Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

PLAMIR : Un sistema para dar soporte a la evaluación de los médicos internos residentes

Estudante: Marcos Graña Domínguez
Dirección: Dr. Marcos Gestal Pose

A Coruña, setembro de 2021.

A mi familia y amigos

Agradecimientos

En primer lugar, quiero agradecer a mi familia, por su apoyo estos años y por los valores que me transmitieron desde siempre. A mis amigos, por hacerme disfrutar y desconectar durante unas horas todas las semanas . A mis compañeros de promoción, que gracias a su ayuda y motivación estos cuatro años se han hecho más fáciles. Y finalmente, a mi tutor de TFG, por implicarse y proponer la elaboración de este proyecto, y contestar siempre de forma rápida y eficaz a todas las preguntas que iban surgiendo.

Resumen

Las nuevas tecnologías están permitiendo continuamente adaptar procesos que antes se hacían de forma manual a formas más óptimas y eficaces de operar, en ámbitos inimaginables en los que pudiera entrar la informática, como el sector sanitario.

La aplicación web desarrollada permite que los residentes de un hospital, futuros especialistas en prácticas realizando el MIR, puedan tener un soporte para su evaluación por parte de sus tutores, proceso supervisado por la comisión académica del hospital, evaluación que constará de la subida de archivos llamados evidencias. Estos archivos pueden ser de varios tipos, como texto, imágenes, documentos en pdf, presentaciones, archivos ofimáticos, vídeos, etc.

La metodología empleada fue una metodología ágil basada en iteraciones. A medida que iban avanzando las iteraciones, se desarrollaban las distintas versiones hasta dar con la versión funcional final.

Abstract

New technologies are continually allowing to adapt processes that were previously done manually to more optimal and efficient ways of operating, in unimaginable areas where information technology could enter, such as the health sector.

The web application developed allows hospital residents, future internship specialists doing the MIR, to have support for their evaluation by their tutors, a process supervised by the academic committee of the hospital.

For the implementation of the system, the ASP.NET environment has been used, with the C# language.

The methodology used was an agile methodology based on iterations. As the iterations progressed, the different versions were developed until the final functional version was found.

Palabras clave:

- Desarrollo Web
- .NET
- Aplicación Web
- Arquitectura por capas
- Sector sanitario
- Residente
- MIR
- Archivos
- Base de Datos
- HTML

Keywords:

- Web development
- .NET
- Web Application
- Layered architecture
- Health sector
- Hospital resident
- Files
- Database
- HTML

Índice general

Índice de figuras

Capítulo 1

Introdución

El trabajo consiste en el diseño e implementación de una aplicación web de soporte a la tutorización de los residentes de un hospital por parte de sus tutores y supervisado por la comisión académica.

Los distintos roles con los que pueden entrar los distintos usuarios son: residente, tutor y comisión académica. Dependiendo del rol asociado al usuario tendrá acceso a sus funcionalidades propias dentro del sistema.

Las funcionalidades básicas de todo usuario son las de iniciar sesión, la cual quedará guardada en la sesión del navegador si el usuario así lo desea, cerrar sesión, modificar sus datos de perfil, enviar y leer mensajes a otros usuarios y consultar las notificaciones que el sistema le ha generado automáticamente. El rol residente puede consultar las rotaciones en las que está presente y subir evidencias que demuestren el conocimiento propio de esa rotación, con uno o varios archivos asociados. También podrá ver los tutores asociados en todo momento y enviarles mensajes si lo necesita, así como leer sus respuestas. El rol tutor puede ver sus residentes asociados, y ver las evidencias que han subido de sus rotaciones, marcando o no como relevantes esas evidencias si así lo desesa. También podrá comunicarse con sus residentes para la modificación de estas evidencias, e incluso puede añadir una evidencia a un residente, a modo de tarea, por lo que el residente deberá subir los archivos correspondientes. La comisión académica podrá asociar a los residentes con sus tutores específicos de rotación, y también ver el trabajo hecho por los residentes, supervisando la acción de los tutores y de los residentes al mismo tiempo.

La aplicación web a su misma vez está dividida en tres proyectos diferentes: el Model, o también llamado Backend , formado por una capa de acceso a datos y unos servicios que hacen uso de ella para implementar la lógica de negocio; la Web o también llamado Frontend, utilizando Web Forms de ASP.NET y C# como lenguaje de acceso a los servicios y un tercer proyecto llamado Test, que implementa las pruebas de integración del Model.

La metodología empleada siguió la filosofía de las metodologías ágiles, siendo un desa-

rrollo iterativo e incremental, en la que partiendo de una base muy sencilla y a partir de una serie de iteraciones se iban añadiendo una serie de funcionalidades que le iban dando forma y utilidad a la aplicación, hasta su versión final.

1.1 Motivación

En la actualidad es innegable que las nuevas tecnologías, en las que se incluye de manera importante la ingeniería informática, aportan facilidades a nuestra vida diaria, facilitando o automatizando tareas manuales, mejorando la comunicación, eficiencia de procesos, etc. Sus aplicaciones no tienen porqué necesariamente sustituir los procesos ya existentes. No es labor de la ingeniería informática decirle a profesionales de otro sector cómo hacer los procedimientos que ya han demostrado su eficacia, si no de facilitar, optimizar o/y automatizar estos procedimientos.

Las ventajas de adaptar procesos que anteriormente se desarrollaban de forma manual a las facilidades del mundo informático son inmensas. Podríamos destacar la facilidad de gestión, evitando tener que manejar grandes cantidades de documentos físicos con sus inconvenientes, la automatización evitándonos realizar tareas manuales repetitivas evitando errores, reflejar la información de manera estandarizada de acuerdo a un formato preestablecido, escalabilidad del sistema, permitiendo ampliarlo y modificarlo cuando y cómo sea necesario, etc. Pero la ventaja más deseable que me gustaría destacar es que permite a los profesionales de cualquier sector centrarse en lo verdaderamente importante, optimizando un recurso tan importante como el tiempo, y haciendo el trabajo más realizante evitando las tareas repetitivas, lo que conlleva a aportar más valor a la sociedad en menos tiempo.

El sector del cuál se ha intentado mejorar un proceso ha sido el sector sanitario. Un sector importantísimo del que depende la salud de muchas personas, permitiéndole a todos vivir con plenas capacidades en la medida de lo posible. En este sector, es importantísimo la figura del médico, ya que toma decisiones diariamente que afectan a la salud de sus pacientes, no siempre con las supervisación continua de su labor de más personas como podría pasar en otro sector. Por este motivo es importantísima su formación, en la que destaca la etapa del MIR, en las que se especializan en un área concreta, pasando una serie de rotaciones anuales. La automatización del proceso de validación de conocimientos por parte del tutor a un residente ha sido el que se ha intentado informatizar, procurando evitar la pérdida de tiempo en tareas repetitivas y permitiendo que empleen su tiempo en lo verdaderamente importante, formarse bien en una especialidad.

La elección de este tipo de proyecto, aplicación web, se basó en la posibilidad de afianzar los conocimientos de varias asignaturas cursadas en el grado, ya que este tipo de proyectos abarcan bastantes campos del conocimiento de la ingeniería informática, de los que podríamos

destacar el diseño e implementación de bases de datos (asignaturas Bases de Datos y Bases de Datos Avanzadas), desarrrollo de interafaces (Interfaces Persona Máquina), diseño de software orientado a objetos (Diseño de Software), arquitectura en capas (Arquitectura Software), las asignaturas propias del desarrollo web (Internet y Sistemas Distribuidos y Programación Avanzada), y en especial el desarrollo web en el entorno .Net (Marcos de Desarrollo).

La elección de metodología ágil en vez de metodologías más tradicionales, ha sido principalemente por dos motivos: la clara tendencia acutal de desarrollo software hacia metodologías ágiles permitiendo que en la inserción laboral sea un poco más familiar este tipo de metodologías, de las que prodríamos destacar SCRUM y que el tipo de proyecto, en el que los requisitos podrían ir cambiando y podría haber errores que requieran volver a etapas del desarrollo anteriores, se adaptaba mejor a este tipo de metodologías más flexibles.

1.2 Colaboración

En las primeras etapas del desarrollo, en el levantamiento de requisitos y en el desarrollo del modelo de datos a usar, se hizo de forma conjunta con otros proyectos, para dar el soporte debido a cada uno. El desarrollo conjunto acabó cuando después de establecer un modelo de datos en los que estuviéramos de acuerdo, se repartieron las funcionalidades a desarrollar y los tipos de plataforma en los que iba a funcionar el sistema, momento desde el cual el desarrollo se bifurcó y se empezó a trabajar en paralelo.

En un futuro se podría juntar los distintos sistemas resultantes y elaborar un sólo sistema que ofrezca todas las funcionalidades desarrolladas de manera conjunta en un buen número de tipos de dispositivos posibles.

1.3 Objetivos

Los objetivos perseguidos durante la realización del proyecto fueron los siguientes:

- Repasar los conocimientos adquiridos en las asignaturas ya mencionadas, poniendo en práctica e intentando reflejar esos conocimientos en un sistema útil y funcional.

- Entender un sector con el cual no estaba familiarizado como el sector sanitario, entendiendo y asimilando la jerga y los procedimientos que utilizan, aprendiendo a manejarse en entornos desconocidos.

- Planificarse y seguir una metodología de desarrollo ágil, para familiarizarse con la forma de actuar de la industria del software actual.

1.4 Organización de la memoria

A continuación se enumeran los capítulos que aparecerán en el presente documento, acompañados de una breve descripción de cada uno.

Estado de la cuestión En el capítulo se analiza la documentación aportada por el hospital y se extraen ideas y conceptos clave para el posterior desarrollo del proyecto, expuestos en la conclusión.

Fundamentos Tecnológicos Se expone y se dan motivos de las tecnologías que se han utilizado durante el proyecto para la realización del mismo, haciendo énfasis en las características que se han usado.

Metodología Se comentan la teoría de las metodologías en las que se ha inspirado la metodología final usada en el proyecto, también explicada y justificada.

Estimación Se calcula el tiempo y costo estimados inicialmente antes del desarrollo del proyecto, atendiendo a la planificación expuesta en este capítulo.

Fases de desarrollo del proyecto Se explica cómo se ha elaborado el proyecto, con el formato de la lista de sprints de Scrum desarrollados, en los cuales se justifican y explican los detalles que se consideran interesantes.

Pruebas Se describe de que tipo son las pruebas que validan el software creado y cómo se han realizado.

Funcionalidades destacadas Se enumeran las funcionalidades que desde el punto de vista del usuario puedan resultar importantes, destacando los detalles que resulten de interés resaltar.

Conclusiones El trabajo se concluye comprobando cómo se han realizado los objetivos del trabajo académico, enumerando y justificando lo que se ha aprendido y lo que puede mejorar el sistema en un futuro si se siguiera desarrollando el sistema.

Capítulo 2

Estado de la cuestión

PARA entender mejor el sector y el objetivo de este proyecto, el hospital ha facilitado documentación en la que se explicaba de manera detallada el procedimiento de evaluación de los residentes internos. A continuación se expondrá lo extraído de los documentos, que se utilizó posteriormente para la implementación de la funcionalidad.

2.1 Documentación

Es importante antes de acometer un proyecto, comprender las características propias de un sector, en este caso el sanitario. Esto permite no sólo comprender los procesos que se van a automatizar, sino aprender también la jerga propia del sector. Esto último es importante sobre todo para etapas tempranas del desarrollo, en la toma de requisitos, donde es necesario reunirse con el cliente que demanda el sistema, para que la comunicación fluya de manera óptima entre las dos partes.

2.1.1 Itinerario Formativo de la Unidad de Alergología del CHUAC

El documento presentado nos sirve como ejemplo de una especialización que puede escoger para su formación un médico interno residente, como Alergología. Del documento extraemos que una especialización consta de varios años como residente, con varias rotaciones cada año. Cada año de residencia, hay que cumplir con los objetivos del período formativo, realizando una serie de rotaciones, que consiste en rotar en varios servicios (como podría ser Medicina Interna, Neumología, Dermatología, etc.) cumpliendo los objetivos marcados de cada rotación.

2.1.2 Roadmap

El documento presentado como Roadmap, ayuda a comprender el proceso secuencial de la evaluación de los residentes. La secuencia empieza con la comisión académica asignando

los tutores específicos de rotación a un residente, después el tutor se encarga de subir las evidencias que necesitan ser realizadas por el residente mediante la subida de archivos. En este punto el residente podrá opcionalmente proponer sus propias evidencias aparte de rellenar las subidas ya por su tutor. Después, el tutor deberá seleccionar las evidencias más relevantes que demuestren el conocimiento adquirido por el residente, facilitando así su posterior evaluación.

2.1.3 Mockups

También se ha presentado una serie de mockups, también llamado maquetas de interfaz, en la que se nos deja claro los roles diferenciados que deberá constar la aplicación: residente, tutor y comisión académica. También se pone el objetivo de que la aplicación web sea sencillo de adoptar y de incluir nuevas funcionalidades a posteriori.

El documento consta de tres pantallas a modo de maqueta de interfaz, que muestran la funcionalidad deseada de cada rol: residente, tutor y comisión académica.

La pantalla del residente muestra las funcionalidades de ver el perfil del residente que haya iniciado sesión, después de introducir correctamente el nombre de usuario y contraseña y las notificaciones que le haya generado el sistema. También podemos apreciar que puede comunicarse tanto con sus tutores, y puede consultar una lista de las evidencias que vaya añadiendo, así como la información que puede aportar la comisión académica.

La pantalla del tutor también muestra la funcionalidad de ver su perfil y las notificaciones, además de poder enviar mensajes a cualquiera de sus residentes. También puede ver algunos datos de sus residentes y ver las evidencias que estos hayan subido, además de marcar como relevante las que considere oportunas.

Por otro lado, la pantalla de Comisión Académica muestra la funcionalidad de ver los residentes y los tutores. Aparte de asignar tutores a los residentes, la Comisión Académica

2.2 Conclusión

Tras analizar la documentación aportada, podemos destacar que es necesario que la aplicación web maneje diferentes roles, ya que es importante que la funcionalidad vaya acorde a lo que hace el usuario en el sistema. También se puede decidir que las actividades que es necesario realizar por parte del residente, pueden reflejarse en diferentes archivos, ya que es una forma de reflejar que se han completado las competencias demandadas en cada rotación, ya sea mediante redacciones por parte del residente, imágenes, publicaciones, presentaciones, etc. A estos archivos les llamaremos evidencias.

Por otro lado, que los mockups sean sencillos, y que se pida expresamente que sea fácil de adaptar, nos debe obligar a realizar el interfaz de manera que no se pierda tiempo en aprender

a manejar la aplicación, siendo necesario que esta sea lo más intuitiva posible y familiar para un usuario de ordenador promedio.

Capítulo 3

Fundamentos tecnológicos

En el capítulo expuesto a continuación, se expondrá lo que se ha utilizado para desarrollar el proyecto.

3.1 Hardware

Para el desarrollo del proyecto se ha contado con un portátil Lenovo Y50-70, con las siguientes características de hardware relevantes:

- Procesador: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
- Memoria RAM: 8,00 GB
- Sistema Operativo durante el desarrollo: Windows 10 Home de 64 bits
- Disco duro SSD de 512GB
- Conexión a internet vía ethernet

3.2 Entorno de desarrollo y librerías

3.2.1 Microsoft .NET

Es un entorno de desarrollo y ejecución creado por Microsoft que permite un rápido desarrollo de aplicaciones web con independencia del hardware utilizado. La empresa intenta una estrategia horizontal que rivalide contra la plataforma Java de Oracle u otros entornos de desarrollo web basados en PHP.

.Net permite ser independiente del lenguaje porque de los múltiples lenguajes disponibles (como C#, Visual Basic, Cobol, Python, Perl, etc.). Esto es posible gracias al CLR (Common Language Runtime), que carga y ejecuta el código fuente utilizando JIT (Just in Time) que

traduce IL a código máquina. El CLR también se encarga de la recolección de basura (objetos desreferenciados, etc.), mantener las distintas aplicaciones independientes unas de otras, servicios de depuración, etc. También permite desacoplar la aplicación del Sistema Operativo (no es obligatorio utilizar Windows, aunque sí recomendable)

Otra característica interesante es que es una plataforma Orientada a Objetos, que nos permite que el código sea reutilizable, organizado y fácil de mantener. Es más fácil seguir el principio fundamental DRY (don't repeat yourself), que enfatiza que no se repita código a lo largo del sistema permitiendo construir programas más eficientes y fáciles de mantener. También hace sencillo desarrollar un código encapsulado y modular, el cual nos va a ser más fácil de ampliar y las labores de mantenimiento.

3.2.2 Librerías de clases .NET

Las librerías de clases de la plataforma .NET abarca un conjunto de clases, interfaces, funcionalidades, etc. que vienen incluídas en .NET Framework. Incluye un conjunto de tipos básicos que son independientes del lenguaje de programación que se haya utilizado para implementar el código fuente. Estos tipos se organizan en los llamados namespaces, organizaciones lógicas jerárquicas en base a un nombre (ejemplo: System.Data). Estos tipos ya hablados están disponibles para todos los lenguajes compatibles con la plataforma, permitiendo que cualquier lenguaje no tenga ningún tipo de limitaciones.

Las librerías de clases están divididas en dos partes fundamentales: la llamada Base Class Library, que incluye un pequeño subconjunto de clases disponibles en todas las diferentes implementaciones de .NET Framework, y la Framework Class Library, un conjunto de clases más extensa que la anterior, que incluye librerías muy importantes para el desarrollo del proyecto como LINQ, ASP.NET, ADO.NET, etc. algunas de las cuales se expondrán a continuación.

El framework ADO.NET es un mapeador que convierte automáticamente filas de una base de datos relacional a objetos, emparejando con las conversiones de tipos necesarias los atributos de las columnas de la base de datos con las propiedades de los objetos. Esto permite al programador despreocuparse y abstraerse de las tablas y columnas de la base de datos donde se van a almacenar la información necesaria para el sistema, resultando un código con menos longitud y menos acoplamiento con la fuente de datos. Un aspecto relevante a destacar es que hay dos formas de acceder a la base de datos con esta librería: el entorno conectado, en la cual el sistema se mantiene en todo momento conectado a la base de datos mejorando el control de la concurrencia pero consumiendo más recursos, y el entorno desconectado, en la que el sistema accede a los datos realizando una copia de estos en local, óptimo si los datos únicamente se necesitan para lectura, con el inconveniente de poder llegar a pasar de tener los datos desactualizados.

No siempre es posible desacoplar completamente el código de acceso a datos de la propia

base de datos, siendo necesario realizar operaciones más complejas que las ofrecidas automáticamente. En este caso, el lenguaje LINQ permite desde C# proporcionar una interfaz de consulta muy potente, de manera muy similar a las consultas SQL tradicionales. Gracias a esto el tiempo de adaptación a LINQ sabiendo SQL es mínimo, y permite que el código sea más legible para la mayoría de los programadores, además de que permite que la consulta a distintas fuentes de datos sea bastante uniforme.

ASP.NET es una versión para la plataforma .NET basada en la tecnología ASP (Active Server Pages) que consta de tres partes muy diferenciadas: Formularios Web, los llamados Web Forms, los controles de servidor, tanto HTML como WebControls y los Servicios Web.

De los anteriores citados, destacamos por su uso en la aplicación los formularios Web Forms, que son páginas que mediante demanda del usuario, se combina HTML, scripts del lado del cliente y controles y código del servidor para dar resultado un marcado HTML que el navegador puede mostrar después de un compilado y ejecución del lado del servidor. Tiene integración con Visual Studio, que permite de forma sencilla arrastrar y colocar controles de servidor, siguiendo la filosofía WYSIWYG (What You See Is What You Get), y posteriormente definir de manera más personalizada tanto el comportamiento de la página y de los controles, además de la apariencia y funcionamiento. Como está basado en el protocolo HTTP, añade un control de eventos que hace más sencillo la implementación de comportamientos entre distintas páginas de la misma sesión, además de permitir la reusabilidad sencilla entre páginas similares, tanto dentro de un mismo sistema como de sistemas diferentes.

3.3 Lenguajes de programación y de marcado utilizados

3.3.1 C#

[1] C# es un lenguaje de programación desarrollado por Microsoft, orientado a objetos, que ha sido diseñado para compilar diversas aplicaciones que se ejecutan en .NET Framework. Se trata de un lenguaje simple, eficaz y con seguridad de tipos. Las numerosas innovaciones de C# permiten desarrollar aplicaciones rápidamente y mantener la expresividad y elegancia de los lenguajes de estilo de C.

La sintaxis viene derivada de C y C+ + y utiliza el modelo de objetos de la plataforma .NET, muy parecido al de Java, aunque incluye mejoras propias de otros lenguajes . Como curiosidad, el nombre de este lenguaje fue inspirado por la escala musical. En ella, la letra C equivale a la nota musical do y el símbolo # significa sostenido, lo que indica que es un semitono más alta. Así, C# sugiere que es superior a C y C++.

3.3.2 SQL

El lenguaje de consulta SQL (de sus siglas en inglés, Structured Query Language), permite consultar, manipular y crear bases de datos relacionales estrucuturadamente. Es casi un estándar en la industria del software por ser el lenguaje más usado para bases de datos relacionales, que a su vez es el tipo más común de bases de datos.

Para consultar y manipular la base de datos del sistema, se ha utilizado la herramienta SQL Server Management Studio por su sencillez de uso, que es un entorno integrado para administrar cualquier infraestructura de SQL. Con ella se ha comprobado mediante consultas en etapas tempranas de desarrollo la correctitud de los datos que se iban introduciendo, cuando por ejemplo se ha desarrollado primero en el Frontend la funcionalidad de crear antes que la de ver alguna entidad. También se ha utilizado para manipular datos ya existentes buscando un determinado comportamiento en el sistema que se debería dar en función de esos datos.

3.3.3 HTML

[2] HTML (que significa Lenguaje de Marcas de Hipertexto, del inglés HyperText Markup Language) es el componente más básico de la Web. Define el significado y la estructura del contenido web. Además de HTML, generalmente se utilizan otras tecnologías para describir la apariencia/presentación de una página web (CSS) o la funcionalidad/comportamiento.

”Hipertexto” hace referencia a los enlaces que conectan páginas web entre sí, ya sea dentro de un único sitio web o entre sitios web. Los enlaces son un aspecto fundamental de la Web.

HTML utiliza ”marcas” para etiquetar texto, imágenes y otro contenido para mostrarlo en un navegador Web. Las marcas HTML incluyen ”elementos” especiales como <head>, <title>, <body>, <header>, y muchos otros.

Un elemento HTML se distingue de otro texto en un documento mediante ”etiquetas”, que consisten en el nombre del elemento rodeado por ”<” y ”>”. El nombre de un elemento dentro de una etiqueta no distingue entre mayúsculas y minúsculas. Es decir, se puede escribir en mayúsculas, minúsculas o una mezcla. Por ejemplo, la etiqueta <title> se puede escribir como <Title>, <TITLE> o de cualquier otra forma.

Una de las ventajas de HTML es que es un estándar que cualquier navegador que se precie debería interpretarlo de forma óptima y rápida, permitiendo que un mismo desarrollo pueda ser utilizado por cualquier tipo de dispositivo capaz de ejecutar un navegador.

3.3.4 CSS

Es un lenguaje utilizado para mejorar el aspecto gráfico de presentación de un documento generado a partir de un lenguaje de marcado como podría ser HTML, anteriormente desarrollado. Permite darle una mejor apariencia sin tener que duplicar código, si se utiliza de forma

correcta.

3.4 Herramientas de desarrollo

Las herramientas que facilitaron el desarrollo del sistema son los siguientes:

3.4.1 Visual Studio

[3] Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para Windows y macOS. Es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET MVC, Django, etc.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno compatible con la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y videoconsolas, entre otros.

Entre las ventajas del uso de esta herramienta podremos comentar la compilación integrada al lenguaje que queramos, en este caso C#. También cuenta con integración con otras herramientas extremadamente útiles como por ejemplo un sistema de control de versiones como pueda ser Git. Otras de las facilidades que aporta son: IntelliTest, que genera tests automáticos que pueden servir como base para la elaboración de los tests de integración o de unidad; depuración; completado de texto adaptado siempre al lenguaje, con atajos que nos permiten por ejemplo generar constructores con parámetros de manera rápida, generación de equals y hashcode; completado de estructuras como ifs, whiles, etc.

Una extensión de Visual Studio extremadamente útil es NuGet. Las referencias de un proyecto suelen ser motivos de problemas y de errores que consumen mucho tiempo al desarrollador, por culpa de sus dependencias entre las distintas versiones como principal motivo. Para esto NuGet ofrece facilidad al agregar, eliminar y actualizar referencias (a librerías u otras herramientas), encargándose la extensión de labores como cambiar archivos en los que sea necesario hacer la modificación que ha hecho el usuario desde el interfaz, sin que el desarrollador tenga que preocuparse de qué archivos debe editar.

3.4.2 Git

[4] Git es un sistema de control de versiones distribuido de código abierto y gratuito diseñado para manejar tanto proyectos pequeños a muy grandes, con velocidad y eficiencia.

Git es fácil de aprender y ocupa poco espacio con un rendimiento increíblemente rápido . Supera a otras herramientas SCM con características como ramificación local, áreas de preparación convenientes y múltiples flujos de trabajo.

Gracias a Git se puede mantener el proceso de desarrollo del sistema, comparando distintas versiones, restaurándolas o fusionándolas si es necesario. También es útil a modo de copia de seguridad remota, resolviendo el problema de que el equipo de desarrollo pueda potencialmente presentar un problema que causa la pérdida total o parcial del código.

3.4.3 GitFlow

[5]

El flujo de trabajo Gitflow es un flujo de trabajo de Git que favorece el desarrollo continuo de software y las prácticas de implementación de DevOps. Fue Vincent Driessen quien lo publicó por primera vez y quien lo popularizó. El flujo de trabajo Gitflow define un modelo de creación de ramas estricto diseñado con la publicación del proyecto como fundamento. Para el proyecto, la filosofía de GitFlow casa perfectamente con la metodología iterativa e incremental, dejando reflejadas los distintos incrementos. También permite paralelizar el desarrollo con las ramas de funcionalidad.

GitFlow consta de las siguientes ramas principales y de desarrollo: En lugar de una única rama main, este flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama main o principal almacena el historial de publicación oficial y la rama develop o de desarrollo sirve como rama de integración para las funciones. Asimismo, conviene etiquetar todas las confirmaciones de la rama main con un número de versión.

La rama develop contendrá el historial completo del proyecto, mientras que main contendrá una versión abreviada.

Ramas de función. Por cada funcionalidad o agrupación lógica de funcionalidades, de deberá crear una rama propia ramificándose de develop como rama primaria. Cuando una función está terminada, se vuelve a fusionar en develop. Las funciones no deben interactuar nunca directamente con main. Las ramas feature suelen crearse a partir de la última rama develop.

Cuando develop haya adquirido suficientes funciones para una publicación (o se acerque una fecha de publicación predeterminada), se debe bifurcar una rama release (o de publicación) a partir de develop. Al crear esta rama, se inicia el siguiente ciclo de publicación, por lo que no pueden añadirse nuevas funciones una vez pasado este punto (en esta rama solo deben producirse las soluciones de errores, la generación de documentación y otras tareas orientadas a la publicación). Cuando está lista para el lanzamiento, la rama release se fusiona en main y se etiqueta con un número de versión. Además, debería volver a fusionarse en develop, ya que esta podría haber progresado desde que se iniciara la publicación.

Utilizar una rama específica para preparar publicaciones hace posible perfeccionar la publicación actual mientras se sigue trabajando en las funciones para la siguiente publicación. Asimismo, crea fases de desarrollo bien definidas, ideales para metodologías ágiles.

También existen las llamadas ramas de mantenimiento, de corrección o de hotfix. Sirven

para reparar rápidamente las publicaciones de producción. Las ramas hotfix son muy similares a las ramas release y feature, salvo por el hecho de que se basan en la rama main y no en la develop. Esta es la única rama que debería bifurcarse directamente a partir de main. Cuando se haya terminado de aplicar la corrección, debería fusionarse en main y develop (o la rama release actual), y main debería etiquetarse con un número de versión actualizado.

Tener una línea de desarrollo específica para la corrección de errores permite abordar las incidencias sin interrumpir el resto del flujo de trabajo ni esperar al siguiente ciclo de publicación. Se puede concebir las ramas de mantenimiento como ramas release ad hoc que trabajan directamente con la rama main.

Capítulo 4

Metodología

Una metodología en el ámbito de la ingeniería del software es el conjunto ordenado y estructurado que contempla elementos como los planteamientos, filosofías, procesos, fases, reglas, técnicas, herramientas, etc. que asegura a los desarrolladores de software la realización en tiempo y forma, con la calidad deseada del sistema final.

4.1 Desarrollo incremental

Las metodologías más tradicionales, como el desarrollo en cascada presentan como desventaja principal que un proyecto software normalmente no sigue una secuencia lineal. Sobre todo, cuando el desarrollador no tiene años de experiencia a la espalda, es normal que se encuentren errores que obliguen a rehacer partes del desarrollo ya realizadas, con lo que una metodología tan rígida como en cascada no permite.

La metodología de desarrollo en espiral, compensan las limitaciones del diseño en cascada ya que permite pasar por las distintas fases del desarrollo varias veces, solventando errores y aplicando lo aprendido a medida que avanza el proyecto.

El desarrollo incremental sigue solucionando los problemas del desarrollo en cascada, cuya base es el avance a través de iteraciones con resultados funcionales. Estas iteraciones, empezando desde una inicial muy sencilla y sin apenas funcionalidad, van secuencialmente añadiendo funcionalidad y siguiendo como base para las siguientes iteraciones.

4.2 Metodologías Ágiles

Las metodologías ágiles es una filosofía (más información en el manifiesto ágil [6])que cambia la forma de trabajar y organizarse comparándose con las metodologías más tradicionales. El proyecto se divide en partes que se deben ir diseñando e implementando a lo largo de las semanas, intentando desarrollar un sistema en el que los requisitos puedan cambiar sobre

la marcha o puedan surgir nuevos problemas.

Las principales ventajas del desarrollo siguiendo una metodología ágil son:

- Mejorar la calidad del proyecto, minimizando los errores a medida que avanzan los distintos entregables, pudiendo enseñarle al cliente la funcionalidad que va adquiriendo el sistema
- Más satisfactorio desde el punto de vista del desarrollador, ver cómo el sistema va aumentando viendo las funcionalidades funcionando a medida que las implementa.
- Acorta los tiempos de desarrollo y permite reaccionar y tomar decisiones a medida que avanza el proyecto.

4.2.1 SCRUM

Scrum es una metodología ágil de desarrollo, útil para proyectos en los que los requisitos pueden ir cambiando y requieren flexibilidad y rapidez. Es sencillo de aplicar 4.1 pero requiere de una gran productividad, pues en vez de ser una metodología muy planificada, se basa en la adaptación continua a las circunstancias de la evolución del proyecto.

Como metodología, es basada en iteraciones y revisiones, es decir, es incremental. Algunos de las características por las que es interesante usar Scrum como metodología/filosofía de desarrollo para un proyecto podrían ser los siguientes:

- El equipo o desarrollador individual no requiere de un jefe propio, se puede autogestionar de manera adecuada.
- Se gestiona un Product Backlog, que es una lista de los requisitos iniciales que se van a llevar a cabo en el desarrollo, es decir, los requerimientos.
- Se pauta el desarrollo en sprints, que con longitud no definida, aunque generalmente puedan ir desde 15 días a un mes.

4.3 Metodología usada en el proyecto

El proyecto ha sido desarrollado siguiendo un ciclo de vida iterativo e incremental, siguiendo la filosofía general de Scrum, por todas las ventajas y características expuestas anteriormente.

El desarrollo se ha centrado en la funcionalidad, teniendo eso en mente en todas las fases del proyecto, siempre intentando ir mejorando iteración a iteración el sistema. Cada sprint (iteraciones tenemos dentro de un proyecto desarrollado bajo Scrum) tendrá una duración de

tres semanas, en las que se intenta cumplir con la funcionalidad/es de la lista de requerimientos seleccionada para ese sprint.

Para organizar la lista de requerimientos, utilizaremos el concepto de backlog de Scrum, que es una lista de trabajo ordenador por prioridades, de la que se irán sacando las funcionalidades por orden de importancia a medida que el desarrollador pueda llevarlas a cabo. Este orden de importancia se basa en lo necesario desde el punto de vista del cliente, teniendo en cuenta las funcionalidades que más se van a usar, o de las cuales su funcionamiento dependan otras funcionalidades críticas.

Para estimar de forma sencilla el esfuerzo de cada sprint y equilibrar la carga de trabajo a lo largo del proyecto se ha utilizado el concepto de Scrum puntos de historia, que representan un valor relativo que sólo tiene significado para este proyecto que nos da una idea del tamaño y esfuerzo que se necesita para cada tarea del backlog del sistema.

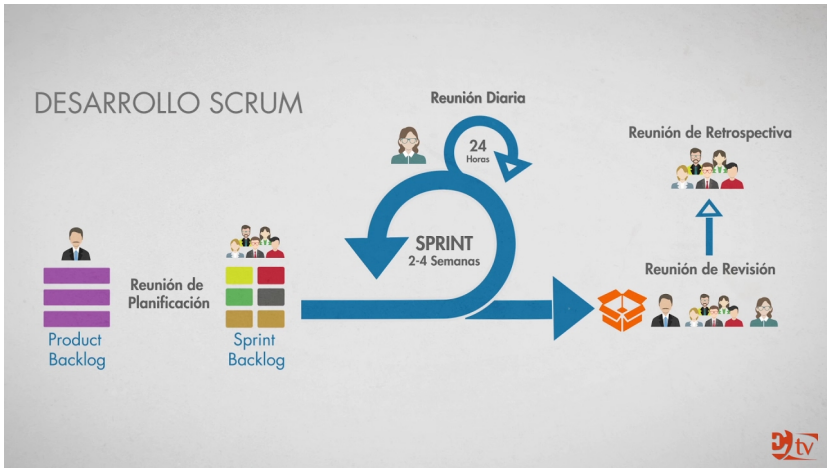

Figura 4.1: Diagrama sencillo del desarrollo con Scrum

Capítulo 5

Planificación y viabilidad económica

Para este capítulo, se calcula el tiempo aproximado de desarrollo del proyecto, estimando las horas de esfuerzo dedicadas y su planificación. Además del tiempo de desarrollo, se estimará el coste del proyecto total.

5.1 Importancia de la estimación

Antes de acometer un proyecto es fundamental para su planificación estimar en la medida de lo posible el tamaño de este. Hay varias formas de hacerlo, en este caso estimaremos el tamaño funcional en requerimentos. Esto es el paso previo para estimar el tiempo y el costo del proyecto, dos de los recursos más imortantes a valorar cuando acometemos un proyecto.

5.2 Estimación de tiempo

Siguiendo Scrum, se han planificado 7 sprints de 3 semanas cada uno. Para estimar el tamaño del sistema, se ha usado el concepto puntos historia explicado en (sección 4.3, página 18).

Los puntos historia es un valor relativo a este proyecto, no se debería asignar una conversión fija a horas, por lo que establecemos como referencia el requerimento añadir archivo a una evidencia, y le damos un valor arbitrario de 8. Esto nos servirá como escala para definir los puntos de historia del resto de requerimentos, en función de la complejidad, dificultad e incertidumbre de estos. Según estas directrices, definimos los puntos historia de cada elemento de la lista de requerimentos (sección 5.2, página 22).

Como estimación, podríamos establecer como regla que cada punto historia equivale a una hora de trabajo, en los que se incluyen las labores de análisis, diseño, implementación y pruebas. Desde el punto de vista funcional puede parecer que sea mucho tiempo por requerimiento, pero no sólo hay que desarrollar el requerimento, sino documentarlo, integrarlo en

el resto del sistema y adaptar los recursos necesarios como podría ser añadir más soluciones al proyecto, conectar las distintas capas de la arquitectura, etc.

Por cada funcionalidad, se estima que un 10% del tiempo se dedica a análisis, un 25% a diseño, un 55% a implementación y un 10% a pruebas.

Por lo tanto, teniendo en cuenta toda la información anterior, la estimación resulta un total de 240 horas de desarrollo, de los cuales 24 horas corresponden a análisis, 60 horas a diseño, 132 a implementación y 24 a pruebas. Aparte del propio desarrollo funcional como tal, añadimos 40 horas de análisis de la documentación inicial, reuniones de toma de requisitos y resolución de dudas. También añadimos unas 80 horas para la redacción de la memoria, por lo que queda una estimación aproximada de 360 horas.

ID	Requerimento	Puntos historia
1	Como Usuario quiero iniciar sesión en el sistema	8
2	Como Usuario quiero cerrar sesión en el sistema	2
3	Como Usuario quiero ver la información de mi perfil	4
4	Como Usuario quiero ver mis notificaciones no leídas	5
5	Como Usuario quiero ver mis notificaciones leídas	5
6	Como Usuario quiero enviar un mensaje a otro usuario	5
7	Como Usuario quiero ver los mensajes que me envió otro usuario	8
8	Como Residente quiero ver mis evidencias de una rotación	5
9	Como Residente quiero ver quién es mi tutor general	3
10	Como Residente quiero ver quienes son mis tutores de rotación	5
11	Como Residente quiero ver el detalle de una evidencia	5
12	Como Residente quiero añadir una evidencia a una rotación	10
13	Como Residente quiero añadir un archivo a una evidencia	8
14	Como Tutor quiero ver mis residentes generales	5
15	Como Tutor quiero ver mis residentes específicos de rotación	10
16	Como Tutor quiero ver las evidencias de una rotación	5
17	Como Tutor quiero destacar/desmarcar una evidencia	2
18	Como Tutor quiero ver el detalle de una evidencia	5
19	Como Tutor quiero añadir una evidencia	8
20	Como Comisión quiero asignar un residente a una rotación	12

5.3 Planificación

5.3.1 Sprint 1

Diagrama de Gantt del sprint 1, figura 5.1

5.3.2 Sprint 2

Diagrama de Gantt del sprint 2, figura 5.2

5.3.3 Sprint 3

Diagrama de Gantt del sprint 3, figura 5.3

5.3.4 Sprint 4

Diagrama de Gantt del sprint 4, figura 5.4

5.3.5 Sprint 5

Diagrama de Gantt del sprint 5, figura 5.5

5.3.6 Sprint 6

Diagrama de Gantt del sprint 6, figura 5.6

5.3.7 Sprint 7

Diagrama de Gantt del sprint 7, figura 5.7

5.4 Estimación de coste total

Una vez estimado las horas totales dedicadas al proyecto, y teniendo en cuenta que el salario medio por hora en España de un desarrollador en .NET es similar a 15€/hora [7], podemos estimar que el costo del desarrollador va a ser 15*360 = 5 400€

A esto hay que sumarle el trabajo del tutor como jefe de proyecto, 33€/hora [8] , calculando que se le dedique una hora por requerimento, saldría a 20*33 = 660€

El costo del portátil es de unos 800€. Se podría estimar que su vida útil es de cuatro años con un uso medio de cuatro horas diarias de media, por lo que corresponden 800/(4x365x4)=0.137 céntimos de euro por hora de uso. El tiempo de desarrollo del trabajo (360 horas), resultarían 340*0.137= 49.32€ de costo prorrateando el tiempo de vida útil estimado del portátil y el tiempo que se asigna al proyecto.

Por lo tanto, sumando los dos recursos humanos y físicos nos da un total de costo del proyecto de 6 109.32€ .

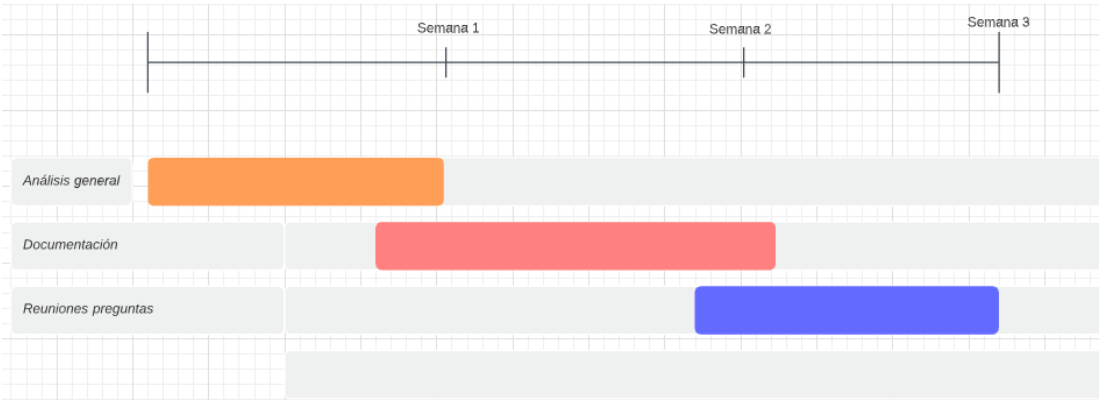

Figura 5.1: Diagrama de Gantt de planificación del Sprint 1

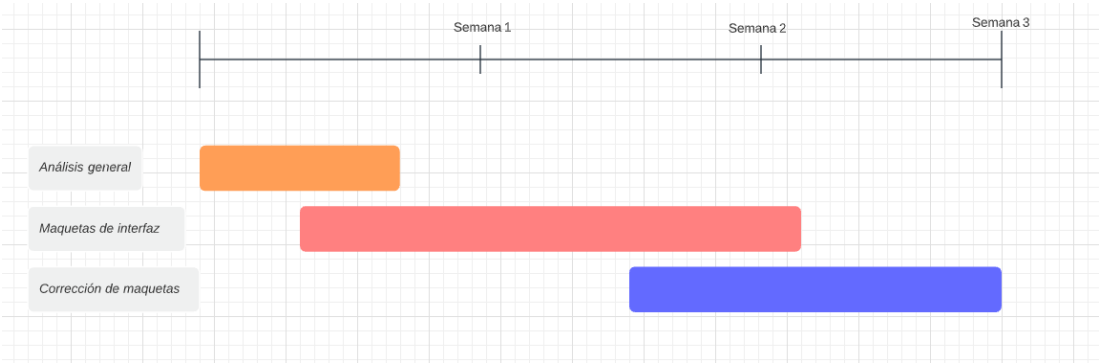

Figura 5.2: Diagrama de Gantt de planificación del Sprint 2

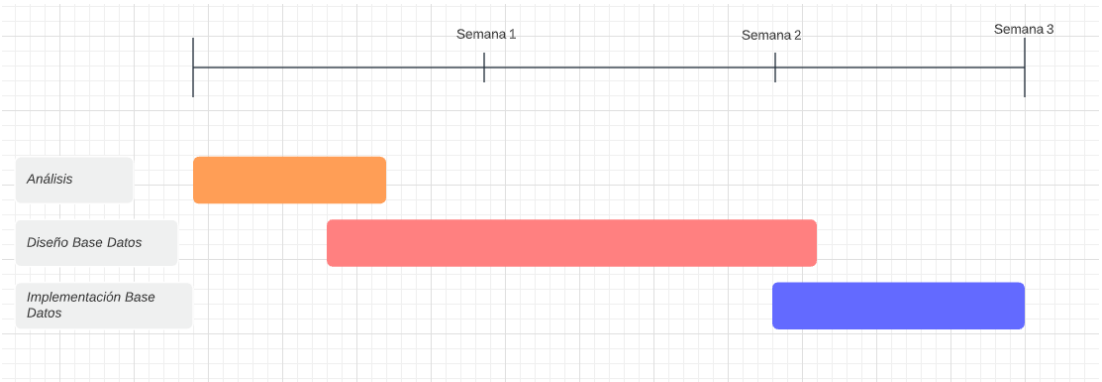

Figura 5.3: Diagrama de Gantt de planificación del Sprint 3

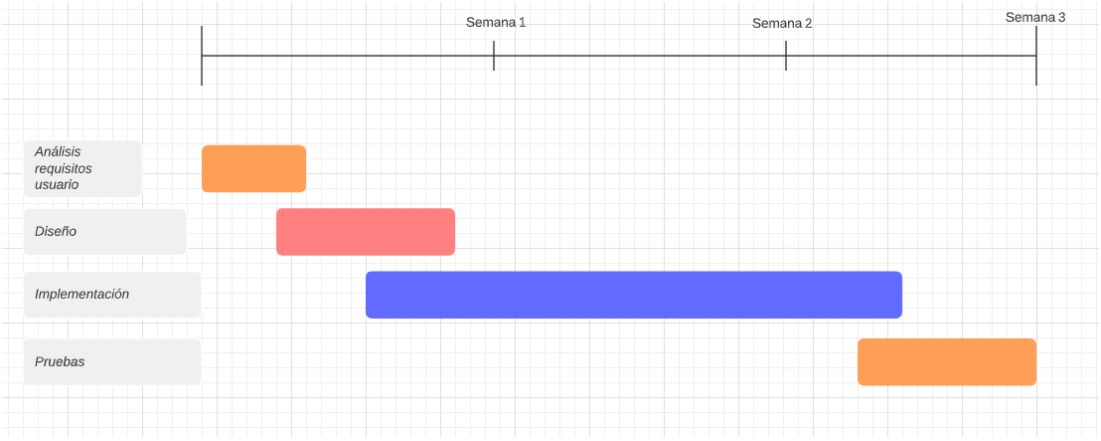

Figura 5.4: Diagrama de Gantt de planificación del Sprint 4

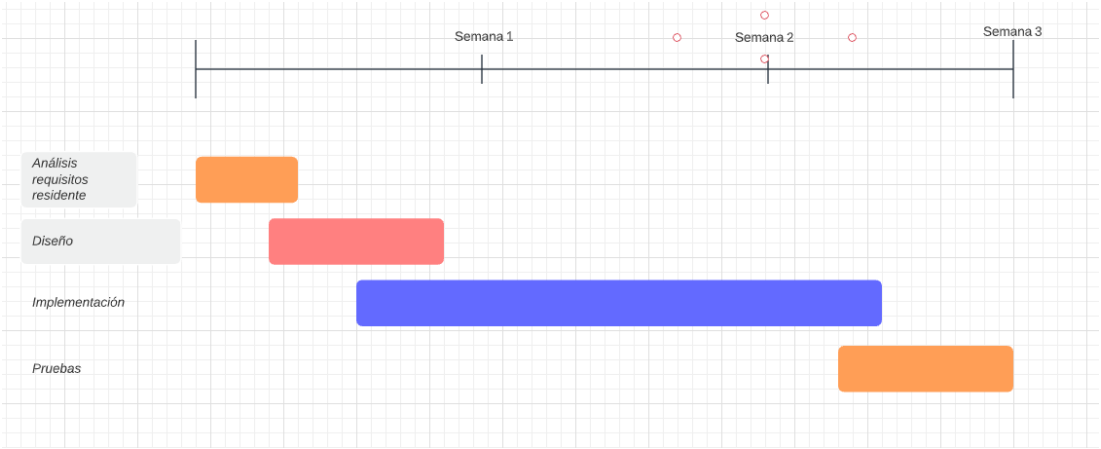

Figura 5.5: Diagrama de Gantt de planificación del Sprint 5

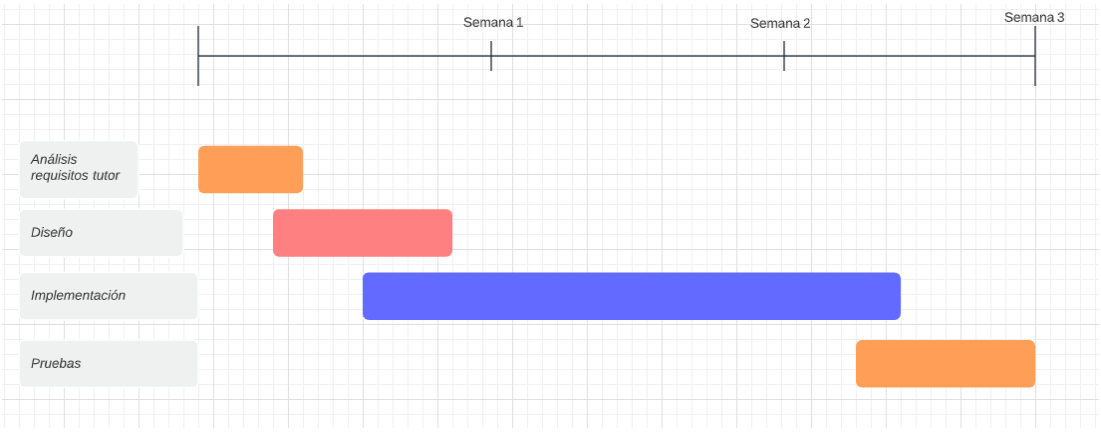

Figura 5.6: Diagrama de Gantt de planificación del Sprint 6

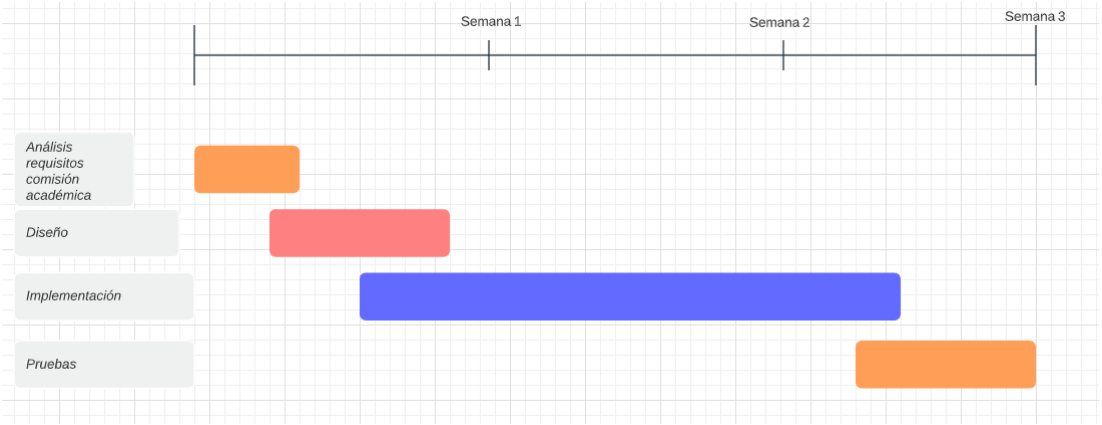

Figura 5.7: Diagrama de Gantt de planificación del Sprint 7

Capítulo 6

Fases de desarrollo del proyecto

En este capítulo se explica de manera más detallada los detalles de análisis, diseño e implementación del sistema. Para eso, se ha organizado el capítulo con los sprints de Scrum que se han desarrollado en orden cronológico.

6.1 Sprint 1

6.1.1 Actores y levantamiento de requisitos

En el primer sprint del proyecto se ha realizado el levantamiento de requisitos junto a la definición de actores. Para esto, fue fundamental la documentación aportada por el hospital, cuyas conclusiones se han explicado en (sección 2.1, página 5).

Después de leer a fondo los documentos, establecemos los actores del sistema, que fundamentalmente son tres: los residentes, los tutores y la comisión académica. Estos roles van a tener funcionalidades propias, que se explicarán a continuación. En este punto, fue una decisión importante añadir un actor genérico llamado usuario, pues hay bastantes funcionalidades comunes a residente, tutor y comisión académica. Para no implementar tres veces lo mismo, se ha usado herencia para que los tres actores hereden de usuario con sus características comunes, extendiendo la clase con sus funcionalidaes propias, como se ve en el diagrama 6.1, página 44)

Una vez tenemos claro los actores que participarán en el sistema y su rol, basándonos en la documentación, elaboramos una lista de requisitos que necesiten hacer cada rol. Para esto nos basamos en el objetivo con el que entra cada rol en el sistema. La lista de requisitos es la siguiente:

- Usuario: iniciar sesión en el sistema
- Usuario: quiero cerrar sesión en el sistema

- Usuario: quiero ver la información de mi perfil
- Usuario: quiero ver mis notificaciones no leídas
- Usuario: quiero ver mis notificaciones leídas
- Usuario: quiero enviar un mensaje a otro usuario
- Usuario: quiero ver los mensajes que me envió otro usuario
- Residente: quiero ver mis evidencias de una rotación
- Residente: quiero ver quién es mi tutor general
- Residente: quiero ver quienes son mis tutores de rotación
- Residente: quiero ver el detalle de una evidencia
- Residente: quiero añadir una evidencia a una rotación
- Residente: quiero añadir un archivo a una evidencia
- Tutor: quiero ver mis residentes generales
- Tutor: quiero ver mis residentes específicos de rotación
- Tutor: quiero ver las evidencias de una rotación
- Tutor: quiero destacar/desmarcar una evidencia
- Tutor: quiero ver el detalle de una evidencia
- Tutor: quiero añadir una evidencia
- Comisión: quiero asignar un residente a una rotación

Para la elaboración de la lista, aparte de la documentación, se han hecho varias sesiones de preguntas con el tutor para aclarar las dudas que iban surgiendo.

6.1.2 Diseño de la arquitectura del sistema

En la parte del diseño de este sprint, se ha pensado qué arquitectura sería la mejor para cubrir los requisitos ya mencionados. Una división que al principio es lógica, es dividir el sistema en interfaz de usuario (lo que se llama frontend) y la capa de lógica de negocio (lo que se llama backend), por lo que se opta por una arquitectura cliente-servidor 6.2, página 45). En este tipo de desarrollos, una de las arquitecturas que más ventajas aporta es la arquitectura por capas.

La arquitectura por capas (más información sobre este tipo de arquitectura en [9])consiste en que cada capa soporten a la capa jerárquicamente superior, aportándole servicios mediante contrato, sin que las capas superiores se preocupen de los detalles de implementación de las inferiores. Las ventajas de este tipo de arquitecturas son muchas entre las que destacamos:

- En caso de un desarrollo en equipo, permite el desarrollo por capas de manera paralela, optimizando los tiempos de desarrollo. En este caso, aunque sólo haya un desarrollador, permite que en una capa te abstraigas de las demás, ganando calidad general al ganar calidad cada una de las capas.

- Más facilidad de reutilización. Al tener cada capa una funcionalidad bien definida, puede utilizarse el mismo módulo para diferentes funcionalidades, por ejemplo, buscar evidencias por alumno, en el backend está definido cómo tiene que buscarlo en el modelo de datos, y el interfaz de usuario usa la misma llamada tanto para que un residente busque sus propias evidencias como para que un tutor busque las evidencias de un residente. Al estar en capas, permite esta reutilización tanto dentro del proyecto como de otros proyectos.

- Corrección de errores. Si en alguna capa se detecta un error, se puede cambiar sin temor a que se propague el cambio a otras capas, ya que son independientes.

- Como cada capa tiene un contrato de servicios que expone lo que debería hacer cada capa, con sus precondiciones y postcondiciones, la elaboración de pruebas, con la consiguiente detección de pruebas se hacen muy rápido y se encuentran rápido los defectos del software.

6.1.3 Implementación base del proyecto

En este sprint, lo más importante fueron las dos etapas anteriores, ya que ahora simplemente crearemos el proyecto y le añadiremos tres soluciones, en Visual Studio. Las tres soluciones serán el modelo, donde estarán los servicios (creamos las carpetas en las que irán estos) y el acceso a datos, así como el script de creación de base de datos; el proyecto web, donde se alojarán los recursos necesarios que le den soporte a las páginas que va a ver el usuario como interfaz, y por último el proyecto de pruebas, en donde se harán las pruebas de integración y unidad oportunas, como se explica en (sección 7, página 49).

También se añade el proyecto creado al repositorio Git utilizado, en este caso GitHub, y establecemos las ramas siguiendo la filosofía GitFlow, ya explicado en el capítulo (sección 6, página 27).

6.2 Sprint 2

6.2.1 Confirmación de requisitos mediante maquetas

Una forma de confirmar que el resultado del levantamiento de requisitos está bien orientado según quiere el cliente, es la creación de maquetas de interfaz, o en inglés, mockups.

Estos fotomontajes constan de un diseño parecido al que va a tener el interfaz de usuario del lado del cliente. Ayuda a que el cliente vea reflejado cómo el desarrollador entiende las funcionalidades que ha entendido en un formato familiar a un cliente no técnico. También ayuda a que el propio cliente realmente sepa lo que quiere.

Es importante minimizar todo lo posible el riesgo que un cliente cambie los requisitos cuando el desarrollo esté más avanzado, ya que volver atrás y redefinir lo que ya está hecho provoca pérdidas de tiempo importantes.

Estas maquetas no sólo deberán ser fotos estáticas, si no que deberían emular la secuencia de un actor hasta la funcionalidad deseada.

6.2.2 Elaboración esqueleto del interfaz de usuario

Para la elaboración de estos mockups, se ha propuesto implementar ya el interfaz como se va a ver en la versión final, en formularios ASP.NET Web Forms. Esto se debe a que el cliente va a ver desde un principio de manera clara qué va a poder hacer el sistema en el formato final.

Otro motivo por el que se ha decidido esto, es por ahorrar tiempo elaborando un mockup y posteriormente elaborar el interfaz, así ya se usa el interfaz con doble propósito.

Después de enseñar al cliente una demostración del interfaz, se ha tenido que cambiar aspectos a mejorar, correcciones que han llevado muy poco tiempo comparado al tiempo que llevaría corregir si se hubieran detectado más tarde.

Como pequeño aspecto negativo, se podría destacar que enseñar una maqueta tan parecida al diseño final en un navegador, puede provocar que un cliente no técnico tenga la sensación que el sistema está en fases finales del desarrollo cuando en realidad está en fases tempranas.

6.3 Sprint 3

6.3.1 Diseño Base de Datos

En este sprint definimos e implementamos la base de datos que el sistema va a usar para alojar los datos necesarios para dar soporte a la funcionalidad. Las tablas de la base de datos van a tener una relación uno a uno con las entidades que el servicio va a manejar.

Para implementar la persistencia (modelo de datos, acceso a datos y el mapeo objeto-relacional) con ADO.NET EF hay tres aproximaciones. Model First, que consiste en diseñar el modelo EDM en el proyecto, y a partir de ahí ADO.NET genera la base de datos y el código correspondiente a los objetos. Después, Code First, a partir del código de las entidades que deberemos escribir nosotros, genera la base de datos. La aproximación empleada finalmente fue DataBase First, por su nivel de personalización, que a partir del script sql de creación de las tablas en la base de datos, genera el modelo y el código.

Por lo tanto, antes de nada creamos con el SQL Management Studio dos bases de datos, la de producción, que usará el sistema cuando esté operativo, y la de test, en la que los test introducirán los datos necesarios para ejecutarlos.

Después de tener creadas las bases de datos, creamos el script sql, prestando mucha atención a las claves primarias, foráneas, que las entidades tengan los atributos necesarios, etc. Después, ejecutamos el scritp, previamente conectando a Visual Studio con el servidor local de Microsoft SQL Server. Una vez que comprobemos que las tablas se han creado correctamente, en la solución del modelo, añadimos un nuevo elemento llamado EDM (Entity Data Model). Lo que va a hacer es, siguiendo la filosofía DataBase First, generar el modelo de la base de datos y las entidades que van ir asociadas al código, como un mapeador objeto-relacional.

Una vez hecho esto, es necesario aclarar tres aspectos relevantes:

- Como ya hemos explicado, los tres roles de la aplicación heredan los atributos de usuario. El problema es que en el script sql se ha modelado como una relación uno a varios normal, por lo que tanto el modelo como el código no lo detectan como herencia. Como solución a esto, fue necesario con las herramientas proporcionadas por Visual Studio, eliminar la relación, sustituyendo la relación por la generalización, y eliminar el id generado a los tres roles, ya que será el de usuario. El diagrama quedaría de la siguiente forma (sección 6.3, página 46)

- Otro aspecto a destacar es que el código generado puede que no se ajuste a lo que queremos. La solución rápida que se nos podría ocurrir es modificar las clases generadas manualmente, pero no serviría de nada ya que al volver a compilar se borrarían los cambios. Para solucionar esto se ha recurrido a la modificación de las plantillas .tt, usando T4 Templates, que generan el código partiendo de la base de datos. En este caso, el cambio más relevante fue que sobrescribimos el equals por defecto, que compara por referencia, por un equals que compare por valor.

- La librería ADO.NET también establece la conexión con la base de datos, añadiendo la propiedad "ConnectionString" al archivo del Modelo App.config. Posteriormente, cuando se necesite inyectar la dependencia de la conexión con la base de datos, simplemente accediendo a esta propiedad puede conectarse de forma sencilla, como se muestra en

el fragmento de código a continuación, donde vemos que a partir de la propiedad ConnectionString llamada ”nombreConexión”, se inyecta en el kernel el contexto de la base de datos, siguiendo el patrón de instancia única.

```
///*** DbContext ***/
        string connectionString = ConfigurationManager.
        ConnectionStrings["nombreConexión"].ConnectionString;

        kernel.Bind<DbContext>().ToSelf().
        InSingletonScope().WithConstructorArgument(
        "nameOrConnectionString", connectionString);

```

Para gestionar la capa de acceso a datos, ya que tenemos implementada las entidades y la base de datos, emplearemos el patrón DAO. Este patrón se basa en el uso de llamadas que conforman una interfaz, que no revelan detalles de implementación de la capa inferior, la base de datos, respetando la arquitectura por capas. Esto permite que si cambiamos la base de datos, sólo es necesario cambiar los DAOs, ya que la capa servicios llaman a estos sin depender de la implementación concreta de la base de datos.

Las interfaces extienden de un DAO genérico, con las operaciones más comunes implementadas, el cual se ha seleccionado la implementación que se nos ha proporcionado para la asignatura Marcos de Desarrollo, que inlcuye las operaciones básicas de tipo CRUD (crear, buscar, actualizar y eliminar). Los DAOs que vamos creando, uno por entidad, extienden al DAO genérico. De esta forma, adquieren su funcionalidad, y además pueden añadir funcionalidades utilizando el lenguaje LINQ, similar a SQL. El diseño seguido para cada uno de los DAOs es el siguiente, en donde se muestra el modelo de ejemplo para la entidad Usuario (sección 6.4, página 47)

En el apartado de seguridad, se ha optado por encriptar la contraseña de usuario, pues si por cualquier fallo de seguridad algún atacante puede acceder a la base de datos, no podría saber cual es la contraseña en claro que tiene el usuario, ya que se encontraría con el hash. El algoritmo de encriptado utilizado fue SHA256 por su comprobada fiabilidad y rendimiento. Está definido en una clase, en el que se ofrece algunas operaciones necesarias. Esto se ha realizado así, porque en el resto del código simplemente se hacen llamadas a esta clase, sin conocer sus detalles de implementación. De esta forma, si en algún momento necesitamos cambiar fácil y rápido el algoritmo, simplemente habría que modificar una clase.

6.4 Sprint 4

En este sprint, se realizan las operaciones propias a usuario, que utilizarán posteriormente tanto los residentes, como los tutores, así como la comisión académica. Estas funcionalidades

se conectarán debidamente al interfaz, del que ya hemos definido su estructura a modo de maqueta en el sprint 2.

En el desarrollo de este sprint, se explicarán conceptos necesarios comunes a los sprints a partir de este, por lo que sólo se explicarán una vez.

Los requisitos que se seleccionan del Product Backlog para este sprint son los siguientes:

- Usuario: iniciar sesión en el sistema
- Usuario: quiero cerrar sesión en el sistema
- Usuario: quiero ver la información de mi perfil
- Usuario: quiero ver mis notificaciones no leídas
- Usuario: quiero ver mis notificaciones leídas
- Usuario: quiero enviar un mensaje a otro usuario
- Usuario: quiero ver los mensajes que me envió otro usuario

Primero, se implementará el backend correspondiente a los requisitos seleccionados para este sprint. Para esto, es necesario pensar, atendiendo a la interfaz propuesta en el sprint 2, qué operaciones debería exponer el backend, a lo que le llamamos servicio. Estos servicios se ofrecen al frontend en forma de interfaz, ocultando los detalles de implementación y sólo atendiendo a su funcionalidad, debidamente documentada, siguiendo el patrón fachada. En este sprint concreto, se ha implementado el interfaz llamado UsuarioService, que ofrece las funcionalidades de iniciar sesión (que devuelve los detalles del usuario si la contraseña y el login es correcto) y cerrar sesión. También se ha creado e implementado NotificacionService, que expone las operaciones de ver notificaciones leídas y no leídas de un usuario y el MensajeService, del cual se expone enviar mensaje a otro usuario y ver conversación con otro usuario.

Para conseguir las instancias tanto de los servicios como de los DAOs, emplearemos el patrón Inyección de Dependencias. El patrón se basa en que para conseguir un objeto, en ver de crearlo, se pide a otra clase específica que suministre una referencia. Para esto utilizamos el framework de inyección de dependencias Ninject, con el uso de la anotación [Inject].

Una llamada a un servicio puede que no nos devuelva un resultado esperado, ya sea por errores de permisos, de introducción de datos o error del sistema. Para que el usuario sepa correctamente lo que está pasando, y si ha sido error de el indicarle qué debería cambiar, se lanzan excepciones, que son objetos que devuelve una operación de un servicio ante un comportamiento diferente del normal. Estas excepciones se gestionarán de forma correcta en el frontend, ya que simplemente hay que prever los tipos de excepción y preparar una

respuesta diferente para cada uno de ellos. En este caso, a modo de ejemplo, mostramos la excepción IncorrectPasswordException.

```
public class IncorrectPasswordException : Exception
{
    /// <summary>
    /// Inicializa una nueva instancia de la clase
    /// <see cref="IncorrectPasswordException"/>.
    /// </summary>
    /// <param name="usuarioLogin"><c>clientLogin</c> that causes
    the error.</param>
    public IncorrectPasswordException(String usuarioLogin)
        : base("Incorrect password exception
            => usuarioLogin = " + usuarioLogin)
    {
        this.UsuarioLogin = usuarioLogin;
    }

    /// <summary>
    /// Almacena el login del usuario del cual
    ///  la contraseña introducida ha sido incorrecta
    /// </summary>
    /// <value>The name of the login.</value>
    public String UsuarioLogin { get; private set; }
}
```

Como vemos, las excepciones extienden de la clase ya existente Exception, la cual nos ofrece las utilidades del try catch en el frontend, que nos permite gestionar las excepciones de forma sencilla. En este caso, para aportar información al usuario, la excepción devuelve el usuario del cual ha dado error la contraseña introducida.

Una vez las operaciones de los servicios no hayan generado ninguna excepción, en el caso que se necesiten mandar datos al frontend, se enviarán siguiendo el patrón DTO (Data Transfer Object). Este patrón consiste en devolver los datos seleccionados en un objeto sólo con propiedades de lectura. De esta forma, creamos la información específica que necesite enviar el backend, sin enviar de más (peor optimización y rendimiento) o de menos (no conseguiríamos el objetivo de la operación). En este caso a modo de ejemplo, enseñamos la clase DetallesUsuarioDTO, devuelta cuando se requiere la información básica del perfil, por ejemplo.

```
   [Serializable()]
public class DetallesUsuarioDTO
{

    public long usuarioId { get; private set; }

```

```
7       public String Nombre { get; private set; }
8
9       public String PrimerApellido { get; private set; }
10
11      public String SegundoApellido { get; private set; }
12
13      public String Login { get; private set; }
14
15      public String Rol { get; private set; }
16
17      public String Contacto { get; private set; }
18
19      public String EncryptedPassword { get; private set; }
20
21      /// <summary>
22      /// Inicializa una instancia de la  <see
        cref="DetallesUsuarioDTO"/>
23      /// clase.
24      /// </summary>
25      /// <param name="usuarioId">El id del usuario.</param>
26      /// <param name="nombre">El nombre del usuario.</param>
27      /// <param name="primerApellido">El primer apellido del
        cliente.</param>
28      /// <param name="segundoApellido">El segundo apellido del
        cliente.</param>
29      /// <param name="login">El login del usuario.</param>
30      /// <param name="rol">El rol del usuario.</param>
31      /// <param name="contacto">El contacto del usuario.</param>
32      /// <param name="encryptedPassword">Indica si la contraseña
        está encriptada</param>
33      public DetallesUsuarioDTO(long usuarioId, String nombre, String
        primerApellido, String segundoApellido, String login, String
        rol, String contacto, String encryptedPassword)
34      {
35          this.usuarioId = usuarioId;
36          this.Nombre = nombre;
37          this.PrimerApellido = primerApellido;
38          this.SegundoApellido = segundoApellido;
39          this.Login = login;
40          this.Rol = rol;
41          this.Contacto = contacto;
42          this.EncryptedPassword = encryptedPassword;
43      }
44
45      public override bool Equals(object obj)
46      {
```

```
47          // Código que compara dos objetos de
48          //  esta clase por valor, y no
49          // por referencia, ocultado por reducir
50          // cantidad de código
51      }
52
53      public override int GetHashCode()
54      {
55          // Código del hashcode, ocultado por
56          //  no ser relevante para la
57          //  explicación actual
58      }
59  }
```

Como detalles a destacar, decir que la clase está anotada como [Serializable] para poder transportarla por una conexión de red de forma correcta. También observamos que las propiedades de lectura son públicos, mientras que los de escritura son privados. También destacamos que los DTO no sólo sirven para enviar datos desde en backend, sino también para recibirlos de forma estandarizada.

Desde el Frontend, para acceder a estos servicios, desarrollaremos la capa de acceso a servicios, que de forma conjunta con la interfaz conforman el frontend. La capa de acceso a servicios se ha implementado dentro de las carpetas Page que llamaban a estos servicios, con una clase específica, que tiene operaciones estáticas que se encargan de llamar a las operaciones del servicio.

La capa propia del interfaz, contiene un elemento llamado Master Page, que define la vista general de la web, es decir, los componentes comunes a todas las páginas (El encabezado, el menú, etc.). Después, el resto de páginas, llamadas Page, extienden a la Master Page, y añaden interfaz propia de cada página. Ejemplos de Page podrían ser, formulario de autenticación (sección 6.5, página 47), ver mi perfil, etc. En el ejemplo mostrado de iniciar sesión, vemos el recuadro rojo, parte común a toda la web implementada por la Master Page, y el resto el formulario de autenticación que implementa la página concreta IniciarSesion.aspx

Estos elementos Page, que son clases que extienden System.Web.UI.Page, contienen tres archivos que implementan el interfaz:

- Page.aspx : Esta página mediante código HTML y controles propios de ASP.NET definen la vista que recibe el navegador. Para su visualización, el servidor procesa todo el código y devuelve solamente código HTML.
- Page.aspx.cs : Este archivo contiene el código fuente que controla la vista y su comportamiento, lo que se denomina el Code Behind. Esto se hace así para separar la vista del controlador y facilitar el desarrollo y legibilidad.

- Page.aspx.designer.cs : Contiene código generado automáticamente, que se encarga de definir los controles propios de ASP.NET.

El code behind, el archivo Page.aspx.cs contiene una serie de propiedades y métodos en C# interesantes y fundamentales para el desarrollo que nombraremos a continuación:

- PageLoad() : Método donde definimos qué tiene que hacer el sistema cada vez que se carga una página en concreto
- AutoPostBack : Propiedad de un control que si está activada, provoca una llamada al PageLoad() cada vez que se modifica el estado de este.
- IsPostBack : nos indica en el PageLoad() si la carga de la página proviene de un PostBack, útil para saber si es la primera vez que se carga o no
- Session : de la clase HttpSessionState, almacena datos específicos entre la misma sesión, ya que el protocolo HTTP no guarda información ante peticiones consecutivas
- Response.Redirect() : Método que fuerza la carga de una página/recurso que queramos, útil para redireccionar otra página ante la petición de un usuario.
- Control.Visible : Atributo con el que marcamos que un control esté visible o no desde el Code Behind.

Siguiendo las directrices anteriores, se implementa en este sprint las funcionalidades propias de usuario. Estas funcionalidades hacen uso de las cookies, fragmentos de información guardados en el navegador, con el que se mantiene los detalles de la sesión iniciada por un usuario, hasta que cierre sesión, momento en el que se eliminan estas cookies.

Para ver una lista de datos de forma cómoda para el usuario, se ha usado el control llamado GridView, en el cual con la propiedad DataSource() le asignas una fuente de datos, generalmente una lista de DTOs, de la cual pone cada atributo de cada objeto de la lista en una columna diferente. Los nombres de las columnas pueden ser diferentes a los de los atributos, definiéndolo en el Page.aspx.

Como último detalle relevante en este sprint, en el desarrollo de ver mensajes que nos ha enviado un usuario, se ha considerado que el número de mensajes que nos puede enviar un usuario puede llegar a ser un número muy alto como para mostrarlos de una sóla vez en el GridView. Por lo tanto, se ha implementado con paginación, mediante el cual, con los atributos Count (número de mensajes a mostrar) y AreMoreMensajes (indica si hay más mensajes a parte de los ya mostrados) el usuario va pidiendo bloques de mensajes, haciendo mas eficiente la carga de mensajes de forma fragmentada, sin saturar el backend más de lo necesario. Además de paginación en ver mensajes, también se ha implementado en ver notificaciones de la misma forma.

6.5 Sprint 5

En este sprint se implementan las funcionalidades descritas a continuación para los Residentes del sistema. Se ha implementado siguiendo las directrices explicadas anteriormente, y sólo destacaremos los detalles relevantes que han sido diferentes en este sprint.

- Residente: quiero ver mis evidencias de una rotación
- Residente: quiero ver quién es mi tutor general
- Residente: quiero ver quienes son mis tutores de rotación
- Residente: quiero ver el detalle de una evidencia
- Residente: quiero añadir una evidencia a una rotación
- Residente: quiero añadir un archivo a una evidencia

Para ver las evidencias de una rotación pasándole el id de la Rotación desde el Frontend, se ha necesitado ampliar la funcionalidad del DAO genérico de evidencias, el cual será llamado por el servicio EvidenciaService. Esto es debido a que necesitamos recuperar las evidencias de las cuales estén relacionadas con la rotación pasada por parámetros. Además, también es necesario implementar la paginación a nivel de DAO

```
1  public List<Evidencia> FindByRotacion(long rotacionId,
2  int startIndex, int count)
3  {
4
5      DbSet<Evidencia> evidenciaList = Context.Set<Evidencia>();
6
7      List<Evidencia> evidenciaListReturn =
8          (from e in evidenciaList
9           where e.rotacionId == rotacionId
10          orderby e.fechaEntrega.Value descending
11          select e).Skip(startIndex).Take(count).ToList<Evidencia>();
12
13     return evidenciaListReturn;
14
15 }
```

En el fragmento de código podemos apreciar cómo se ha utilizado el lenguaje LINQ, con el que se ha hecho la consulta. Primero seleccionamos todas las evidencias en la línea 5, y posteriormente en el where se descartan todas las evidencias que no estén asociadas a la rotación de la cual pasamos al DAO por parámetros. También seleccionamos el bloque de

paginación que demanda el usuario cortando la lista desde el startIndex, hasta startIndex + Count.

Una operación del servicio relevante por su diferencia con las demás es el de añadir archivo a una evidencia. Para cargarlo, utilizamos el control propio de ASP FileUpload, el cual es un explorador de archivos en el que seleccionamos el archivo de nuestro equipo a añadir. Este genera un PostedFile que en el code behind del frontend lo enviamos al servicio. El servicio, por su parte, guarda los detalles del archivo en la tabla de la base de datos denominada ArchivoEvidencia, como vemos en el código a continuación.

```
// Nuevo objeto ArchivoEvidencia
ArchivoEvidencia archivoEvidencia =
new ArchivoEvidencia();

// Cargamos el objeto HttpPostedFile que nos manda
el usuario en el DTO
HttpPostedFile archivo = evidenciaDetalles.Archivo;

// Cargamos la ruta física de la carpeta
evidenciasArchivos
string rutaFisica = HttpContext.Current.Server.
MapPath("~/evidenciasArchivos");

// Si no existe la carpeta, la creamos
if (!Directory.Exists(rutaFisica))
{
    Directory.CreateDirectory(rutaFisica);
}

// Creamos el path del archivo que vamos añadir
string archivoPath = String.Format("{0}\\{1}",
rutaFisica, archivo.FileName);

// Verificar que el archivo no exista. Si no
// existe, lo guardamos con ese path
//    y guardamos el path en BD
if (File.Exists(archivoPath))
{
    throw new FileAlreadyExistsException(
    archivoPath);
}
else
{
    archivo.SaveAs(archivoPath);

}
```

```
37
38 archivoEvidencia.nombre = archivo.FileName;
39 archivoEvidencia.tipoArchivo =
40 archivo.ContentType;
41 archivoEvidencia.direccionFisica = archivoPath;
42 archivoEvidencia.evidenciaId = evidencia.id;
43
44 ArchivoEvidenciaDao.Create(archivoEvidencia);
```

Como podemos ver en el código, cargamos la ruta completa del servidor a partir de una ruta relativa con el método Server.MapPath. Después, guardamos el archivo en la ruta, y guardamos la ruta entera en la entidad archivo, además de otros detalles como el tipo de archivo o la clave foránea al id de la evidencia a la que pertenece el archivo. De esta forma, guardamos el archivo en el sistema de archivos del sistema operativo del sistema operativo del servidor, guardando la ruta del archivo, evitándonos guardar el contenido del archivo en la base de datos como un binario, solución mucho menos eficiente, ya que cargamos demasiado la base de datos con consultas que devuelven gran volumen de datos.

En ver el detalle de una evidencia, veremos los archivos que completan la evidencia y los podremos descargar a nuestro equipo para su cómoda visualización. La descarga del archivo se implementa seleccionando la ruta del atributo correspondiente de la tabla de la base de datos ArchivoEvidencia del cual queremos descargar su contenido. Después, desde el servicio construimos un objeto System.IO.FileInfo, que gracias al constructor de la clase nos devuelve una instancia a partir de la ruta de un archivo. El FileInfo construido, se lo pasamos al Frontend, desde el cual mediante la propiedad HttpContext.Response, podemos transmitir el archivo por el protocolo HTTP con la secuencia de código que se enseña a continuación:

```
1  FileInfo archivo = EvidenciaServiceConnector.DescargarArchivo(
2    idArchivoEvidencia);
3
4  Response.Clear();
5  Response.ClearHeaders();
6  Response.ClearContent();
7  Response.AddHeader("Content-Disposition",
8    "attachment; filename=" + archivo.Name);
9  Response.AddHeader("Content-Length",
10   archivo.Length.ToString());
11 Response.Flush();
12 Response.TransmitFile(archivo.FullName);
13 Response.End();
```

6.6 Sprint 6

En este sprint se implementan las funcionalidades descritas a continuación para los Tutores del sistema. Se ha implementado siguiendo las directrices explicadas anteriormente, y sólo destacaremos los detalles relevantes que han sido diferentes en este sprint.

- Tutor: quiero ver mis residentes generales
- Tutor: quiero ver mis residentes específicos de rotación
- Tutor: quiero ver las evidencias de una rotación
- Tutor: quiero destacar/desmarcar una evidencia
- Tutor: quiero ver el detalle de una evidencia
- Tutor: quiero añadir una evidencia

Para ver los residentes específicos de rotación, se necesita extender el DAO genérico y añadir una operación que pasándole el login de un usuario, devuelva una lista de residentes de los cuales tienen alguna rotación tutorizada por el login que se le ha pasado por parámetros.

```
public List<Residente> FindResidentes(string usuarioLogin)
{

    DbSet<Residente> usuarios = Context.Set<Residente>();
    DbSet<Rotacion> rotaciones = Context.Set<Rotacion>();

    var result =
        (from u in usuarios
         join r in (from r in rotaciones
                    where r.Tutor.usuarioLogin == usuarioLogin
                    select r) on u.usuarioId equals r.residenteId
         select u).ToList();

    List<Residente> usuario = result;

    if (usuario == null)
        throw new InstanceNotFoundException(usuarioLogin,
            typeof(Usuario).FullName);

    return usuario;
}
```

En la operación, primero seleccionamos todos los residentes y rotaciones. Después, utilizando el lenguaje LINQ, con una subconsulta correlacionada similar a cómo lo haríamos en

SQL, seleccionamos los usuarios cuyo login del tutor de alguna rotación coincida con el del login del tutor pasado por parámetros. Si no hay ninguno, devuelve la excepción InstanceNotFoundException, con la información del login del usuario.

Para aportar más seguridad al sistema y evitar errores innecesarios, se ha añadido una comprobación del rol de un usuario antes de permitir que llame a alguna operación no definida para su rol. Siguiendo con ejemplo de ver los residentes específicos de rotación, mostramos cómo se hace esa capa de seguridad en el servicio.

```
1  [Transactional]
2  public List<Residente> ObtenerResidentesRotación(string
       usuarioLogin)
3  {
4
5      Usuario usuario = UsuarioDao.FindByLogin(usuarioLogin);
6      if (usuario is Tutor)
7      {
8          List<Residente>  residentes =
9              UsuarioDao.FindResidentes(usuarioLogin);
10
11         return residentes;
12     }
13     throw new NotAllowedException(usuario.usuarioLogin);
14 }
```

Como vemos en el código, en la línea 6 de la operación del servicio UsuarioService, se comprueba que el usuario que quiere buscar sus residentes específicos de rotación sea de tipo Tutor. Esto se puede hacer por polimorfismo, ya que la clase Tutor extiende a Usuario, por lo que un objeto de la clase Tutor es un Usuario. Por lo tanto, si un tutor llama a la operación, ejecuta la llamada al DAO, si lo llama un rol distinto, lanza la excepción NotAllowedException, lo cual indica que esa operación no está permitida para el usuario.

Por último, en este sprint, cabe destacar cómo la acción de un usuario puede provocar la generación de notificaciones a otro usuario de forma automática, sin pedirlo directamente, mostrando un fragmento de código dentro de la operación Añadir Evidencia, en el servicio EvidenciaService.

```
1  // Si el que añade la evidencia es el tutor, notificarselo al
       residente
2  if (EsTutor)
3      {
4          NotificacionService.CrearNotificacion("Nueva evidencia
       añadida por el tutor",
5          "El tutor de tu rotación " + evidencia.Rotacion.contenido +
       " ha añadido una nueva evidencia a rellenar",
```

```
6         evidencia.Rotacion.Tutor.usuarioId, evidencia.relevante,
    evidencia.Rotacion.residenteId, evidencia.Rotacion.contenido);
7     }
```

El fragmento de código anterior está justo después de añadir la evidencia. Esto se hace de esta forma porque si se hace antes, se genera la notificación, y puede ser que la evidencia no se añada por algún motivo, y la notificación no tenga significado. La llamada al servicio, desde el frontend, indica la propiedad EsTutor, ya que en el frontend tenemos guardado en la sesión el rol del usuario. Por lo tanto, si es el tutor, es decir, el booleano EsTutor = true, se llama al servicio NotificacionService para que añada la notificación, pasándole los parámetros adecuados. Por lo tanto, la próxima vez que el residente vea sus notificaciones, verá que su tutor le ha añadido una evidencia a completar.

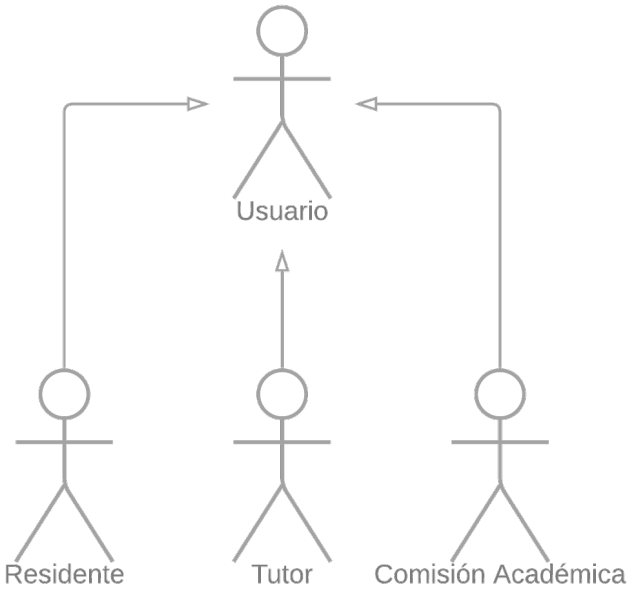

Figura 6.1: Herencia de actores en el diagrama de casos de uso

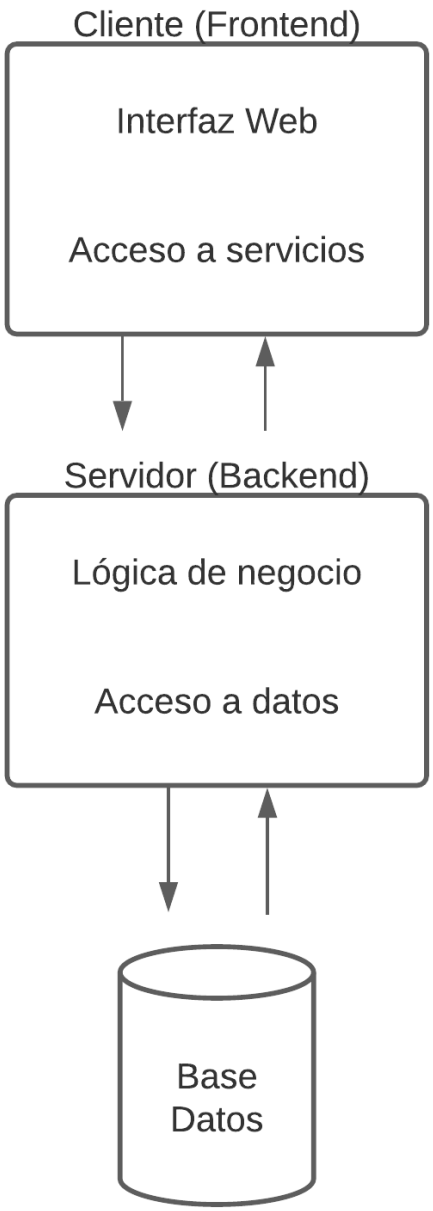

Figura 6.2: Arquitectura general desarrollada

Figura 6.3: Herencia de los tres roles con usuario

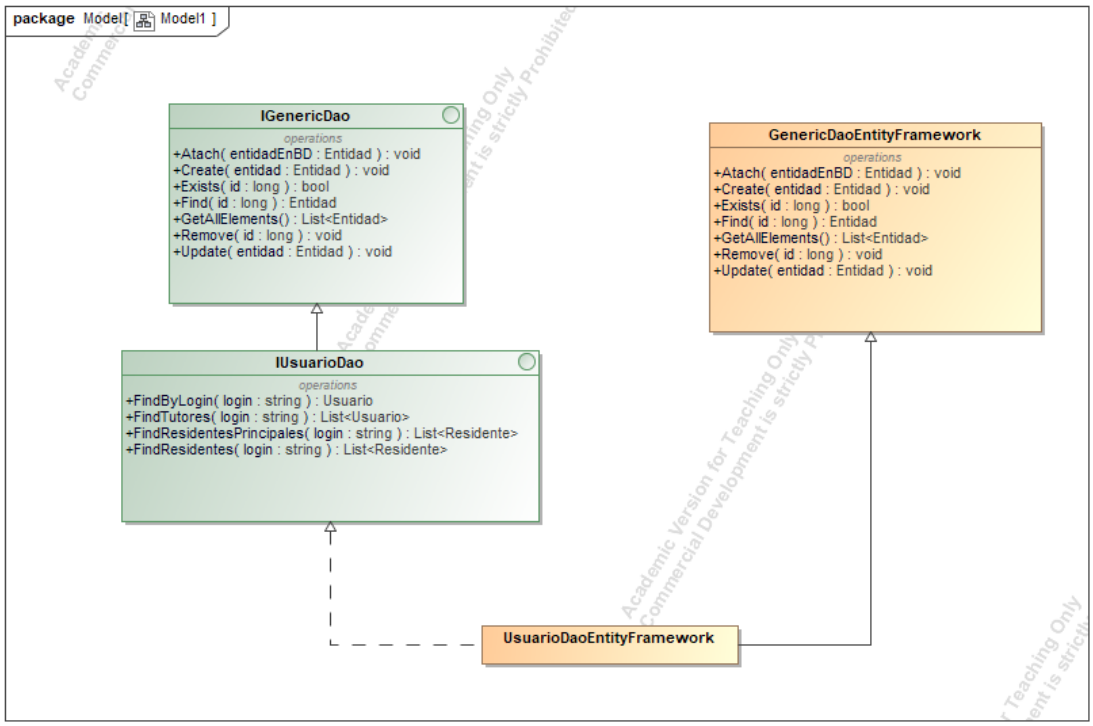

Figura 6.4: Diseño de ejemplo del DAO usuario

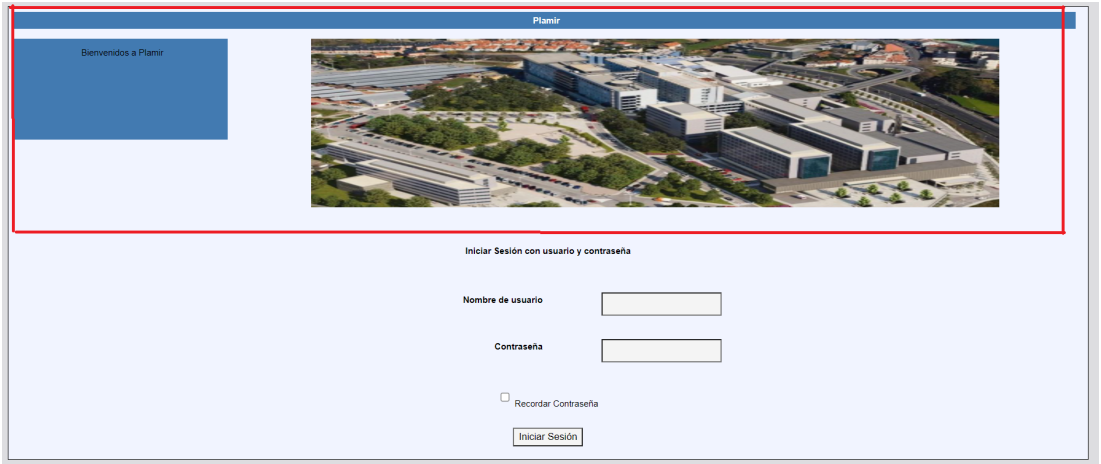

Figura 6.5: Página de Iniciar Sesión

Capítulo 7

Pruebas

Para comprobar el buen funcionamiento del software a medida que iban avanzando las iteraciones, se realizaban pruebas (para ver cómo afrontar las pruebas software, es interesante el libro [10]) que constaban de tres tipos: test de unidad, de integración y pruebas funcionales que se describirán con algo más de detalle a continuación.

7.1 Test de unidad

Los test unitarios se utilizan para comprobar exclusivamente una unidad de código. La mayoría de pruebas del sistema fueron utilizando un conjunto de unidades de código, que permiten comprobar más código y el acoplamiento entre los distintos módulos.

Aún así, se han realizado pruebas contra el DAO genérico utilizado por el sistema como interfaz de acceso a la base de datos. Como en fases tempranas del desarrollo está implementada la base de datos y las entidades correspondientes, se necesita probar que el DAO genérico funcione correctamente. En este caso hemos probado las operaciones más utilizadas e importantes, como son buscar por id, comprobar la existencia por id, las actualizaciones, las eliminaciones y el método de creación.

Para probar el DAO genérico, se ha probado con un ejemplo, en este caso con la entidad usuario, el cual tiene un DAO que extiende del DAO genérico ampliándolo en funcionalidad, pero en este caso sólo se han probado las operaciones propias del DAO genérico, que estarán de la misma forma en el resto de entidades.

7.2 Test de integración

Los test de integración realizados prueban los servicios que el backend ofrece, atendiendo sólo al interfaz de los servicios, sin tener en cuenta los detalles de implementación, es decir son pruebas de caja negra. De forma indirecta, también se prueba el funcionamiento tanto de

los DAO's como de las entidades generadas automáticamente.

Los test crean los parámetros que pide la interfaz de cada uno de los servicios, llamando a estos, y comprobando que el efecto que hacen estas llamadas concuerdan con lo anunciado en el interfaz. De esta forma, para el frontend, sólo es necesario tener a mano el interfaz de los servicios, buscando el servicio que satisfaga el cambio que queremos hacer en las entidades o la información buscada, y pasándole los atributos necesarios.

Para la implementación de estas pruebas, se ha contado con la ayuda de los Test Tools que ofrece Visual Studio. Cuando se necesitaba probar un servicio se anotaba la clase de prueba con [TestClass], y se probaban las diferentes operaciones del servico de diferente forma, anotando cada forma de prueba en un método con [TestMethod].

Al principio de cada clase de test, que prueba un servicio completo, se ejecuta un método que se debe marcar con [ClassInitialize], en el que inyectamos las dependencias que necesitemos para la ejecución de los test, como podrían ser los DAO utilizados. Después, antes de cada método de test anotado como [TestMethod], se ejecuta el método anotado como [TestInitialize()], en el que creamos los objetos necesarios que vayamos a usar en los métodos, como podrían ser la creación de entidades. Después se ejecutan los métodos en los que se hace una sóla comprobación después de una llamada al servicio, en el que se revisa que el método del servicio probado realice su tarea correctamente.

Como vemos en el ejemplo, se ha utilizado en las pruebas la clase TransactionScope, la cual permite crear bloques de código que se ejecuten en una sola transacción. Cada método ejecutará las acciones y comprobaciones que crea necesarias y acabará con un rollback después, de forma que las modificaciones realizadas no lleguen a realizarse en la base de datos.

```
1  /// <summary>
2  /// Test que comprueba el correcto funcionamiento del
3  ///    método cambiar contraseña
4  /// </summary>
5  [TestMethod]
6  public void CambiarContraseñaTest()
7  {
8          using (var scope = new TransactionScope())
9          {
10
11         // Contraseña que intentará actualizarse como nueva, la cual
12         //  es la misma pero añadiéndole al final una "X"
13         var newClearPassword = "contraseña" + "X";
14
15         // La propia llamada al servicio, pasándole el id del
16         //  usuario al quequeremos cambiar la contraseña,
17         //  la contraseña anterior,y la nueva. El método sólo
```

```
18          //  funciona si la contraseña anterior es correcta,
19          //  comprobación después de la cual se actualizará.
20          UsuarioService.CambiarContraseña(usuario.usuarioId,
    "contraseña",            newClearPassword);
21
22          // Para comprobar que la contraseña está correctamente
23          //  actualizada, encriptamos la contraseña que hemos
24          //  querido actualizar (pues en la Base de Datos está
25          //  encriptada) y la comparamos con la contraseña  actual
26          //  que está en el campo contraseña de ese usuario.
27          var encryptedNewPassword =
    EncriptadorContraseña.Crypt(newClearPassword);
28          Usuario usuarioActualizado =
    UsuarioDao.Find(usuario.usuarioId);
29
30          // Si las dos contraseñas encriptadas coinciden, no
31          //   se ha podido detectar un error
32          Assert.AreEqual(encryptedNewPassword,
    usuarioActualizado.contraseña);
33          }
34  }
```

Después de la ejecución de cada test concreto, es recomendable llamar a un método anotado con [TestCleanup], el que se encargará de eliminar las propiedades creadas en el método anteriormente nombrado que las creó. Esto es importante porque las pruebas tienen que ser independientes entre ellas, ya que si utilizan objetos comunes podemos llegar a conclusiones erróneas por la contaminación entre test, logrando efectos erróneos en el resultado de estos.

Después, una vez que se hayan ejecutado correctamente todos los test de la clase anotada con [TestClass], se ejecuta un método anotado con [ClassCleanup], en el que se limpian las dependencias inyectadas en el test marcado como [ClassInitialize].

7.3 Pruebas funcionales

Las pruebas funcionales es lo más cercano posible a la experiencia que va a tener el usuario final interactuando con el sistema. Por lo tanto se han considerado las más importantes en este proyecto, ya que no sólo se prueba que la funcionalidad se comporte correctamente, sino que también se revisa que el interfaz de usuario sea sencillo e intuitivo. Esto se ha comprobado pidiendo a personas ajenas al proyecto que realicen determinada funcionalidad, tomando nota de sus comportamientos para mejorar la usabilidad.

La funcionalidad se ha probado siguiendo la secuencia que seguiría el usuario final, comprobando que el sistema realice el comportamiento que queramos, e intentando probarlo de

formas distintas, revisando también las excepciones que puedan salir reaccionando ante determinados eventos.

Para esto, Visual Studio permite ejecutar los Web Forms con un simple movimiento de ratón. Estos Web Forms se visualizan en un navegador, en el que podemos interactuar y realizar las pruebas oportunas. También en paralelo podemos revisar los efectos de las funcionalidades en la base de datos, lanzando consultas de forma cómoda con la herramienta SQL Management Studio (sección 3.3.2, página 12)

Capítulo 8

Funcionalidades destacadas

En este capítulo, se enumerarán y explicarán las funcionalidades más destacadas de lo que pueden hacer los usuarios del sistema mediante el interfaz web.

8.1 Añadir evidencias desde el rol residente

Un usuario con la sesión iniciada con el rol residente, podrá seleccionar en el menú el elemento evidencias. Una vez seleccionado, lo primero que vemos es un desplegable en el que veremos las rotaciones en las que esté el residente.

Una vez seleccionado la rotación del desplegable, podremos ver las evidencias añadidas en esa rotación con paginación, de las que podemos ver los detalles pulsando el botón Ver Detalle al lado de las evidencias.

Si consideramos que falta alguna evidencia, podremos pulsar el botón Añadir Evidencia (botón que se señala en la imagen 8.1). Esta acción nos llevará a una nueva página en forma de formulario, en la que está explicado el procedimiento a seguir. Debemos añadir los siguientes datos:

- En nombre de la evidencia, en el cuadro de texto con este nombre, debemos introducir un nombre descriptivo que vaya a identificar la evidencia
- En el campo observaciones describiremos el objetivo de la evidencia a crear y detalles a destacar
- URL, campo que rellenaremos si consideramos necesario aportar algún recurso, aportando la referencia para llegar hasta él.
- En añadir archivos, nos llevará al explorador de archivos, en el que seleccionaremos un archivo si lo deseamos que complete la evidencia.

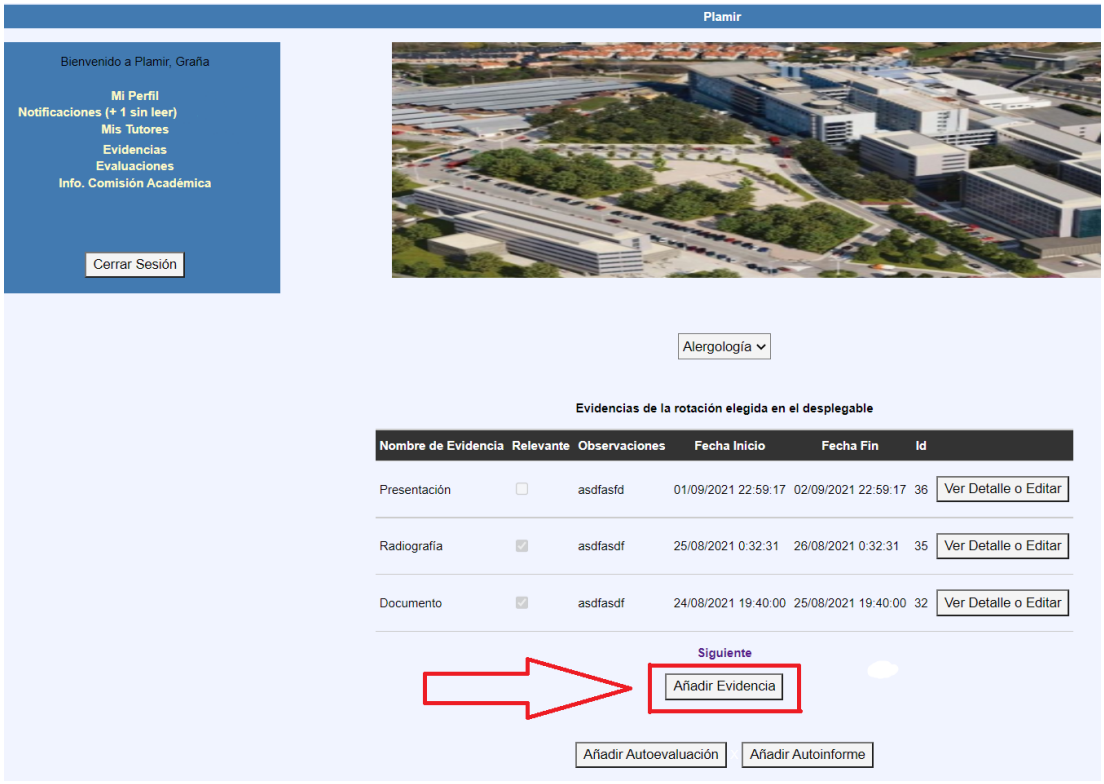

Figura 8.1: Botón añadir evidencias

- En el seleccionable relevante, marcaremos si el residente ve necesario destacar la evidencia.

Después de rellenar los datos que consideremos importantes, le damos al botón de confirmar añadir (botón en figura 8.2), y si no hay ninguna excepción, veremos la evidencia añadida en la lista de evidencias.

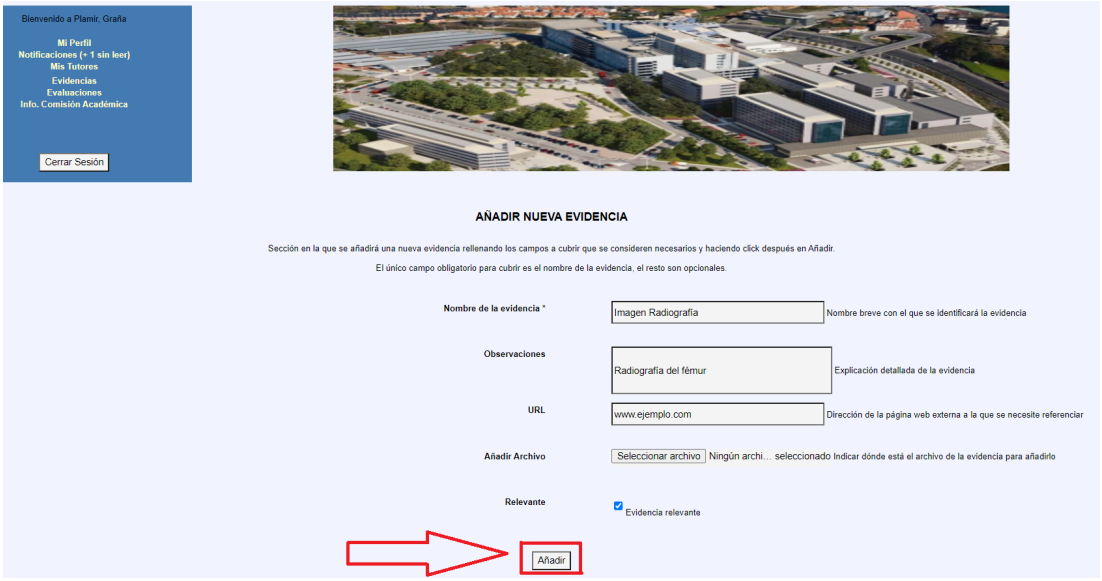

Figura 8.2: Botón de confirmar añadir evidencia

8.2 Añadir archivos a una evidencia desde el rol residente

Desde la pantalla de ver evidencias anteriormente descrita, podremos pulsar el botón Ver o Editar Evidencia al lado de la evidencia a la que queramos añadir un archivo. Después, veremos datos propios de la evidencia, como su nombre, observaciones, si es relevante etc. Y justo debajo, vemos (si existen) la lista de archivos que completan la evidencia. Si consideramos que falta algún archivo, podremos pulsar el botón añadir archivo, el cual nos llevará a otra pantalla con un formulario. En este (vista que podemos ver en 8.3), podremos poner nombre al archivo que vamos a añadir y abrir el explorador de archivos en el botón seleccionar archivo, en el que buscaremos el archivo que vayamos añadir a la evidencia en nuestro equipo. Después de confirmar añadir archivo, volvemos a ver la lista de archivos de la evidencia, con el archivo añadido a la lista si no ha habido ningún error.

8.3 Añadir evidencias a un residente desde el rol tutor

La forma de añadir evidencias desde el punto de vista del tutor es similar a la de un residente, por lo que destacaremos los cambios para no repetirnos. A esta funcionalidad llegamos después de seleccionar un residente que tengamos como específico de rotación. Después veremos su lista de evidencias, y si vemos que falta alguna, pulsaremos el botón añadir evidencia.

La pantalla cambia en el rol de tutor respecto al residente (como podemos ver en la imagen 8.4) que no puede añadir archivo que completen la evidencia, ya que la explicación de la tarea a realizar debería ir en el campo observaciones.

Al darle a añadir evidencia, se genera una notificación que pueda ver el residente, para que se de cuenta de que su tutor de esa rotación le acaba de añadir una evidencia.

8.4 Ver notificaciones generadas por el sistema para cada usuario

Cualquier usuario podrá ver las notificaciones que el sistema le ha generado de forma automática. Esto se puede hacer seleccionando el apartado notificaciones en el menú (en el que podremos apreciar en el nombre si tenemos alguna notificación).

En la pantalla de ver notificaciones podremos ver la lista de notificaciones (como se puede ver en la imagen B.3) sin leer primero, y las ya leídas después. Para ver una notificación, se pulsará en el botón al lado de la notificación, en la que veremos el asunto y el mensaje completo. Una vez leída la notificación, pasará de estar en la lista de no leídas a leídas.

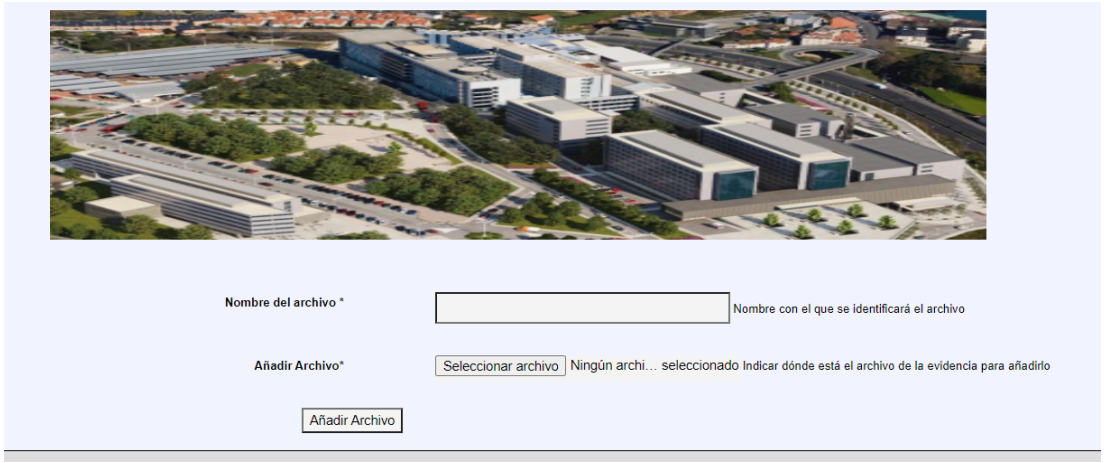

Figura 8.3: Pantalla para añadir archivo

Figura 8.4: Pantalla para añadir evidencia desde tutor

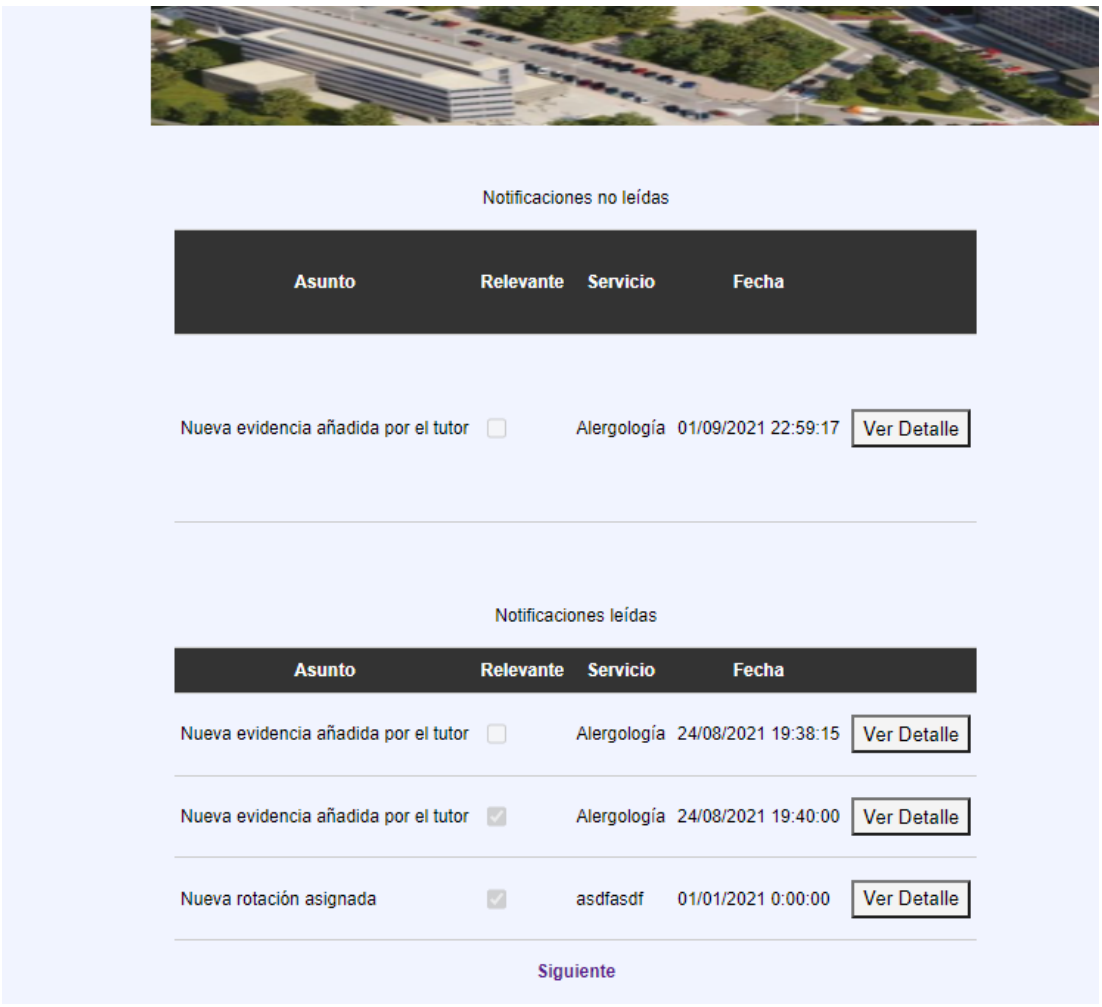

Figura 8.5: Vista de pantalla notificaciones, con paginación

Capítulo 9

Conclusiones

Después del desarrollo del proyecto, concluimos el trabajo académico analizando si se han cumplido y en qué medida los objetivos propuestos al inicio del documento (sección 1.3, pagina 3). También se comentará lo aprendido a lo largo del trabajo y las líneas futuras de mejora.

9.1 Cumplimiento de objetivos

Uno de los objetivos más importantes del trabajo fue repasar los conocimientos adquiridos en asignaturas que se han cursado en el grado. Durante el proyecto fue necesario repasar conceptos de asignaturas de otros años, lo que ayudó a afianzar los conocimientos que se han adquirido, llevando a la práctica la teoría aprendida, y familiaridad con las tecnologías usadas, lo que es de ayuda no sólo para el uso de las mismas, si no por la facilidad de aprender otras nuevas. Muchos de los conceptos repasados son muy importantes independientemente de la tecnología, por lo que fue uno de los aspectos más importantes aprendidos.

Otro objetivo fue entender un sector en el que no estaba familiarizado. En un principio, puede parecer fácil entender procesos de sectores profesionales, pero adaptarse rápido a estos no es tarea trivial, por lo que haciendo el esfuerzo de entender el sector sanitario, aportará facilidad para entender otros sectores en un futuro.

Y como último objetivo, y no por ello menos importante, el de aprender a planificarse y seguir una metodología inspirada en las metodologías ágiles. Aprender los conceptos básicos de este tipo de metodologías es muy importante, ya que es lo más común en los tiempos actuales, por lo que llevarlo a la práctica será de bastante utilidad en un futuro.

9.2 Lecciones aprendidas

Las lecciones aprendidas a destacar durante el trabajo fueron las siguientes:

- A diferencia de otros trabajos académicos, desarrollar un sistema pensando en un entorno real, te das cuenta que los requisitos no están tan claros como en un enunciado de una práctica. Es necesario saber hacer preguntas, ser activo, ponerse en la perspectiva de otras personas, pensar diferentes situaciones que se puedan dar en la interacción con el sistema, etc.

- La facilidad de desarrollar con una metodología basada en el desarrollo iterativo e incremental, ya que permite ir comprobando poco a poco las funcionalidades. Esto permite tanto detectar los errores que se van cometiendo rápidamente, pues se revisa cada poco tiempo, y la satisfacción al desarrollar al ir viendo que el sistema va creciendo poco a poco.

- La importancia de las primeras fases del desarrollo. Es un error muy grave pensar que la fase de implementación es la más importante, pues la que más marca el proyecto son las primeras, ya que una buena definición de requerimientos, y un buen diseño general del sistema permite ganar mucha velocidad y confianza en fases posteriores, y evita tener que corregir errores, lo cual enlentece el desarrollo enormemente.

- Las buenas prácticas de programación son más importantes de lo que pueda parecer. Hacer un código limpio, fácilmente extensible y desacoplado, hace que en posteriores iteraciones, sea mucho más fácil añadir funcionalidades, ya que el sistema está bien estructurado y está previsto que vaya creciendo.

9.3 Líneas futuras

En los siguientes apartados, se propondrán una serie de mejoras al sistema que se podrían hacer para mejorarlo.

9.3.1 Integración con base de datos

Actualmente, los datos del sistema se almacenan en una base de datos propia, de la que se leen todos los datos necesarios. Esto en la realidad no sería mantenible, ya que por ejemplo, los datos de los residentes y tutores están almacenados en una base de datos a la que tiene acceso el hospital, donde está la información actualizada y completa.

La mejora consistiría en que determinados datos como los anteriores, se lean de una base de datos ajena al sistema, guardando en la base de datos propia de la aplicación sólo información propia que genere y mantenga el sistema.

9.3.2 Cliente de correo electrónico

Los usuarios del sistema es posible que no accedan a la aplicación diariamente, y sólo cuando necesiten hacer alguna gestión. Por este motivo, las alertas que genere el sistema sólo las ven cuando accedan al sistema, pudiendo perder un evento que sea de urgencia.

Para solucionar este problema, se podría proponer el uso de un cliente de correo electrónico que cuando el sistema genere una alerta al usuario, le notifique al correo electrónico, que sí debería consultar a diario, por lo que no se perderá ningún evento importante.

9.3.3 Mejoras en el interfaz

El interfaz se ha buscado que sea sencillo, por lo que se ha optado por un diseño funcional y simple. Esto podría provocar que un uso continuado del sistema aburra al usuario por el diseño un poco más arisco de lo deseado.

Para solucionarlo, se podrían usar librerías ya implementadas, que de forma sencilla mejoren el aspecto de interfaz gráfico, con adornos que hagan más agradable el uso continuado, sin perder la funcionalidad y sencillez.

Apéndices

Apéndice A

Manual de Instalación

En este capítulo se expondrán brevemente los aspectos a tener en cuenta para poner en producción la aplicación.

A.1 Paso a producción

En primer lugar, es necesario tener el servicio de bases de datos activado correctamente. Para esto, debemos contar con un Microsoft SQL Server, y crear una base de datos llamada 'plamir' y otra 'plamirTest'. Después, desde línea de comandos, con el comando sqlcmd.exe, ejecutar el script de creación de tablas (llamado SqlServerCreateTables.sql) tanto para la base de datos de producción ('plamir'), como para test ('plamirTest'). Después, debemos crear los usuarios de la aplicación, ya que no está conectada a una base de datos real de residentes y tutores, con SQL Server Management Studio.

Para poner en producción la aplicación después de tener lista la base de datos, se deberá instalar ASP.NET, y habilitar los módulos de de IIS y ASP.NET. Después, se deberá configurar el servidor IIS y debemos alojar el sitio web en cuestión para que funcione con dicho servicio.

Como recomendación, se deberá instalar el sistema cerca del directorio raíz, ya que si la ruta es demasiado larga, a la hora de añadir archivos desde el interfaz del sistema, puede dar problemas no deseados. También hay que tener en cuenta que los archivos que se suban al sistema desde el interfaz se van a guardar en el servidor, por lo que el espacio de almacenamiento debe poder almacenar todos los archivos que se calculen que se vayan a subir.

Apéndice B

Manual de Usuario

En este anexo, se enseñará el interfaz web del sistema y se explicará brevemente cómo se debe usar.

Para su correcta visualización, sólo se mostrará la parte del interfaz web que cambie. Un ejemplo de vista completa del interfaz es B.2. El resto de funcionalidades sólo muestran la parte de abajo.

Otro detalle a tener en cuenta es que existe paginación, es decir, donde se muestran los datos en formato de tabla, se verá un anterior y siguiente para navegar por los distintos bloques de datos.

B.1 Funcionalidades generales de Usuario

En la imagen B.1 podemos ver que para el inicio de sesión debemos introducir el login de usuario y la contraseña (que se verá con asteriscos por seguridad). Si deseamos que al cerrar el navegador y al volver a abrirlo nos guarde la sesión, debemos marcar el recuadro Recordar contraseña. Si los datos son correctos (en caso contrario el interfaz indicará si el fallo es que el login no existe o la contraseña es incorrecta), iniciará sesión y nos llevará a B.2. En esta pantalla, aparte de ver el menú propio del rol de cada usuario, en este caso residente, vemos en la parte central detalles como el nombre, apellidos o contacto.

Un usuario también puede ver las notificaciones B.3 que genera el sistema para él. En esta pantalla ve tanto las notificaciones no leídas como las leídas. En el caso de darle a ver detalle a una nueva evidencia, donde ve el contenido del mensaje completo, la notificación pasará a notificaciones leídas.

Un usuario, además, puede enviar mensajes a otro usuario, en la ventana enviar mensaje B.4. En esta ventana se verá a quién se envía el mensaje y se podrá escribir el contenido que deseamos enviar. También se podrá ver los mensajes que nos ha enviado un usuario, en la ventana de ver mensajes B.5.

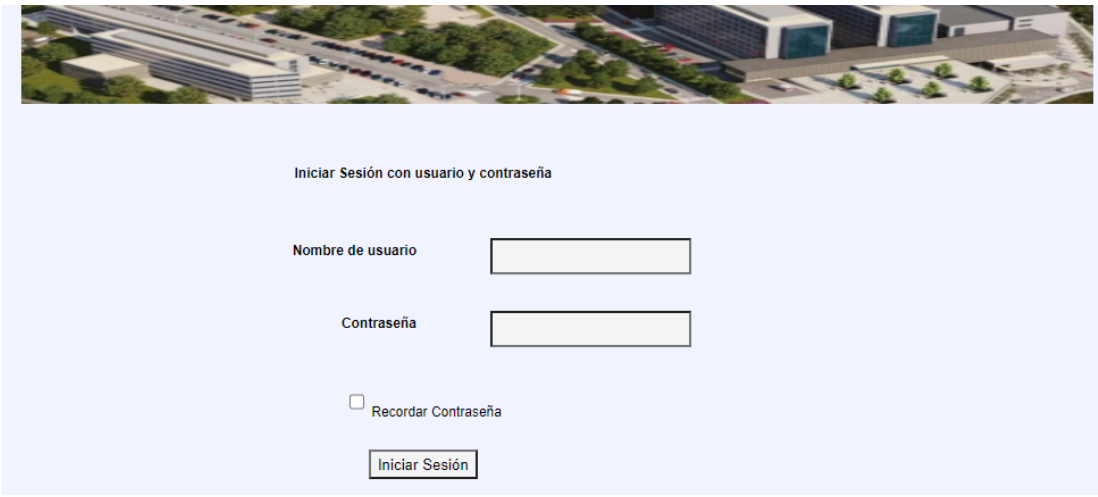

Figura B.1: Pantalla en la cual el usuario inicia sesión

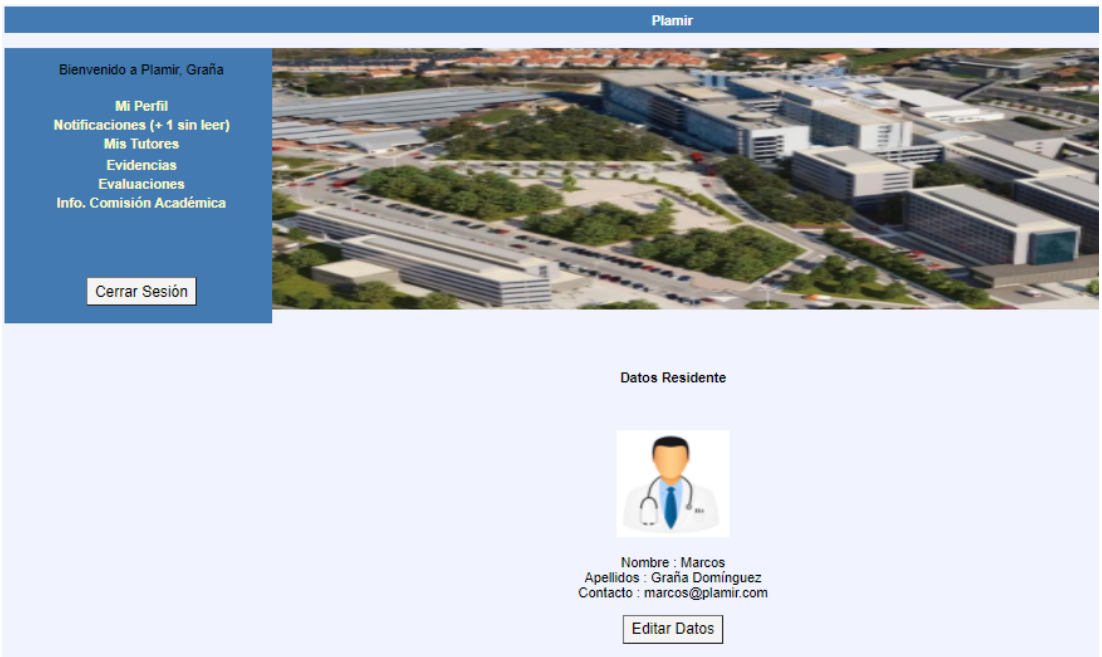

Figura B.2: Pantalla en la cual el usuario puede ver sus datos en el sistema

Figura B.3: Pantalla donde el usuario puede ver sus notificaciones

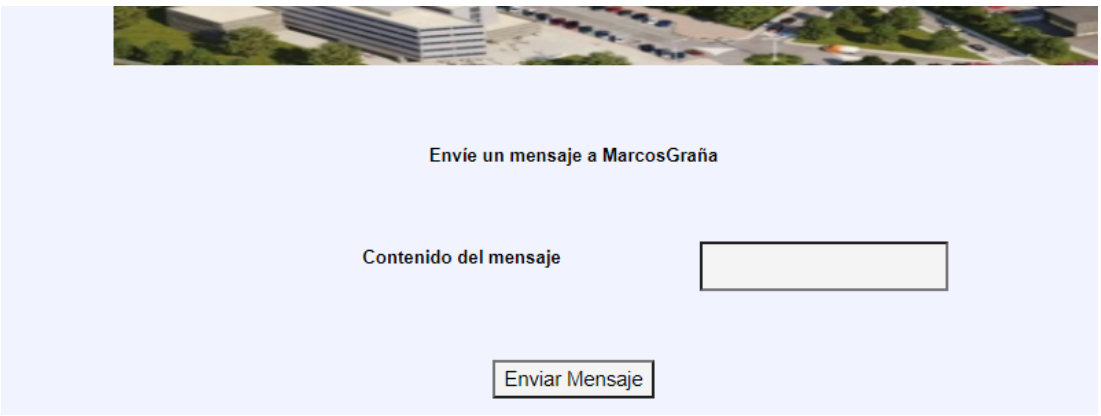

Figura B.4: Pantalla donde el usario puede enviar un mensaje a otro usuario

Figura B.5: Pantalla donde el usario puede ver mensajes de otro usuario

B.2 Funcionalidades generales de Residente

Un usuario que es residente, puede ver en la pantalla mis tutores B.6 su tutor de residencia, y sus tutores específicos de las rotaciones en las que están. Además de ver su nombre o correo electrónico, puede enviar o ver mensajes con ese usuario.

Figura B.6: Pantalla donde el residente puede ver sus tutores

En esta pantalla B.7 se puede elegir la rotación en el desplegable. Después, se verán las evidencias de esa rotación, evidencias que se podrán ver el detalle de evidencia B.8 (en la que

podrá ver los detalles, archivos y mismo añadir un archivo) o añadir una nueva evidencia (en los que añadirá el nombre, otros datos y el archivo desde el explorador de archivos) B.9

Figura B.7: Pantalla donde el residente puede ver evidencias de una rotación seleccionada

B.3 Funcionalidades generales de Tutor

Un usuario que tenga de rol en el sistema de tutor, podrá ver sus residentes B.10. En esa vista podrá ver los residentes generales de residencia, y los específicos de rotación, en los que podrá ver evidencias, enviar y leer mensajes y añadir calificaciones. Al ver las evidencias, puede destacarlas como importantes, o ver detalles de evidencia. En esa pantalla B.11, puede ver los datos de la evidencia, y además descargar los archivos que haya subido el residente a esa evidencia. En el caso de que quiera añadir una nueva evidencia, en la pantalla ver evidencias de un residente, pulsará añadir evidencia, y en la pantalla añadir evidencia 8.4 le añadirá un nombre de la evidencia, las observaciones, una url que pueda ser de ayuda para hacer la evidencia y si es relevante o no. Después, pulsará añadir y verá la evidencia añadida en la lista de evidencias.

Figura B.8: Pantalla donde el residente puede ver el detalle y los archivos de una evidencia

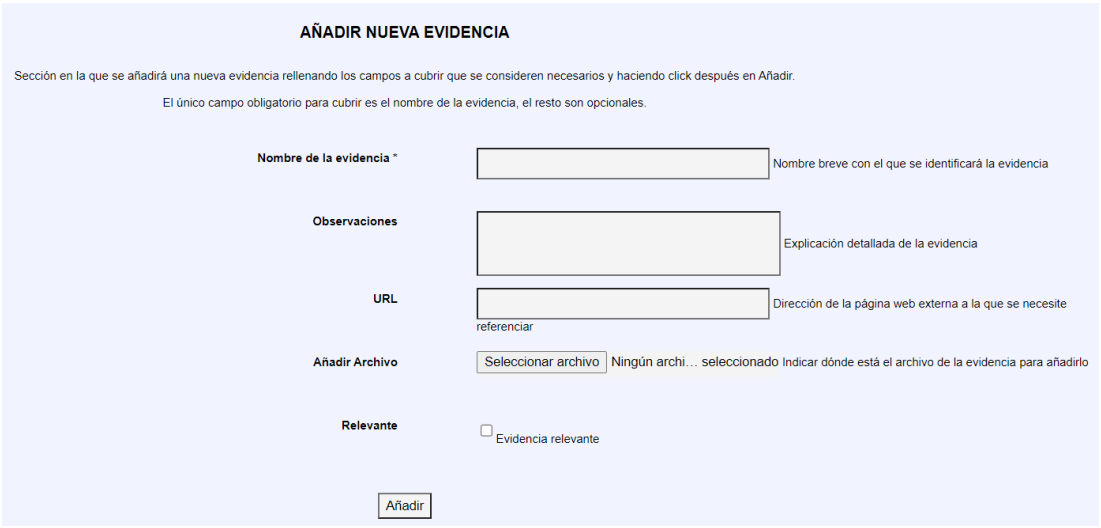

Figura B.9: Pantalla donde el residente añade una evidencia

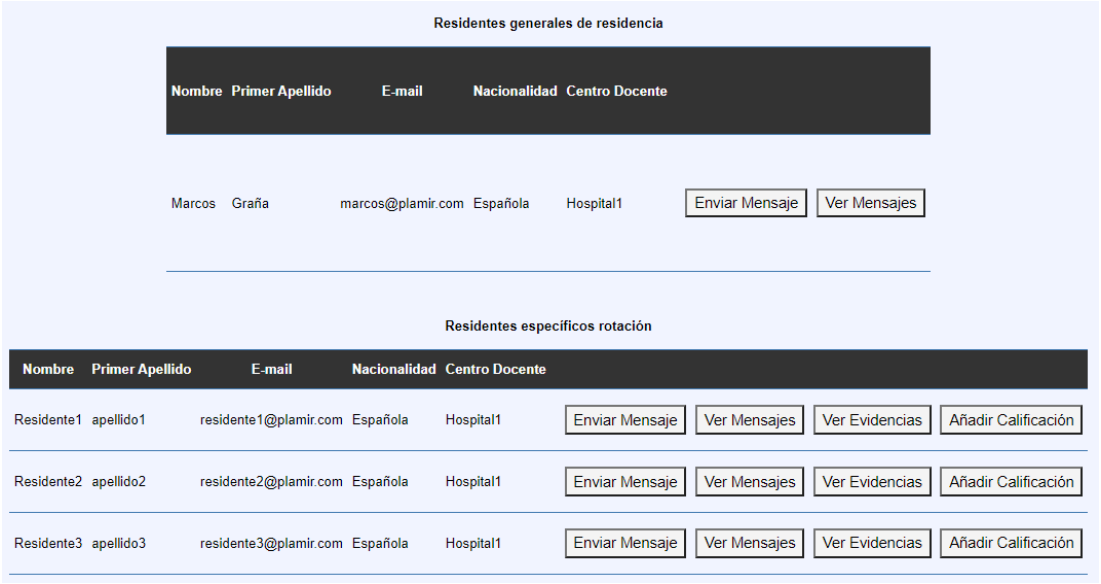

Figura B.10: Pantalla donde un tutor ve sus residentes asociados

Figura B.11: Pantalla donde un tutor puede ver los detalles y archivos de una evidencia

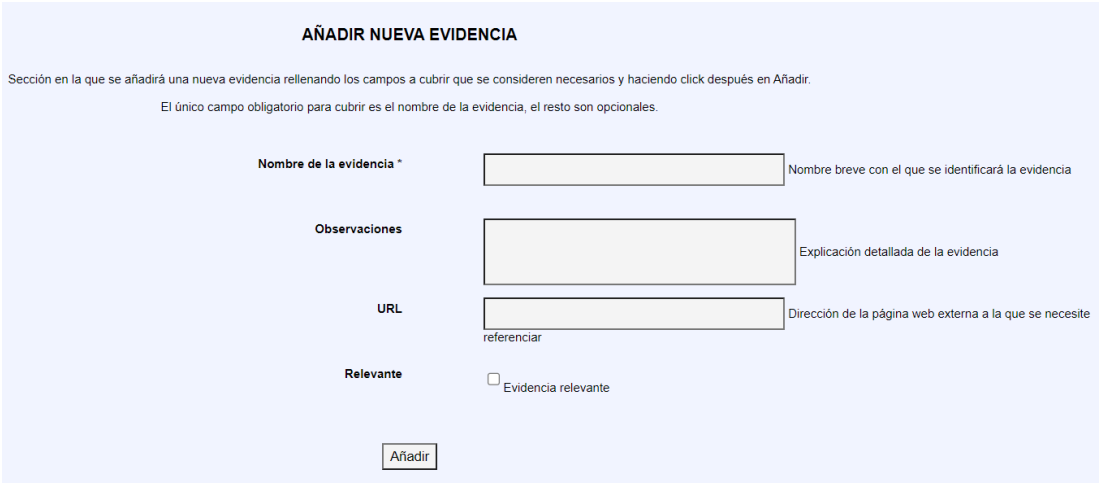

Figura B.12: Pantalla donde un tutor le añade a un residente una evidencia a rellenar por el residente

Bibliografía

[1] D. Ortego, "¿qué es c#?" 2017. [En línea]. Disponible en: https://openwebinars.net/blog/que-es-c-introduccion/

[2] "Html: Lenguaje de etiquetas de hipertexto," 2021. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML

[3] "Microsoft visual studio," 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Microsoft_Visual_Studio

[4] "Git," 2021. [En línea]. Disponible en: https://git-scm.com/

[5] "Flujo de trabajo de gitflow," 2021. [En línea]. Disponible en: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow

[6] "Manifiesto por el desarrollo Ágil de software," 2001. [En línea]. Disponible en: http://agilemanifesto.org/iso/es/manifesto.html

[7] "Salario medio para programador .net en españa 2021," 2021. [En línea]. Disponible en: https://es.talent.com/salary?job=programador+.net

[8] "¿cuánto gana un jefe de proyecto?" 2021. [En línea]. Disponible en: https://www.glassdoor.es/Sueldos/jefe-de-proyecto-sueldo-SRCH_KO0,16.htm

[9] R. C. Martin, *Arquitectura Limpia*, 1st ed. Anaya, 2018.

[10] ——, *El limpiador de código: Código de conducta para programadores profesionales*, 1st ed. Anaya, 2019.