un cambio en una función requiere cambiar código en varias clases quiere decir que esa función no está bien diseñada. Si para cambiar una configuración de, por ejemplo, una conexión a base de datos tienes que hacer múltiples cambios en lugar de crear una nueva configuración y asignarla en un único sitio, querrá decir también que tenemos uno o más problemas de diseño.

El uso de patrones de diseño facilita enormemente la mantenibilidad del código ya que además de ser una guía seguir cuando se codifica también será un estándar de búsqueda en caso de necesidad de modificación.

7.5 Debug

Pretender programar y que el código haga lo que queremos a la primera cuando el problema es medianamente complejo es algo utópico en la mayoría de los casos. Por ello necesitamos tener un sistema para poder "debugear" el código, que no es más que ir haciendo paradas o avisos por cada acción hasta que averiguamos donde está el error.

Para poder depurar nuestro código solemos seguir un proceso más o menos similar pero que difiere en como de automático y delegado es este proceso.

La forma más rudimentaria para depurar código es la impresión de una línea en consola con los valores actuales cada vez que sucede una acción. De esta forma si los valores coinciden con nuestros deseos es que hasta ahí no ha habido errores, si los valores no coinciden el error estará en esa zona de código.

Para poder depurar he utilizado un sistema que no solo vale para esto si no también para poder mantener una trazabilidad completa y un sistema de avisos utilizando un Logger. Se explica de forma detallada en Anexo 1 Logger de este documento.

Capítulo 8 - Cronograma y costes

Una vez explicado el proyecto casi en su totalidad, debemos plantear cuanto tiempo nos ha llevado y el coste que esto hubiese tenido.

8.1 Cronograma

Para poder hablar del tiempo que nos ha llevado este proyecto recurro a un diagrama de Gantt que podemos ver en la Figura 47

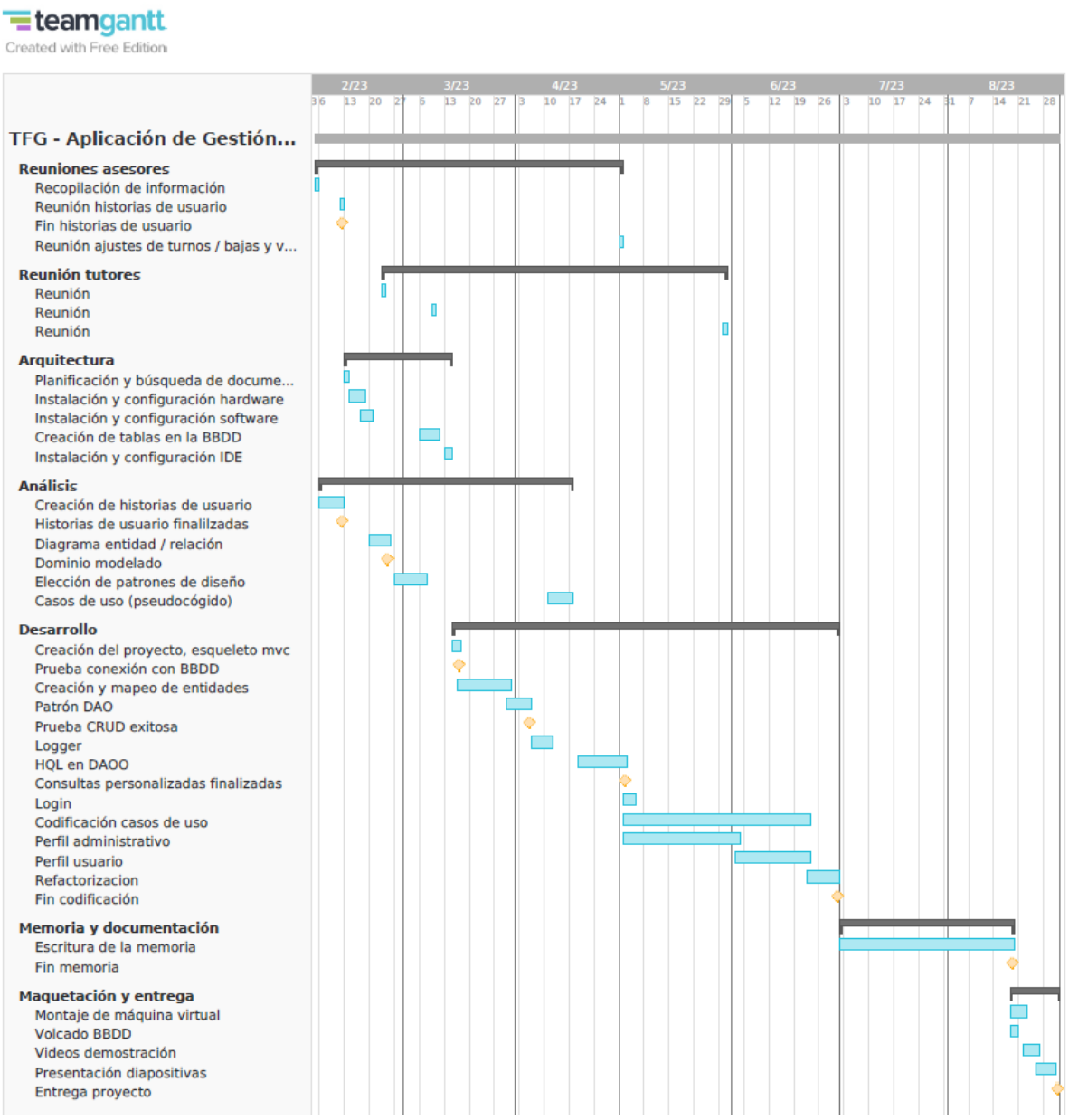

Figura 47 Cronograma del proyecto. Diagrama de Gantt

Fuente: Creación propia utilizando el software de TeamGantt.com

Además de todo lo destacado en este diagrama temporal, no se han incluido las primeras reuniones con los tutores que comenzaron en mayo del 2022, al quedar tan lejanas del inicio propiamente dicho del proyecto hubiesen desenfocado el diagrama.

8.2 Presupuesto

Para poder calcular los costes valoramos salario, costes equipos, licencias software y gastos complementarios. Entendamos que muchos de estos gastos son estimaciones y no podemos ajustarlos 100 % a lo que hubiese sido la realidad de un caso real.

8.2.1 Salario

Dado que tenemos un convenio colectivo laboral se han utilizado sus tablas de salario. Si bien es cierto que en un caso real tendríamos diferentes tipos de salarios porque habría varios tipos de trabajadores con distintos perfiles, por simplicidad he decidido aglutinarlo en uno solo.

El grupo al que yo considero que mejor se podría adaptar esta situación ya que he tenido que encargarme yo de todo si exceptuamos del asesoramiento por parte de los tutores del proyecto es el grupo B2 cuya definición es la siguiente:

> "Grupo B:
>
> Pertenecen a este grupo profesional las personas que, tienen atribuidas funciones relacionadas con el análisis, definición, coordinación y supervisión de proyectos, tareas, actividades propias del sector, línea, área a las que pertenece, velando por la consecución de los objetivos perseguidos, y que dispongan de la necesaria formación, conocimiento y experiencia profesional. Planifican y gestionan, por proyecto, los recursos humanos y técnicos disponibles.
>
> Desarrollan sus funciones con autonomía y capacidad de supervisión media-alta.
>
> Nivel 2.
>
> Personas con el perfil profesional adecuado, con poca experiencia profesional en las tareas del grupo y que poseen los conocimientos necesarios. Actúa con autonomía en la ejecución de sus tareas. Aplica iniciativa en las tareas asignadas. Supervisa y asigna tareas a personas a su cargo." (BOE, 2023)

Si aceptamos un cálculo del salario en función de los 8 meses que ha llevado este proyecto entre planificación y ejecución, teniendo según la tabla salarial el grupo B2 un salario anual bruto de 26614,83 obtenemos el siguiente coste:

$$26614{,}83 * 8 / 14 = 15.208\,€$$

Esto es así porque la tabla salarial nos computa 14 pagas al año y nosotros hemos trabajado 8 meses. No obstante, entendamos que este es el sueldo bruto y habría que aplicar las deducciones fiscales reales correspondientes.

8.2.2 Licencias

Aclaramos que realmente al ser alumno universitario todas las licencias que tienen un coste económico real han sido gratuitas. Estos costes intentan representar el gasto real.

Se han utilizado varios softwares que tienen uso gratuito, son los siguientes: Java openJDK, Draw.IO, GitHub, MySQL, HeidiSQL, Linux ubuntu.

Sin embargo, se han utilizado dos productos que si tienen un coste de licencia:

- Microsoft office

  Porque se ha utilizado Microsoft Word para el desarrollo de esta memoria y Microsoft OneDrive para el almacenamiento CLOUD.

  Para calcular el coste de la licencia utilizamos la versión personal con coste anual

  $$69{,}9 \text{ €} / 12\ meses = 5{,}75 \text{ €} \cdot mes$$

  Estimando que el proyecto dura de marzo a junio tenemos 4 meses y por tanto

  $$5{,}75 \text{ €} \cdot 4\ meses = 23 \text{ €}$$

- IDE de desarrollo

  Se ha utilizado la versión Ultimate del software IntelliJ fabricado por la empresa jetbrains

  $$724{,}79 \text{ €} / 12\ meses = 60{,}4 \text{ €}\ al\ mes$$

  Como antes tenemos en cuanta los 4 meses de uso y obtenemos

  $$60{,}4 \text{ €} \cdot 4\ meses = 241{,}6 \text{ €}$$

8.2.3 hardware

Para este proyecto se han utilizado tres equipos con los siguientes costes:

- Equipo portátil 349 €
- Equipo sobremesa 473,04 €
- Servidor linux 384,05 €

$$349 + 473{,}04 + 384{,}05 = 1206{,}09 \text{ €}$$

8.2.4 Gastos complementarios

Si bien es difícil establecer gastos como electricidad, transporte, dietas … he tomado la decisión de establecer estos costes en 80 € al mes.

$$80\ € \cdot 4\ meses = 320\ €$$

Una vez unimos todos los conceptos tenemos que los costes totales del proyecto.

8.2.5 Coste total

Concepto	Coste
Salarios	15208
Licencias	264,6
Hardware	1206,09
Gastos generales	320
Costes totales	16998,69 €

Figura 48 Tabla de costes del proyecto

Capítulo 9 – Conclusiones

La totalidad de las funcionalidades principales proyectadas han sido desarrolladas. El software es plenamente funcional y estable tras realizar un gran batería de pruebas.

En cuanto a su funcionamiento, hay que destacar la rapidez de procesamiento. El coste temporal de elaborar un mes de turno de trabajo no excede los 10 segundos de tiempo y los cambios más complejos por bajas y vacaciones por reasignación de personal no exceden los 5 segundos. Por lo que si bien aceptamos que todo es mejorable podemos afirmar que el rendimiento está muy cerca del punto óptimo.

9.1 Posible aplicación

Tras finalizar el desarrollo de la aplicación se probó en un escenario real y los resultados fueron más que satisfactorios. Sin embargo, el programa no podría ser utilizado en un escenario real sin tener que aplicar una serie de ampliaciones que dotasen de las mil y una particularidades que tiene un entorno laboral real.

El proyecto considera un escenario puramente horizontal, es decir, todos los médicos tienen las mismas obligaciones laborales, sin restricciones. Todos tienen turnos de mañana, tarde o guardia, pero como podemos suponer, esto dista mucho de la realidad.

Cito a modo de documentación restricciones típicas asociadas a los sanitarios:

- Exención de guardias por edad

  Depende totalmente de la comunidad autónoma y también del propio centro en el que se trabaja. Lo habitual es que cuando la edad es cercana a los 60 ya no se hagan guardias o si se hacen sea de forma voluntaria.

- Reducciones de jornada por cuidados

  Sobre todo, está asociado a la paternidad o maternidad, pero también en cualquier caso de cuidados de personas dependientes a tu cargo. En estos casos los servicios intentar organizarse de tal forma que los grupos de trabajo estén balanceados y no tengan más de un miembro con este tipo de restricciones.

- Restricciones por contratos antiguos

  Un caso curioso encontrado durante mi investigación fue un empleado, cercano a la edad de jubilación, cuya plaza de funcionario venía de un proceso de consolidación bastante peculiar y complejod de explicar. Su peculiaridad era que, si bien podía hacer guardias nocturnas, no podía hacer tardes, es decir, podía trabajar de 8 a 15 y por la noche de 00 a 08 pero no se le podía asignar un turno fuera de esas horas. Aunque estos casos son muy extraños, en un escenario real deberíamos implementarlo.

- Minusvalías

  El tener algún tipo de discapacidad no te inhabilita para trabajar, pero sabemos que para que un servicio pueda ser funcional y no sufrir mermas en su rendimiento ha de intentar balancearse este tipo de situaciones distribuyendo personas con minusvalías entre los distintos grupos. De esta forma se logran dos objetivos: Primero una mejor integración del sujeto y segundo un mejor funcionamiento del servicio.

- Paridad de sexos

  Si bien a priori no podemos nunca discriminar ni hacer distinciones de trabajo por sexo el ámbito sanitario tiene particularidades en este sentido. En todos los centros de los que recabé información me hicieron constar que existen acuerdos internos para intentar equilibrar el número de hombres y mujeres por turno siempre que sea posible. La razón que me dieron es que en multitud de ocasiones ciertos pacientes se han sentido más cómodos siendo atendidos por médicos de un sexo determinado. Además de por comodidad también se incluyen otros factores como el tema religioso asociado a este tipo de situaciones, por ello suele ser tenido en cuenta siempre que sea posible.

- Liberados sindicales

  Al igual que en todas las empresas de un cierto tamaño existe un número de empleados liberados para ser representantes sindicales del resto de sus compañeros. Sería necesario definir uno o varios escenarios a medida para ellos ya que existen muchas posibilidaes tales com el número, si están liberados para la función salarial o nó, si esto les comporta estar exentos de guardias o no … En resumen, habría que establecer varias restricciones para esta variable concreta.

9.2 Trabajos futuros

Como explicamos en el primer capítulo durante la presentación de este proyecto, este software es un gestor de turnos que es una función que suele venir incluida en muchos ERP más amplios. Por esto mismo, hemos de pensar que muchas de sus evoluciones y ampliaciones deben ir enfocadas en esa dirección.

Una cualidad habitual en los ingenieros es la ilusión y las altas expectativas al pensar que queremos que haga nuestra aplicación. En este momento surgen muchas ideas que obviamente no se llevan a cabo ya que exceden el alcance del trabajo.

- Sistema multiplataforma

  Desde el punto de vista de la interfaz y la comunicación con el usuario siempre es deseable más modos de acceso ya que en este trabajo solamente disponemos de una interfaz de escritorio. Gracias al diseño en capas poder implementar acceso vía smartwatches (independientemente de su plataforma) es una ampliación lógica.

- Gestión de nóminas

  Habitualmente el sueldo de los empleados sanitarios está basado en muchos complementos asociados a guardias, festivos y fines de semana. Al tener acceso a todo el calendario no sería dificil calcular las nóminas de todos los empleados basándonos en unas constantes típicas como el sueldo base y añadirle posteriormente los complementos salariales.

- Impresión de documentos

  Otra funcionalidad muy típica que no se valoró en un primer momento por no ser necesaria es la posibilidad de imprimir los calendarios en un formato estándar como pdf o similar. No supondría tampoco un gran esfuerzo debido a que es una función muy habitual y existe mucha documentación al respecto.

- Comunicación por correo electrónico

  Existen tres cambios en los datos que son de interés de notificación para los médicos: creación de un calendario mensual, modificación de un calendario, estado de solicitudes. Como la aplicación no es extensa y se consulta con facilidad no fue considerado un imprescindible. También influyó para su descarte la necesidad de contar con un servidor de correo lo que cual supondría más tiempo de configuración y mantenimiento que no es propio de la aplicación.

Bibliografía

Barnes , D., & Kölling, M. (2013). *Programación orientada a objetos con java usando BlueJ.* Pearson Education.

Beaulieu, A. (2009). *Learning SQL.* O'Reilly.

Blasco, F. (2020). *Programación java: JDBC y Swing.* Ra-Ma.

BOE. (26 de Julio de 2023). *Boletín Oficial del Estado.* Obtenido de https://www.boe.es/boe/dias/2023/07/26/pdfs/BOE-A-2023-17238.pdf

Cohn, M. (2006). *Agile Estimating and Planning.* Prentice Hall.

Drebauwer, L. (2023). *Patrones de diseño en Java. Los 23 modelos de diseño: descripciones y soluciones ilustradas en UML 2 y Java.* Ediciones ENI.

Drebauwer, L., & Van der Heyde, F. (2020). *UML 2.5 Iniciación, ejemplos y ejercicios corregidos.* Ediciones ENI.

Groussard, T., & Richard, T. (2020). *Java 11 Los fundamentos del lenguaje Java.* Ediciones ENI.

HeidiSQL. (s.f.). Recuperado el 19 de 9 de 2023, de Wikipedia, la enciclopedia libre: http://es.wikipedia.org/wiki/HeidiSQL

Hibernate. (s.f.). Recuperado el 19 de 9 de 2023, de Wikipedia, la enciclopedia libre: http://es.wikipedia.org/wiki/Hibernate

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process.* Pearson Education.

Java. (s.f.). Recuperado el 19 de 9 de 2023, de Wikipedia, la enciclopedia libre: http://es.wikipedia.org/wiki/Java_(lenguaje_de_programación)

Largman, C. (2022). *Applying UML and Patterns: An introduction to Object-Oriented Analisys and Design and the Unified Process.* Pearson Education.

Martin, R. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship.* Pearson Education.

MySQL. (s.f.). Recuperado el 19 de 9 de 2023, de Wikipedia, la enciclopedia libre: http://es.wikipedia.org/wiki/MySQL

Parsons, D. (2009). *Desarrollo de aplicaciones web dinámicas con XML y Java.* Anaya multimedia.

Pérez Martínez, E. (2015). *Conoce todo sobre Hibernate, persistencia de objetos en JEE.* Ra-Ma.

Pérez Martínez, E., & Altadill Izura, P. (2018). *Manual Imprescindible, Spring 5.* Anaya multimedia.

Piattini Velthuis, M., García Rubio, F., García Rodríguez de Guzmán, I., & Pino, F. (2018). *Calidad de Sistemas de Información* . Ra-Ma.

Refsnes Data. (s.f.). *w3schools.com*. Obtenido de https://www.w3schools.com

Tarnum Java SRL. (s.f.). *baeldung.com*. Obtenido de https://www.baeldung.com

Tutorials Point India Private Limited. (s.f.). *tutorialspoint.com*. Obtenido de https://www.tutorialspoint.com/

Wikipedia. (31 de Diciembre de 2022). *Log4J.* Obtenido de Wikipedia, la enciclopedia libre: https://es.wikipedia.org/wiki/Log4j

Anexos

Anexo 1 Logger

Se decide implantar un sistema de log que hace las funciones tanto de depurador como de log de sucesos propiamente dicho.

La mayoría de los lenguajes de programación actuales incluyen algún tipo de librería para trabajar con este tipo de log. En este caso, Java trabaja con una librería llamada log4J desarrollada por Apache.

Log4J está preparado para ofrecernos los siguientes mensajes:

- FATAL:

  Mensajes críticos del sistema. Es deseable que el programa finalice después de esto de forma controlada y que la configuración sea meticulosamente correcta si no el sistema podría colapsar.

- ERROR:

  Mensajes de error de la aplicación. Debe usarse en situaciones que sin colapsar el sistema podrían hacer que la aplicación se cuelgue. Como ejemplo tenemos desde el típico numberFormatException de las operaciones aritméticas hasta el fallo de conexión con una fuente externa.

- WARN:

  Mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan al correcto funcionamiento del programa. En ocasiones este tipo de mensajes suelen llevar asociado asignación automática de valores.

- INFO:

  Utilizado con la intención de seguir una trazabilidad del flujo del programa.

- DEBUG:

  Utilizado para escribir mensajes de depuración. Habitualmente estos mensajes se borran una vez que se solucionan los problemas en la parte del código que se está depurando.

- TRACE:

  Se utiliza para mostrar mensajes con un mayor nivel de detalle que debug.

Esta escala va de menor a mayor, es decir, cada nivel siguiente incluye los mensajes del anterior.

Existen varias posiblidades de implantación de un log, pero yo me decidí por utilizar el patrón wrapper que consiste en envolver el objeto que queremos utilizar, en este caso un objeto log4j, utilizando un contexto estático. De esta forma el objeto estará disponible para toda la aplicación y se le puede proporcionar una mayor funcionalidad.

Otra ventaja de utilizar log4j es que permite varias salidas diferentes simultáneas, pudiendo establecerse no solo la salida de consola si no también a un log de texto.

Anexo 2 El tablero Scrum

Una herramienta que no está detallada como tal dentro de la metodología pero que se adapta casi a cualquier aspecto organizativo en la vida es el tablero Scrum. En su versión más reducida de tres estados (To Do, Doing, Done) sirve para llevar una organización eficiente de las tareas.

Para este proyecto se utilizó a menudo microsoft teams[31] que tiene implementado un tablero scrum simple como el que vemos a continuación en la Figura 49.

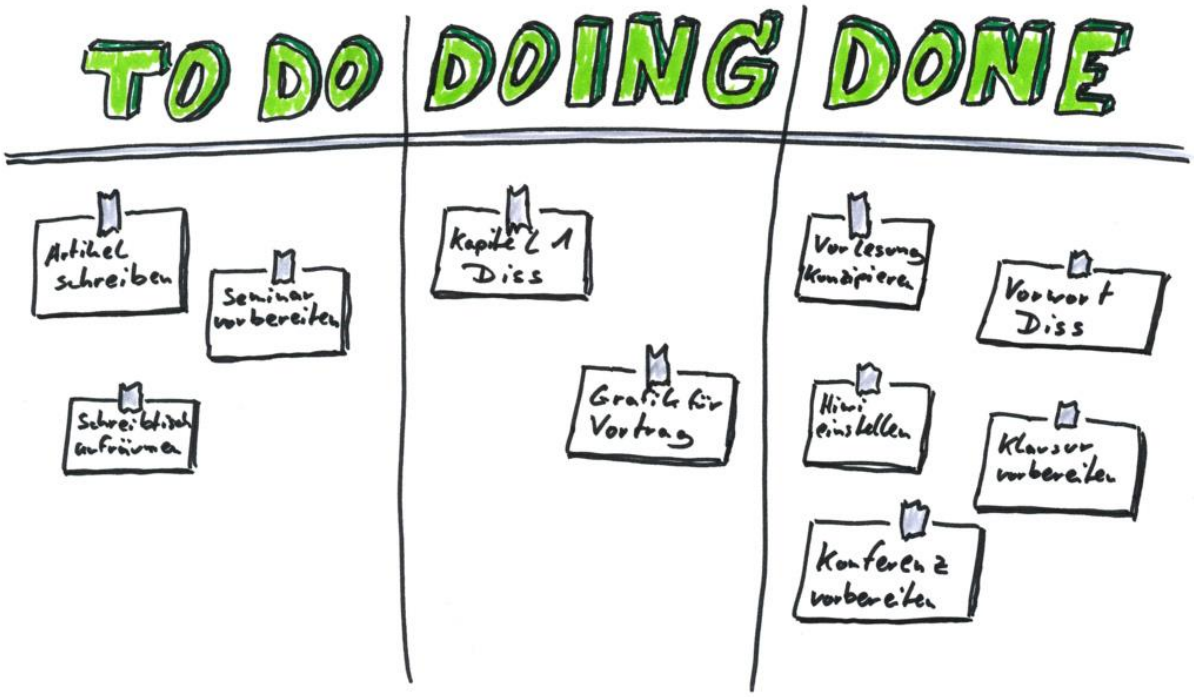

Figura 49 Tablero Scrum

Fuente: organizzazione-qualita.com

Este tablero fue utilizado continuamente ya que cada vez que una nueva tarea era detectada era incluida en él. De esta forma y mediante un sistema de prioridades se llevaba un control de las tareas dentro de cada iteración.

[31] Microsoft Teams – Software empresarial especializado en desarrollo de actividades colaborativas

Anexo 3 Manual de usuario

En esta sección se añade el manual de usuario desde un punto de vista lineal tratando primero el perfil administrativo y posteriormente el de un médico.

Partimos de una situación en la cual hay suficientes médicos registrados en el sistema pero que aún no están contratados por el centro sanitario. No existen grupos, ni turnos, ni ningún otro tipo de registro.

Perfil administrativo

Este perfil tiene las siguientes pantallas de operaciones a elegir, utilizamos la Figura 50 para ilustrarlas y no tener que repetirlas en cada uno de los pasos siguientes:

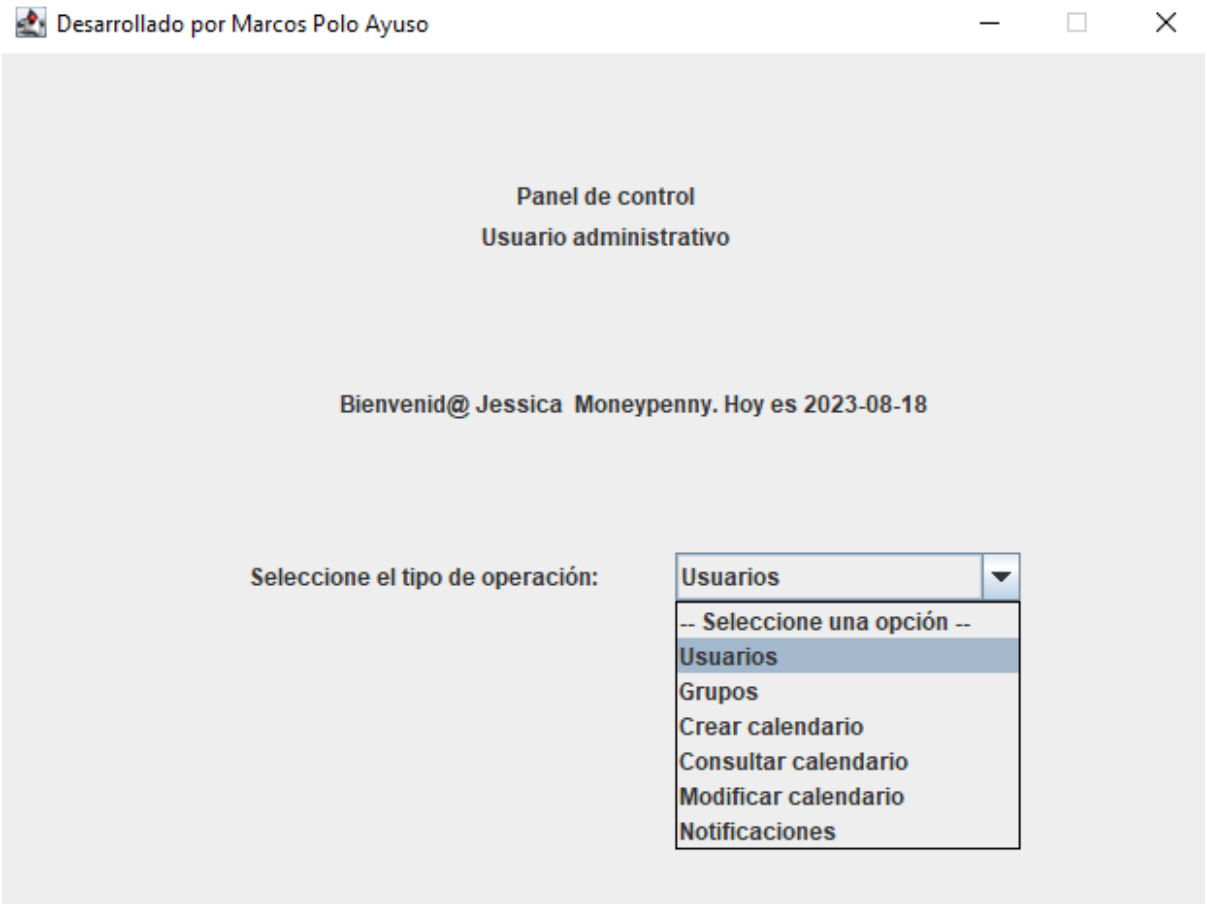

Figura 50 Pantalla de selección del perfil administrativo

1. Añadir / modificar / contratar usuarios en el sistema, ilustrado en la Figura 51

   El personal administrativo puede añadir médicos los cuales no estarán contratados directamente, si no que únicamente serán dados de alta en el sistema. Para contratarlos ha de seleccionarse entre los médicos disponibles y contratarlos.

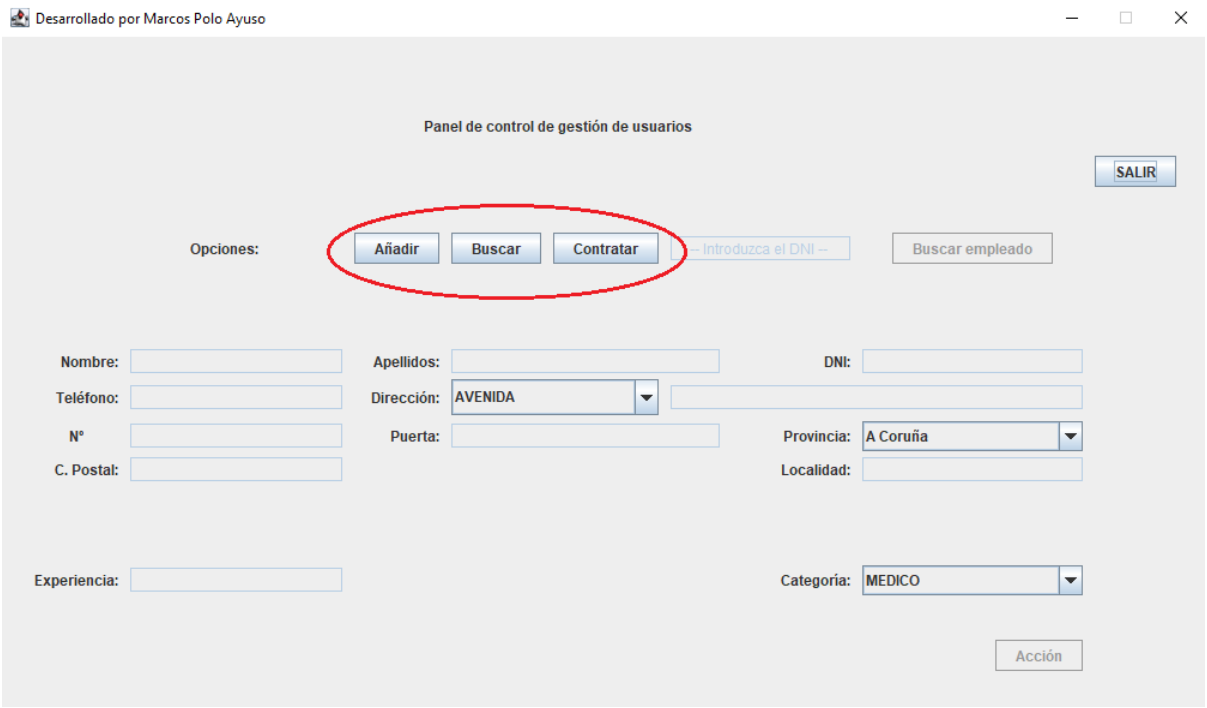

Figura 51 Pantalla de gestión de usuarios, perfil administrativo

Como vemos señalado, mediante esos controles se puede añadir, buscar o contratar a los médicos o a nuevos oficinistas

2. Creación de grupos

Debemos elegir cuantos grupos vamos a tener el sistema, el número de grupos va asociado a la cadencia de guardias que harán los médicos. Si tenemos 8 grupos, el primer grupo hará la primera guardia hasta que le vuelva a tocar que será 8 días después. Análogamente pasará lo mismo si la cadencia es de 6 solo que, obviamente 6 días después en lugar de 8.

En la Figura 52 vemos como si no tenemos médicos suficientes el sistema nos lo notificará con un mensaje de advertencia

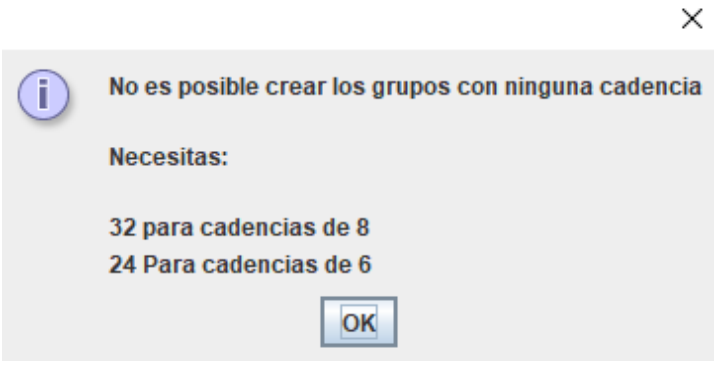

Figura 52 Notificación de personal insuficiente

Para lo cual debermos contratar más médicos como se indica en el paso 1. En función de si tenemos ya médicos suficientes para las dos posibles cadencias tendremos mensajes como los que nos ilustra la Figura 53 y la Figura 54

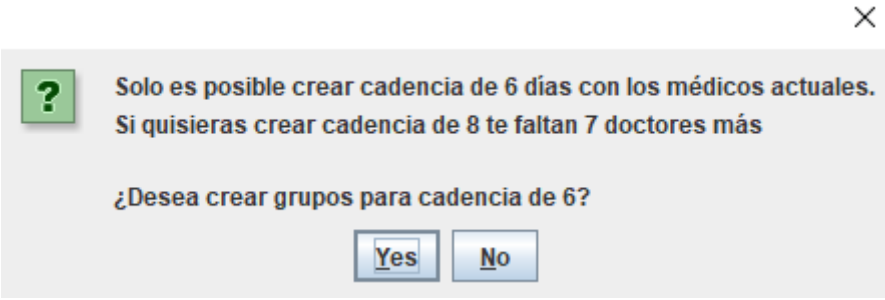

Figura 53 Imagen de confirmación grupo 6

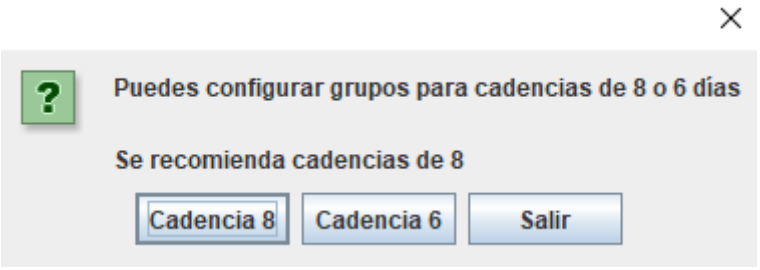

Figura 54 Imagen de confirmación de grupos

Para el ejemplo de este manual diseñaremos 8 grupos. Una vez creados los grupos, la pantalla de grupos que obtenemos permite intercambiar médicos de grupo como nos ilustra la Figura 55. Si se intercambia un médico de grupo, sus turnos seguirán siendo los mismos debiendo revisarse manualmente para corregir incoherencias si las hubiese.

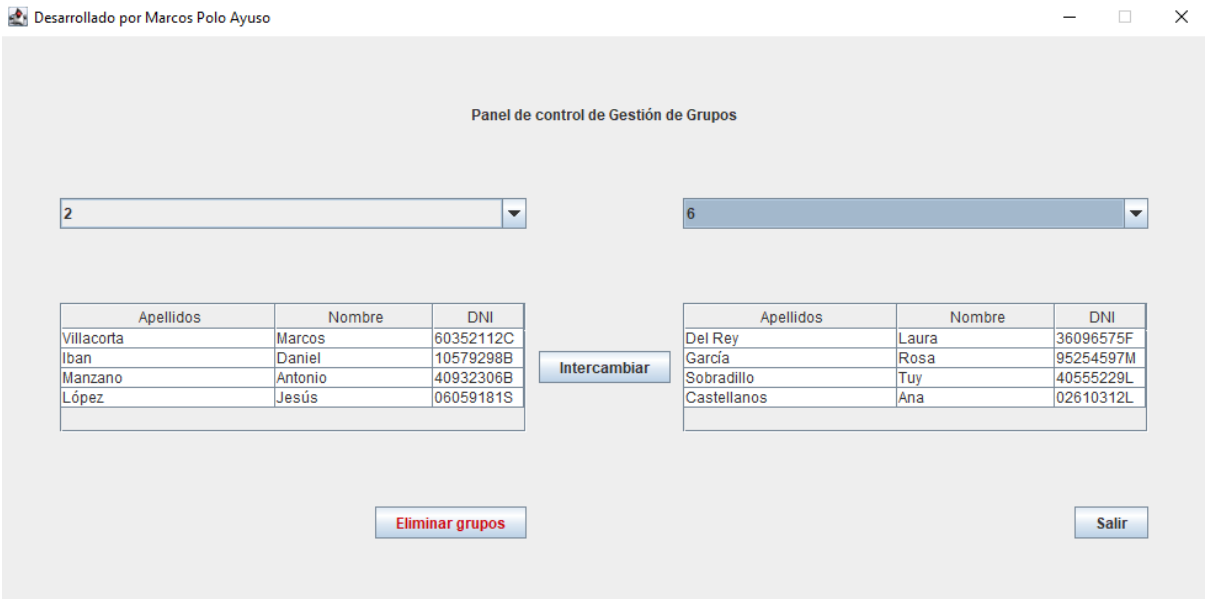

Figura 55 Panel de gestión de grupos

3. Creación de turnos

   Una vez creados los grupos ya podemos crear los turnos. La forma de crear los turnos es realmente sencilla como vemos en la Figura 56. El sistema está preparado para que un mes encaje con el anterior si existiese. Los turnos no pueden eliminarse una vez creados, pero si podrán modificarse manualmente con posterioridad por el administrativo. Para saber si un turno ha sido modificado o no disponemos de un campo bit llamado "original" en la tabla MedicoHorario de la BBDD.

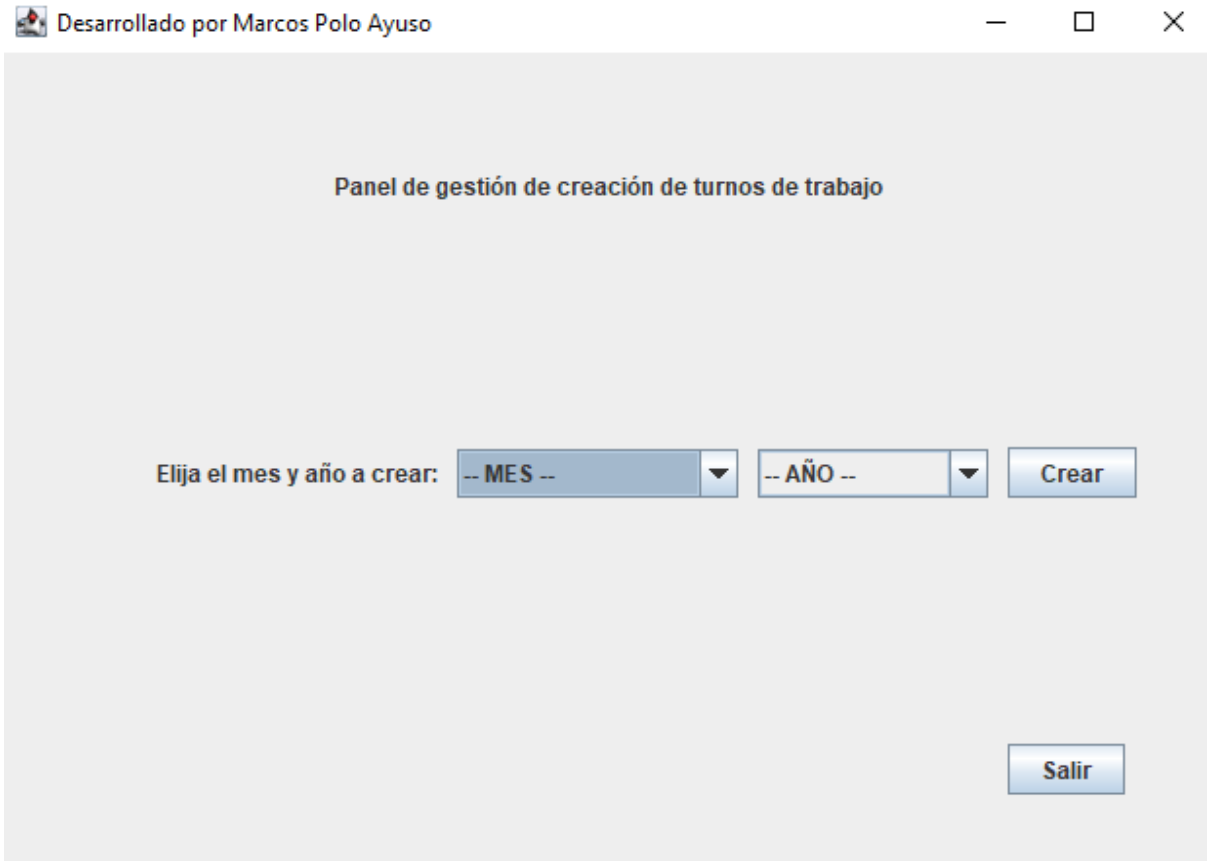

Figura 56 Panel de creación de turnos

4. Consultar el calendario

Una vez que se ha creado un turno mensual es posible consultarlo con el fin de poder revisarlo y valorar posibles modificaciones manuales posteriores como vemos en la Figura 57.

Figura 57 Visualización de calendario, perfil administrativo

5. Modificación del calendario

Con la pantalla que vemos en la Figura 58, los administrativos pueden cambiar el turno de trabajo de cualquier empleado de forma manual. El sistema permite cualquier cambio sin restricción por lo que será responsabilidad de los administrativos que el cambio sea viable.

Al activar el desplegable del dni se muestra automáticamente el turno actual.

Figura 58 Modificación de turno de forma manual

6. Gestionar notificaciones

En la pantalla presentada por la Figura 59 el administrativo gestionará las peticiones de cambio que hacen los médicos. Existen tres tipos diferentes: Bajas, Vacaciones, intercambios.

Las bajas obviamente no necesitan ser autorizadas, pero si gestionadas pudiendo intentar sustituirlas con los empleados disponibles o utilizando uno de los empleados del sistema que no pertenece a ese centro sanitario.

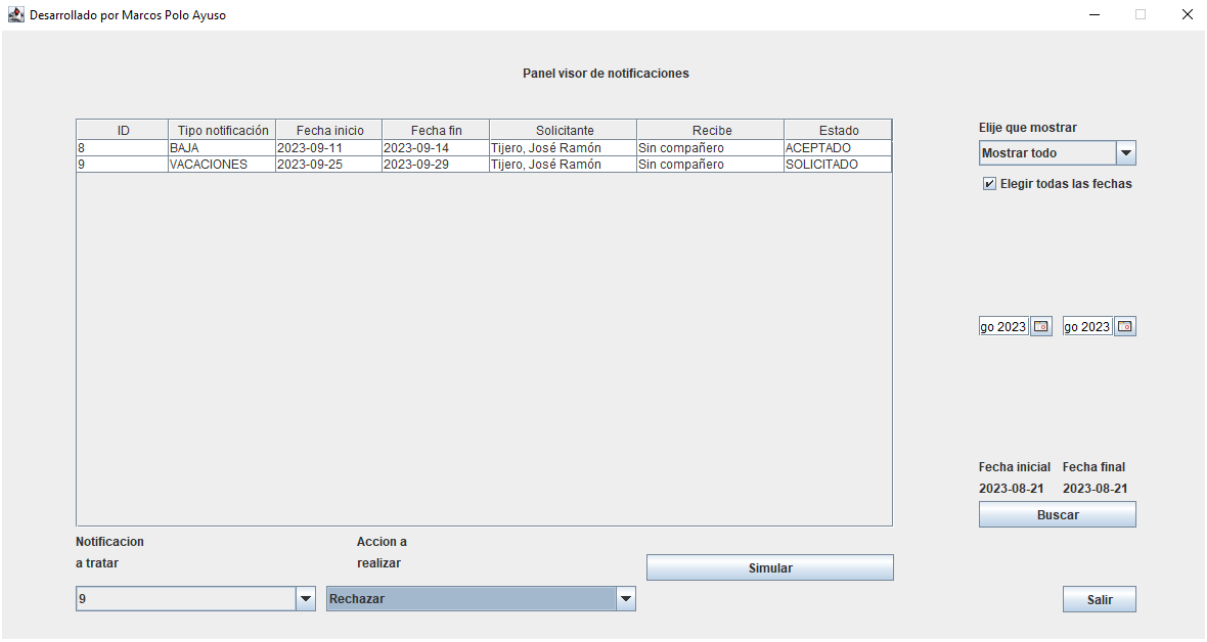

Figura 59 Pantalla de notificaciones, perfil administrativo

Perfíl médico

El médico, debido a su perfil, tiene menos operaciones disponibles que el administrativo como podemos ver en la Figura 60 la cual nos muestra las tres posibles opciones de las que dispone el médico: Calendario, solicitudes y notificaciones.

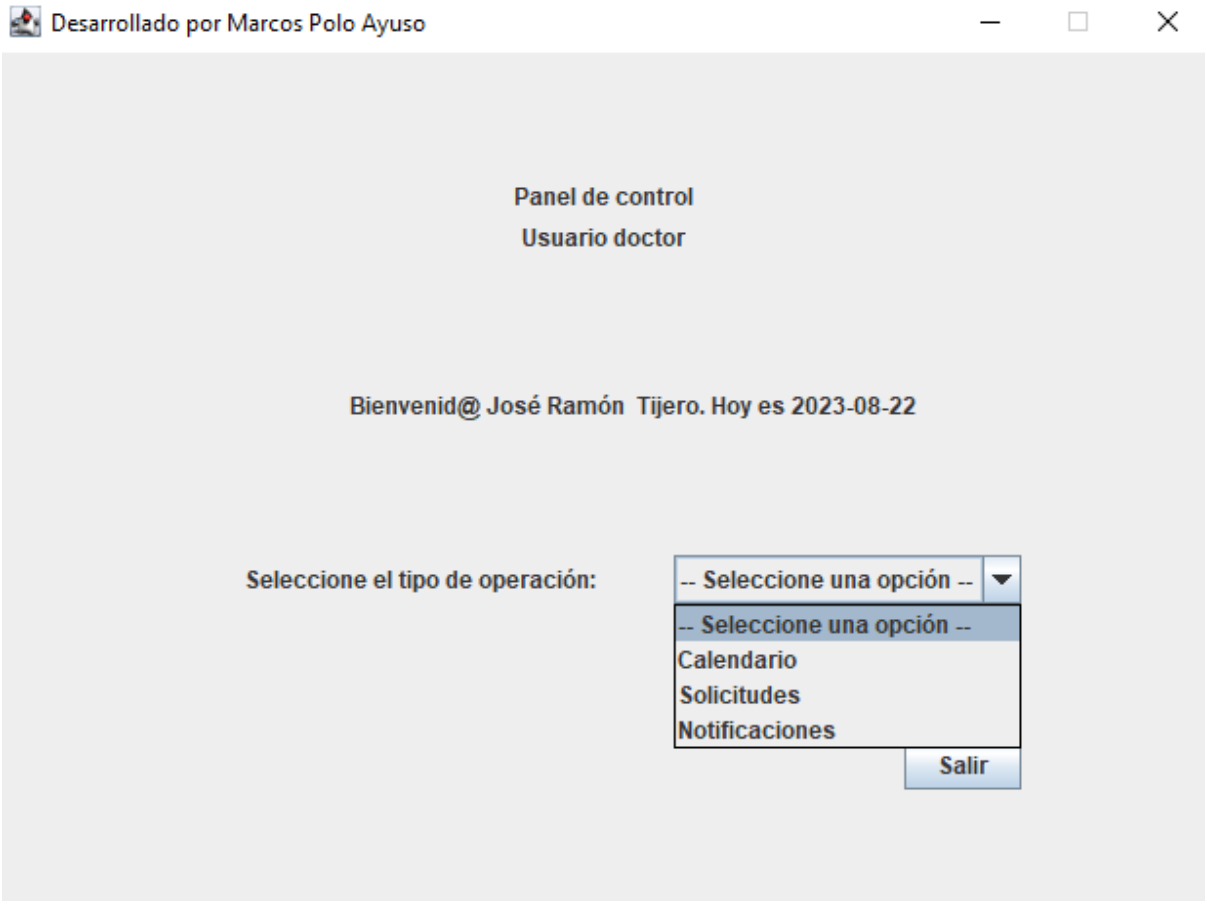

Figura 60 Pantalla de selección, perfil médico

1. Calendario

El médico puede consultar su calendario mensual en caso de que este exista. La pantalla de consulta es la mostrada en la Figura 61. Cualquier modificación que se haya confirmado aparecerá reflejada aquí.

Si seleccionamos cualquiera de los días en el calendario, el sistema nos sugerirá compañeros con los que contactar para cambiar ese día atendiendo a los siguientes criterios:

- Si es una mañana o una tarde

El sistema nos sugerirá un compañero que ese mismo día trabaje y tenga el turno contrario, si es una mañana sugerirá a compañeros que están de tarde y viceversa.

La idea es que contactemos con el compañero o compañera personalmente y concertemos el intercambio, posteriormente el interesado debería crear la la petición en el menú de solicitudes y esperar a que el personal administrativo lo autorice, si lo hace, el cambio se producirá en el calendario de ambos médicos.

- Si es una guardia

El sistema nos mostrará nuestro grupo antípoda para que podamos contactar personalmente con ellos e intercambiemos una guardia nuestra con otra que el pueda tener. Dado que el compañero es antípoda del solicitante, el cambio casi siempre será posible. No obstante, al igual que en el caso anterior, será necesaria la autorización y confirmación por parte del personal administrativo para que el intercambio sea efectivo.

Figura 61 Visualización de calendario del médico

2. Solicitudes

   La Figura 62 nos muestra la pantalla de solicitudes donde el médico interesado realizará las operaciones relacionadas con sus cambios de calendario. Las solicitudes que podrá realizar son las siguientes:

- ❖ Notificar una baja
- ❖ Solicitar vacaciones
- ❖ Solicitar un intercambio con un compañero

Si solicitamos un intercambio, se activará el desplegable con los dnis de todos los compañeros debiendo designar el correspondiente.

Figura 62 Pantalla de solicitudes, perfil médico

3. Notificaciones

Finalmente, en la Figura 63, podemos ver la pantalla en la que el médico podrá ver el estado en el que se encuentran sus operaciones. En caso de que hayan sido aceptadas y provoquen un cambio en el calendario este ya se habrá realizado automática e independientemente de que se haya consultado o no esta pantalla.

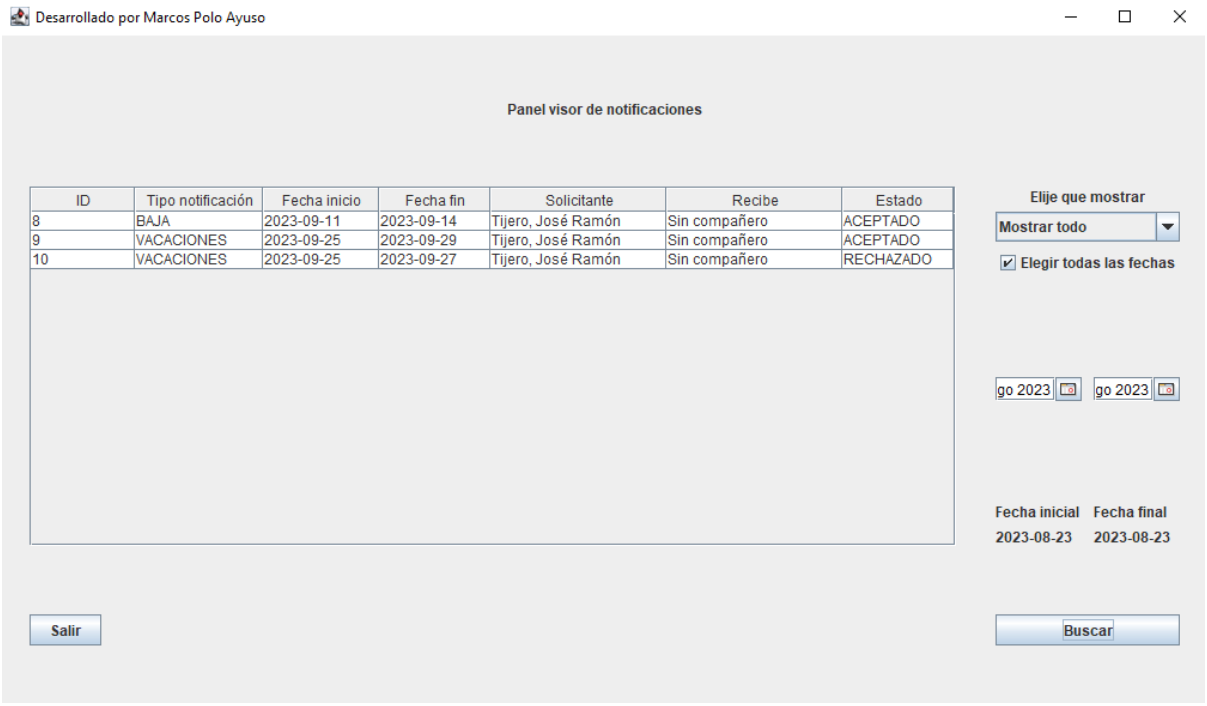

Figura 63 Visor de notificaciones, perfil médico