



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un sistema software para la gestión de las operaciones de una central nuclear

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Héctor Jaraba Romero

Tutor: Santiago Escobar Román

2018-2019

Resumen

El programa informático realizado tiene como función formar los turnos de una central nuclear en la que distintos roles podrán realizar diferentes operaciones, como indicar incidencias, crear maniobras, gestionar órdenes de funcionamiento, generar listados de reportes, entre otras. Durante el ciclo de vida del proyecto informático se ven las diferentes etapas involucradas en el desarrollo del software, desde cómo se modela el producto a partir de los requisitos obtenidos en la fase de análisis hasta su implementación.

Para la realización del producto software se utiliza el lenguaje de programación C# bajo el IDE Microsoft Visual Studio, utilizando una arquitectura por capas, patrones de diseño como Inyección de Dependencias, además se aplican principios SOLID con la finalidad de tener un producto robusto y fácilmente mantenible.

Palabras clave: desarrollo software, gestión, central nuclear, SOLID, C#

Resum

El programa informàtic realitzat te com a funció formar els torns d'una central nuclear en què distints rols podran realitzar diferents operacions, com indicar incidències, crear maniobres, gestionar ordes de funcionament, generar llistats de reports, entre altres. Durant el cicle de vida del projecte informàtic es veuran les diferents etapes involucrades en el desenvolupament del software, des de com es modela el producte a partir dels requisits obtinguts en la fase d'anàlisi fins a la seua implementació.

Per a la realització del producte software s'utilitza el llenguatge de programació C# davall l'IDE Microsoft Visual Studio, utilitzant una arquitectura per capes, patrons de disseny com Injeccio de Dependencies, a més s'apliquen principis SOLID amb la finalitat de tindre un producte robustament i fàcilment mantenible.

Paraules clau: desenvolupament software, gestió, central nuclear, SOLID, C#

Abstract

The performed program serves to form shifts of a nuclear power plant in which different roles will be able to perform different operations, such as indicating incidents, creating manoeuvres, managing operating orders, generating lists of reports, among others. During the life cycle of the project, different stages involved in the development of the software are performed. From how the product is modelled regarding the requirements obtained in the analysis phase to its implementation.

In order to create the software product we use the programming language C# under the IDE Microsoft Visual Studio, using a layered architecture and design patterns as Injection of Dependencies. In addition, SOLID principles are applied in order to have a robust and easily maintainable product.

Keywords : software development, management, nuclear power plant, SOLID, C#.

Dedicado a mis abuelos, Jesús y Carmen, por ser unos segundos padres para mí.

A mis padres, Jesús y Carmen, que me enseñaron que trabajando duro uno puede llegar donde quiera.

A mi chica, Cathlene, por cruzarse en mi vida y completarla.

Tabla de contenidos

1.	Introducción	12
1.1.	Motivación	13
1.2.	Objetivos	13
1.3.	Estructura de la memoria.....	14
2.	Estado del arte	16
2.1.	La empresa.....	16
2.2.	Mis prácticas profesionales.....	16
3.	Análisis y elicitación de requisitos.....	18
3.1.	Técnicas de elicitación de requisitos.....	18
3.2.	Actores	20
3.3.	Descripción de los casos de uso.....	21
3.4.	Diagrama de casos de uso	26
3.4.	Prototipo	26
4.	Diseño	32
4.1.	Arquitectura	32
4.2.	Diagrama de relaciones.....	35
4.3.	Esquema de base de datos	36
5.	Implementación	41
5.1.	Stack tecnológico	41
5.1.1.	C# y .NET framework	41
5.1.2.	Entity Framework.....	41
5.1.3.	LINQ.....	44
5.1.4.	WPF	45
5.1.5.	KOA FRAMEWORK	46
5.2.	Metodología.....	46
5.3.	Workflow.....	48
5.3.1.	Control de versiones.....	48
5.4.	Buenas prácticas empleadas durante el desarrollo.....	49
5.5.	Código limpio.....	49
5.6.	Principios SOLID	51
5.6.1.	Principio de responsabilidad única.....	52
5.6.2.	Principio abierto/cerrado.....	52

5.6.3.	Principio de sustitución de Liskov	52
5.6.4.	Principio de segregación de interfaz	52
5.6.5.	Principio de inversión de dependencias.....	52
5.7.	Patrones de diseño.....	52
5.7.1.	Patrón inyección de dependencias	53
5.8.	Observaciones.....	57
6.	Resultado	59
7.	Conclusiones	66
8.	Glosario de términos.....	68
9.	Referencias.....	69



Tabla de ilustraciones

Figura 1: Actores de la aplicación	20
Figura 2: Diagrama de casos de uso	26
Figura 3: Prototipo de pantalla de inicio de sesión	27
Figura 4: Prototipo de asignación de roles	27
Figura 5: Prototipo de inicio de turno	28
Figura 6: Prototipo de menú.....	28
Figura 7: Prototipo de listado de maniobras.....	29
Figura 8: Prototipo de creación de maniobra	29
Figura 9: Prototipo búsqueda de prueba.....	30
Figura 10: Arquitectura cliente-servidor	32
Figura 11: Arquitectura por capas.....	33
Figura 12: Diagrama de relaciones simplificado.....	35
Figura 13: Esquema completo de la base de datos 1.....	37
Figura 14: Esquema completo de la base de datos 2.....	38
Figura 15: Esquema completo de la base de datos 3.....	39
Figura 16: Entidad MPL	43
Figura 17: Relaciones MPL Entity Framework	43
Figura 18: Comparador cambios base de datos.....	44
Figura 19: Servicios creados.....	44
Figura 20: LINQ obtener todas las fugas.....	45
Figura 21: LINQ búsqueda fugas	45
Figura 22: Fecha Inicio Vista.....	45
Figura 23: Fecha Inicio WPF.....	46
Figura 24: Depurado Fecha Inicio	46
Figura 25: Kanban trabajo	48
Figura 26: Variable fecha inicio	50
Figura 27: Función CanExportar	50
Figura 28: Comentarios	50
Figura 29: Comentarios TODO	51
Figura 30: Regiones	51
Figura 31: Región Commands expandida.....	51
Figura 32: inyección de dependencias 1	53
Figura 33: Inyección de Dependencias 1 código.....	53
Figura 34: Inyección de Dependencias 2	54
Figura 35: Inyección de dependencias 2 PaginaVista.....	54
Figura 36: Inyección de dependencias 2 ContenedorDependencias.....	54
Figura 37: Interfaz y clases.....	55
Figura 38: Clase PaginaVista.....	56
Figura 39: Instalador de dependencias	56
Gráfico 1: Líneas de código por lenguaje.....	57
Gráfico 2: Líneas de código por tipo	57



Figura 40: Pantalla login	59
Figura 41: Pantalla composición turno	59
Figura 42: Pantalla inicio turno.....	60
Figura 43: Pantalla control de turnos	60
Figura 44: Pantalla mis maniobras.....	61
Figura 45: Pantalla listado maniobras	61
Figura 46: Pantalla detalle maniobra.....	62
Figura 47: Pantalla Busqueda de maniobras	62
Figura 48: Pantalla aviso alertas.....	63
Figura 49: Pantalla lista maniobras rol encargado de exteriores	63
Figura 50: Pantalla detalle maniobra rol encargado de exteriores.....	64

1. Introducción

Cuando se va a construir una casa, no basta con apilar ladrillos unos encima de otros, colocar unas ventanas y un techo. Normalmente, construir una casa suele ser más complejo, teniendo que pasar por una serie de procesos, como que un arquitecto haga el diseño y posteriormente se organice su construcción para que se pueda llevar a cabo con éxito.

Un proceso de desarrollo de software no difiere mucho de esta analogía. Aunque este sea intangible, si empezamos a programar antes de siquiera tener un modelo que seguir o unos requisitos correctamente definidos, por muy pequeño que sea nuestro programa, su desarrollo puede acabar siendo un absoluto fracaso y “derrumbarse” sobre nosotros, ya que no contaremos con una idea clara e incurrimos en el desorden. Y si a esto le añadimos que no somos los únicos involucrados en el desarrollo del programa, este trabajo puede convertirse en un infierno tanto para nosotros como para nuestros compañeros.

Por esta razón y otras cuantas más que se explicarán a lo largo de este documento, es importante seguir una serie de pautas a la hora de comenzar el desarrollo de un producto software.

El proceso de desarrollo de software suele estar ligado a una gran incertidumbre. Los que hayáis tenido que desarrollar software para un cliente (o incluso en desarrollos propios), sabréis que lo que en un principio parece muy claro, a medida que el desarrollo avanza, comienzan a surgir cambios y nuevos requisitos que nos obligan a alterar el programa. Por eso lo ideal sería que nuestro programa fuese lo suficientemente susceptible a cambios y tuviésemos una metodología sólida para que no se convierta en un reto y seamos efectivos a la hora de modificarlo o añadir nuevas funcionalidades.

Me llamo Héctor Jaraba, he cursado Ingeniería Informática en la especialización de la rama de Ingeniería del Software en la Universidad Politécnica de Valencia, también soy técnico superior en desarrollo web. Mi desarrollo en el campo de la informática no tan solo es profesional, también vocacional, desde muy niño me sentí atraído por la informática y terminé descubriendo a lo que me quería dedicar cuando a los 8 años descubrí el mundo de la programación realizando mi primer script en Batch. He desarrollado diferentes proyectos software y he participado en eventos relacionados con la informática y el emprendimiento. Este trabajo nace como resultado de unas prácticas profesionales en una consultora multinacional.

1.1. Motivación

La principal motivación a la hora de escoger este trabajo de fin de grado reside en poder participar en el proceso de desarrollo de un programa para una central nuclear debido al reto que puede suponer.

Asistí a reuniones con la unidad radiológica y el encargado designado por la central nuclear para obtener *feedback* sobre que nuevos requisitos se deseaban incorporar (como la distinción entre tipos de pruebas de la central, añadir un botón para lanzar una aplicación externa, etc.) o las funcionalidades a pulir.

Me incorporé al equipo de desarrollo, donde pude solucionar distintos tipos fallos de la aplicación y desarrollar nuevas funcionalidades para la misma. Todo ello siguiendo una metodología de trabajo ágil que se explica más adelante.

1.2. Objetivos

Este trabajo de fin de grado trata el desarrollo de un producto software cuyo objetivo es gestionar diversas operaciones en una central nuclear.

Se parte desde el punto en el que la empresa cuenta con los documentos de especificación de requisitos y funcionalidades que ha pactado con el cliente y se realizara una primera versión del programa.

El programa llevará un seguimiento de tipo administrativo de las distintas operaciones realizadas por los integrantes de los turnos de la central nuclear y se encargará de gestionarlos.

También, mostrará una lectura de información de distintos parámetros obtenidos de la central nuclear y tendrá un acceso directo que servirá para lanzar una aplicación externa perteneciente a SAP desde dentro del programa.

El programa hará peticiones a una serie de servicios SOAP que se encargarán de comprobar las acciones que un usuario puede o no puede realizar en base a los permisos que se le han asignado y comunicárselo al programa. Una emulación de este servicio es proporcionada y no formará parte del desarrollo.

Se pretende ilustrar el ciclo de vida del programa y aclarar por el camino, qué pasos y decisiones se tuvieron en cuenta a la hora de su desarrollo. Se explicarán las diferentes etapas, tecnologías y practicas utilizadas, exponiendo el significado de los conceptos en los que se crea necesario profundizar para resultar lo más didáctico posible.

El software que aquí se expondrá se trata de un producto real, y por lo tanto debido a la política de privacidad de la empresa no se desvela en ningún momento datos reales sobre el producto o la empresa. Todos los datos que se exponen han sido generados expresamente con fines demostrativos para la realización de este TFG.

He de aclarar que aquí se expone una manera de afrontar un proyecto software en un contexto real. Esta no es la única solución, pero sí es de entre las que se sopesaron la que se creyó más conveniente de aplicar a este proyecto. Volviendo a la analogía de la construcción de casas, no construiríamos de la misma manera una cabaña que un rascacielos, y dependiendo de qué tipo de rascacielos se tomarían unas decisiones u otras que conducen a diferentes maneras de construirlo.



1.3. Estructura de la memoria

Este documento está estructurado conforme a los siguientes puntos:

1) **Introducción**

Se presenta la motivación que ha llevado a realizar este documento y los objetivos que se pretenden tratar en él.

2) **Estado del arte**

Se presenta la empresa en la que se ha realizado este proyecto junto con su historia y trayectoria.

3) **Análisis y elicitación de requisitos**

Se hará un breve repaso a algunas técnicas para el análisis y elicitación de requisitos, y se verá cuáles de ellas se han utilizado para conseguir realizar el producto software que aquí se expone.

4) **Diseño**

Se muestra cómo se llega a plantear una solución que satisface los requisitos recolectados en la anterior etapa

5) **Implementación**

Aquí se trata las tecnologías que se han decidido utilizar, algunas buenas prácticas, como el uso del código limpio y la metodología de trabajo que se ha decidido utilizar, además de los aspectos que se consideran relevantes en la implementación del programa.

6) **Resultado**

Se muestra el resultado de la aplicación obtenida mediante capturas de pantalla.

7) **Conclusiones**

Se presenta lo conseguido en este proyecto, los problemas presentados y algunas posibles recomendaciones desde el punto de vista del autor.

8) **Glosario de términos**

Se definen algunos términos que se considera que merecen ser explicados para la comprensión del documento.

9) **Referencias**

Se muestran las referencias que han servido de inspiración a la hora de redactar este documento.

2. Estado del arte

En esta sección se introduce a la empresa gracias a la cual ha sido posible la realización de este trabajo.

2.1. La empresa

La empresa donde realice mis practicas se trata de una empresa multinacional española de consultoría que desarrolla soluciones software a medida y proporciona servicios de *outsourcing*. La compañía está formada por más de 21.000 profesionales y abarca sectores como telecomunicaciones, banca, sanidad, administración pública, educación, defensa, industria, entre otros.

Nació en Madrid en 1996 y se expandió abriendo nuevas oficinas en Barcelona, Sevilla, Valencia, Santiago de Chile, Lisboa, Roma, Milán, Buenos Aires, São Paulo, Monterrey, Bruselas y Bogotá.

Entre sus socios inversores se encuentran el fondo de inversión 3i, el grupo Landon y al fondo de inversión británico Hutton Collins, además de un grupo minoritario de accionistas.

En 2014 fue adquirida por la proveedora de servicios tecnológicos y consultora japonesa NTT Data, la sexta empresa de servicios IT del mundo que cuenta con más de 100.000 profesionales.

2.2. Mis prácticas profesionales

He realizado mis prácticas profesionales en el sector de Industria en las oficinas de Valencia.

Durante mis practicas tuve la oportunidad de incorporarme al equipo de desarrollo prácticamente desde el primer día, pudiendo participar en diferentes tipos de proyectos para diferentes compañías reconocidas a nivel nacional.

He utilizado diferentes tecnologías dependiendo del proyecto. Desde más actuales como Angular, para desarrollar aplicaciones del tipo SPA, a otras más antiguas como Visual Basic, para realizar labores de mantenimiento en sistemas desarrollados hace un buen puñado de años.

Esta etapa me ha permitido poner los conocimientos adquiridos durante la carrera en un contexto profesional.

3. Análisis y elicitación de requisitos

Esta fase nos ayuda a comprender el problema que la aplicación pretende resolver.

Aquí se procede a explicar de una manera breve las diferentes técnicas empleadas a la hora de obtener los requisitos y cuales se han utilizado para elaborar el programa.

Como alguien dijo alguna vez: si vas a hacer algo hazlo bien. A la hora de empezar a realizar un producto software es vital entender qué resultados busca el cliente y cómo podemos lograr nosotros cumplir sus necesidades.

Tenemos que intentar comprender muy bien qué se nos pide, esto puede parecer un tema baladí, pero de ello dependerá el éxito o fracaso de nuestro proyecto.

Los *stakeholders* son todas las personas relacionadas con el proyecto y de las cuales extraeremos los requisitos. Muchas veces, estos tienen puntos de vista diferentes que pueden o utilizan un vocabulario distinto que puede generar confusión. Debemos actuar como un intérprete intentando averiguar qué requisitos desean transmitir y validarlos para que queden lo más claro posible ya que es mejor encontrar los errores en esta fase que encontrarlos en fases posteriores donde costaría más dinero corregirlos, por ejemplo, durante el testeo del programa.

Es por eso por lo que necesitamos identificar, analizar y validar los requisitos antes de siquiera escribir una línea de código.

3.1. Técnicas de elicitación de requisitos

En el campo de la ingeniería del software contamos con diversas técnicas para identificar los requisitos.

Entrevistas

Esta técnica es de las más comunes y consiste en reunirse con cada *stakeholder* con la intención de intentar sacar todos los requisitos posibles y documentarlos. Se puede omitir información o provocar mal entendidos.

Análisis de documentos

Consiste en coger todos los documentos que se considere que pueden ser útiles para intentar sacar requisitos a través de estos.

Brainstorming

Es una técnica de grupo que tiene como objetivo generar el máximo número de ideas. Para ello, se designa a un “moderador” que apuntará las ideas de los *stakeholders*, pero no ejerce ningún control sobre el grupo. Las ideas no deben ser criticadas ni sacar valoraciones. Al final, se enumeran todas, se priorizan y se descartan las que se consideran “no realizables”.

JAD (Joint Application Development)

Esta técnica es similar a la anterior pero enfocada de una manera estructurada que tiene como objetivo agilizar el proceso de toma de requisitos.

Primero se entrevista a los stakeholders y se obtiene una especificación preliminar. Posteriormente se realiza una o varias reuniones en la que participan todos los stakeholders y que tiene como objetivo minimizar los errores y evitar confusiones. Finalmente se elabora un documento con los requisitos y se revisa junto con los stakeholders.

Durante el análisis de requisitos de la aplicación se utilizó la combinación estas cuatro técnicas para obtener los requisitos.

Se estudiaron los informes en papel que la aplicación quería representar y se realizaron entrevistas con cada stakeholder y varias reuniones todos en conjunto hasta obtener un documento con la especificación detallada de los requisitos.

Este documento también contenía el alcance del proyecto, los distintos roles que utilizarían la aplicación, anexos con información que se consideraba relevante para la construcción de los requisitos y una descripción general que explicaba el propósito de la aplicación.

La descripción general resumida dice lo siguiente:

La aplicación por desarrollar corresponde a un programa de escritorio destinado a llevar un seguimiento de tipo administrativo de las distintas operaciones realizadas por parte de los integrantes de los diferentes turnos en la planta. Estas operaciones se clasifican en diferentes "libros".

Cada turno estará formado obligatoriamente por los siguientes roles: un jefe de turno, un supervisor de sala, un auxiliar de control, un operador de reactor y un operador de turbina.

Opcionalmente se podrá contar con un encargado de exteriores, un encargado de residuos, un encargado de turbina, un encargado de reactor división 1 y un encargado de reactor división 2.

Todos los roles obligatorios podrán registrar, consultar y modificar maniobras asociadas a su usuario y rol. También podrán registrar, consultar y modificar fugas, inoperabilidades, pruebas y ordenes de funcionamiento asociadas con su usuario. El jefe de turno y los roles obligatorios en el turno podrán consultar y modificar maniobras, fugas, inoperabilidades, pruebas y ordenes de funcionamiento creadas por otros usuarios.

Las fugas e inoperabilidades se podrán guardar en formato PDF.

Se podrán establecer filtros de búsqueda para filtrar las maniobras, fugas, inoperabilidades, pruebas y ordenes de funcionamiento que coincidan con los criterios de búsqueda proporcionados.

Se realizarán validaciones para comprobar que los datos son correctos de acuerdo con los criterios del turno y en caso de no serlo se mostrarán avisos de los campos requeridos o incorrectos.

No solo se registrarán sucesos, sino que también se obtendrán ciertos parámetros útiles para la monitorización del correcto funcionamiento de la central que se podrán consultar en cualquier momento.

Por último, se podrá utilizar en caso de estar disponible, un acceso directo a un programa externo de SAP

3.2. Actores

Los actores son los stakeholders que usaran el sistema.

En la siguiente figura se puede observar los actores involucrados.

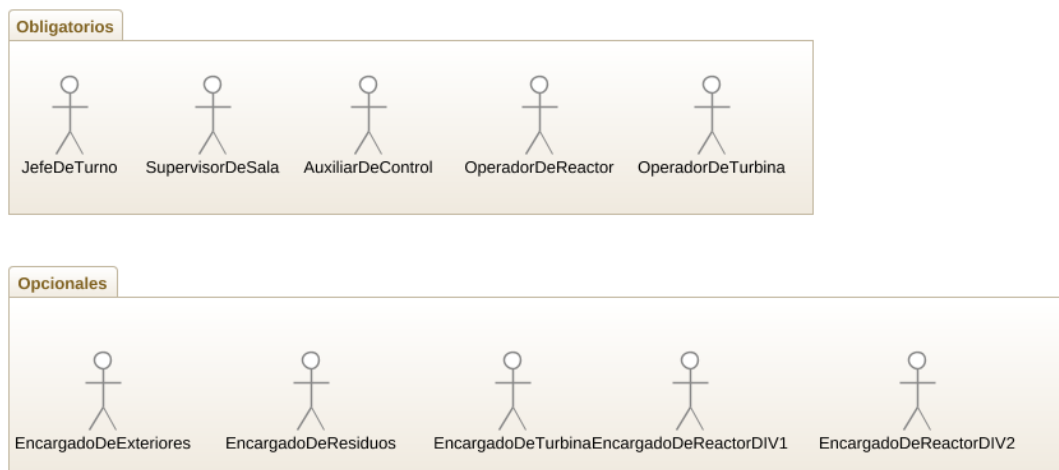


Figura 1: Actores de la aplicación

Se ha dividido los actores en dos grupos con el propósito de aportar claridad, los que son obligatorios y los que son opcionales dentro de un turno.

Los actores “Jefe De Turno” y “supervisor de sala” tienen roles de super administradores, pudiendo realizar todas las operaciones, como editar datos creados por otros usuarios, modificar la composición de los turnos o cerrar un turno.

El resto de los actores obligatorios, no pueden cambiar la composición de un turno ni cerrarlo.

Los actores opcionales tienen el fin de realizar consultas y solo pueden editar sus datos.

3.3. Descripción de los casos de uso

En base a los requisitos y roles obtenidos se han desarrollado los siguientes casos de uso con el objetivo de definir los requisitos.

Los casos de uso sirven para describir funcionalmente la aplicación, sin entrar en mucho detalle.

Se puede observar que la aplicación es en su mayoría operaciones CRUD. Por ese motivo, para no hacer muy "farragoso" este documento se ha decidido solo describir este tipo de casos de uso para fugas siendo los mismo para el resto de los apartados de la aplicación.

CU-01	Iniciar sesión
Objetivos	Iniciar una sesión con un usuario y contraseña para identificar al usuario
Descripción	El usuario inicia sesión mediante un formulario de inicio de sesión
Precondición	El usuario ha iniciado la aplicación
Secuencia normal	<ol style="list-style-type: none">1. El usuario inicia la aplicación2. El usuario introduce su usuario y contraseña3. Se selecciona el libro a utilizar4. El usuario presiona el botón "Entrar"
Postcondición	El usuario es llevado dentro de la aplicación con su sesión iniciada
Excepciones	En caso de ser incorrecto el usuario y la contraseña se mostrará un mensaje de error.
Comentarios	Ninguno

CU-02	Consultar parámetros central
Objetivos	Mostrar a el usuario los parámetros del funcionamiento de la central
Descripción	El usuario puede consultar los parámetros para comprobar el funcionamiento de la central
Precondición	El usuario ha iniciado sesión en la aplicación
Secuencia normal	<ol style="list-style-type: none">1. El usuario despliega el menú presionando el botón "Libro Operación"2. El usuario presiona la opción del menú "Inicio de turno"3. Se visualiza los parámetros de la central
Postcondición	Se muestran los datos de la central.
Excepciones	Ninguna
Comentarios	Ninguno



CU-03	Modificar composición turnos
Objetivos	El usuario puede modificar el personal correspondiente al turno
Descripción	El usuario puede cambiar la composición del turno asociando a un usuario con el rol necesario al turno
Precondición	El usuario ha iniciado sesión en la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Composición" 3. Se selecciona un rol 4. Se asigna el usuario con el rol correspondiente 5. Se presiona el botón guardar
Postcondición	Se notifica el correcto guardado de la información
Excepciones	En caso de no poder establecer conexión con la base de datos se muestra un error. Si no hay cambios pendientes se muestra un mensaje indicando que no hay cambios pendientes.
Comentarios	Ninguno

CU-04	Abrir programa SAP
Objetivos	Lanzar un programa externo.
Descripción	El usuario puede lanzar un programa externo desde dentro de la aplicación.
Precondición	El usuario ha iniciado sesión en la aplicación y el programa se encuentra instalado
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Abrir programa SAP"
Postcondición	Se abre el programa externo
Excepciones	En caso de no estar instalado o no encontrarse en la ruta esperada se muestra un mensaje de error.
Comentarios	Ninguno

CU-05	Crear fuga
Objetivos	El usuario crea una fuga

Descripción	El usuario crea una fuga y esta queda guardada en la base de datos
Precondición	El usuario ha iniciado sesión en la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Fuga" 3. Presiona la opción "Crear Fuga" 4. Rellena los datos necesarios correctamente 5. Presiona el botón "Guardar" <hr/> <ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Fuga" 3. Presiona la opción "Búsqueda fugas" 4. Presiona el botón "Nuevo" 5. Rellena los datos necesarios correctamente 6. Presiona el botón "Guardar"
Postcondición	Se muestran un mensaje de confirmación y se lleva al listado de fugas.
Excepciones	En caso de producirse un error en los datos se muestra un mensaje indicando el tipo de error
Comentarios	Ninguno

CU-06	Copiar fuga
Objetivos	Copiar una fuga ya creada
Descripción	El usuario puede copiar una fuga ya creada para realizar rápidamente una nueva fuga en base a otros datos
Precondición	El usuario ha iniciado sesión en la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Fuga" 3. Presiona la opción "Búsqueda fugas" 4. Busca o lista las fugas 5. Selecciona la fuga 6. Presiona el botón "Copiar" 7. Rellena los datos necesarios correctamente 8. Presiona el botón "Guardar"
Postcondición	Se muestran un mensaje de confirmación y se lleva al listado de fugas.

Excepciones	En caso de producirse un error en los datos se muestra un mensaje indicando el tipo de error
Comentarios	Ninguno

CU-07	Modificar fuga
Objetivos	Modificar los datos de una fuga ya creada
Descripción	El usuario puede modificar los datos de una fuga que ya está creada y guardar las modificaciones hechas si pertenece al mismo turno
Precondición	El usuario ha iniciado sesión en la aplicación y dispone de los permisos necesarios para modificar la fuga
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Fuga" 3. Presiona la opción "Búsqueda fugas" 4. Busca o lista las fugas 5. Selecciona la fuga y hace doble click sobre ella 6. Modifica los datos necesarios correctamente 7. Presiona el botón "Guardar"
Postcondición	Se muestran un mensaje de confirmación y se lleva al listado de fugas.
Excepciones	En caso de producirse un error en los datos se muestra un mensaje indicando el tipo de error
Comentarios	Ninguno

CU-08	Buscar fugas
Objetivos	Mostrar al usuario las fugas que coincidan con los parámetros introducidos.
Descripción	El usuario introduce los filtros que desea utilizar y el sistema le muestra la fuga que coincide con los filtros
Precondición	El usuario ha iniciado sesión en la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Fuga" 3. Presiona la opción "Búsqueda fugas" 4. Rellena los filtros que desea utilizar 5. Presiona el botón "Buscar"

Postcondición	Se muestran las fugas que coinciden con los filtros introducidos.
Excepciones	En caso de producirse un error en los datos se muestra un mensaje indicando el tipo de error.
Comentarios	Ninguno

CU-09	Exportar fugas en PDF
Objetivos	Obtener un listado en PDF con las fugas
Descripción	El usuario puede guardar las fugas en un archivo PDF
Precondición	El usuario ha iniciado sesión en la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario despliega el menú presionando el botón "Libro Operación" 2. El usuario presiona la opción del menú "Fuga" 3. Presiona la opción "Búsqueda fugas" 4. Presiona el botón "Exportar" 5. Se abre un cuadro de dialogo donde elige el nombre del archivo y donde desea guardarlo
Postcondición	Se muestra un mensaje de confirmación indicando que los datos se han guardado.
Excepciones	Se muestra un mensaje de error indicando el motivo del fallo
Comentarios	Ninguno

3.4. Diagrama de casos de uso

Los casos de uso y su relación con los actores quedan representados en el siguiente diagrama de casos de uso:

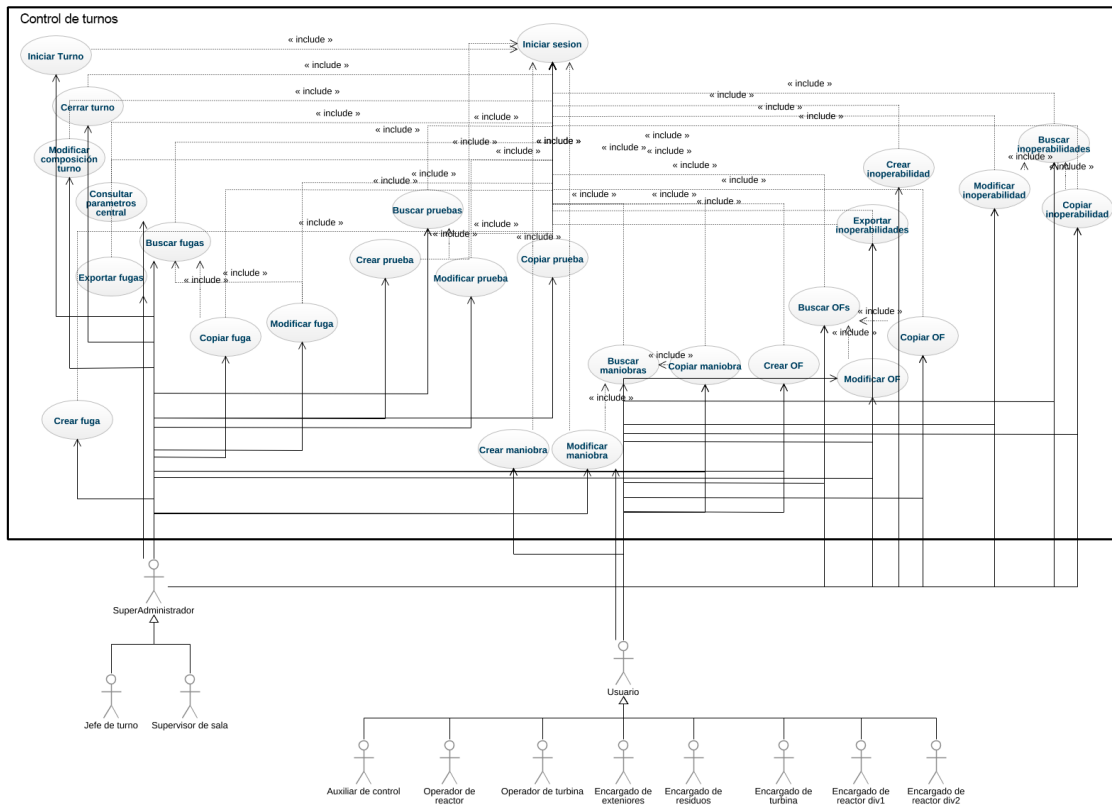


Figura 2: Diagrama de casos de uso

Este diagrama de casos de uso puede ser muy enrevesado a simple vista, ya que tiene muchas relaciones y muchos actores, por eso se ha creado dos actores auxiliares de los que hereda el resto.

Lo que representa este diagrama es que solo los actores *Jefe de Turno* y *Supervisor de Sala* pueden hacer todas las acciones. El resto de los roles solo pueden hacer las acciones CRUD de Maniobras, Inoperabilidades y Ordenes de Funcionamiento.

El siguiente paso que se tomó fue validar los requisitos.

3.4. Prototipo

Para conseguir validar los requisitos se realizó un prototipo mediante mockups. Así los *stakeholders* podían hacerse una idea de cómo sería la aplicación sin tener que

implementar nada. Es más fácil borrar un dibujo con una goma que implementar una funcionalidad no requerida o de una manera errónea.

A continuación, se incluye parte de los prototipos que se realizaron.

USUARIO CONTRASEÑA

0000001

LOGO

Libro

Libro de Operaciones

Entrar

Figura 3: Prototipo de pantalla de inicio de sesión

DATOS PERSONAL DE TURNO

Jefe de Turno	Supervisor de Sala	Encargado de EXTERIORES	Encargado de Reactor DIVISION I
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Operador de Reactor	Operador de Turbina	Encargado de RESIDUOS	Encargado de Reactor DIVISION II
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Auxiliar de Control		Encargado de Turbina	
<input type="text"/>		<input type="text"/>	

Figura 4: Prototipo de asignación de roles

Desarrollo de un sistema software para la gestión de las operaciones de una central nuclear

Inicio de turno

Turno <input type="text" value="A"/>	Fecha <input type="text" value="01/05/2018"/>	Hora <input type="text" value="00:00"/>	Act. L05 <input type="text" value=".05"/> pCi/cm3	Act. P38 <input type="text" value=".2"/> pCi/cm3
Sistemas Críticos <input type="text" value="Operables"/>	Condición de operación <input type="text" value="Recarga"/>	Alineamiento Válvulas Sistemas Críticos <input type="text" value="Correcto"/>		
Reactor				
Presión <input type="text" value="73.9"/> kg/cm3	Nivel <input type="text" value="95"/> cm	Pot. Térmica <input type="text" value="111.5"/> %	APRM <input type="text" value="111.5"/> %	
Generador		Contención		
Pot Activa <input type="text" value="1107"/> MWE	Pot. React <input type="text" value="-26"/> MVAR	Presión <input type="text" value="1107"/> mm HG	Temperatura <input type="text" value="-26"/> °C	
Pozo seco		Piscina Supr		
Presión <input type="text" value="732"/> mm Hg	Temperatura <input type="text" value="45.1"/> °C	Nivel <input type="text" value="582"/> cm	Temperatura <input type="text" value="27"/> °C	
<input type="button" value="Validar"/>				

Figura 5: Prototipo de inicio de turno

LOGO	Usuario
<p>LIBRO TURNO</p> <p>Inicio de Turno</p> <p>Composición</p> <p>Maniobras</p> <p>Búsqueda Maniobras</p> <p>Mis maniobras</p> <p>Crear maniobra</p> <p>Fugas</p> <p>Inoperabilidad</p> <p>Pruebas</p> <p>Ordenes</p> <p>Cierre de turno</p>	
LIBRO Operaciones	

Figura 6: Prototipo de menú

Listado maniobras

Filtros

Turno	Turno	Fecha fin	Estado	Fecha inicio	Mis maniobras
<input checked="" type="radio"/> Turno actual	<input type="text" value="A"/>	<input type="text" value="01/05/2018 10:00"/>	<input type="text" value="Validadas"/>	<input type="text" value="01/05/2018 10:00"/>	<input checked="" type="radio"/> Propias
M.P.L	Otros				
<input type="text" value="MPL"/>	<input type="text" value="Texto"/>		<input type="button" value="Buscar"/>		

Resultados

Figura 7: Prototipo de listado de maniobras

Creación maniobra

Turno	Fecha	Hora	Firma
<input type="text" value="A"/>	<input type="text" value="01/05/2018"/>	<input type="text" value="09:00 00"/>	<input type="text" value="Nombre Apellido Apellido"/>
MPL	Otro		
<input type="text" value="Otro"/>	<input type="text" value="Texto"/>		
Oficializada			
<input type="text" value="Si"/>			

Texto de la maniobra

--

Figura 8: Prototipo de creación de maniobra

Después de presentar el prototipo, el cliente indico que, en la ventana de pruebas, las pruebas debían de dividirse en varios tipos de pruebas independientes: pruebas, pruebas PS, pruebas no ET y requisitos.

Cada uno de estos tipos de pruebas contaría con sus propias opciones CRUD.

Se añadieron estos requisitos al documento funcional y se diseñó el siguiente prototipo:

Desarrollo de un sistema software para la gestión de las operaciones de una central nuclear

LOGO USUARIO
1/04/2018 08:00

LIBRO TURNO

Inicio de Turno
Composición
Maniobras
Fugas
Inoperabilidad
Modificaciones
Pruebas
Búsqueda
Crear
Ordenes
Utilidades
Listados
Configuración
Cierre de turno

Pruebas Pruebas PS Pruebas NO EST. Requisitos

Búsqueda Pruebas

Turno Fecha Inicio Fecha Fin Código Prueba/PS

Turno actual

Resultados

Figura 9: Prototipo búsqueda de prueba

Al finalizar esta etapa se obtiene un documento donde quedan especificados y validados los requisitos.

4. Diseño

En la etapa de diseño se muestra las decisiones tomadas para llegar a una solución que satisfaga el problema que el proyecto plantea y que cumpla con los requisitos anteriormente tomados en la etapa de análisis.

4.1. Arquitectura

La arquitectura es la forma en la que los distintos elementos de un sistema se integran entre si.

Estos elementos no tienen por qué ser necesariamente clases de programación orientada a objetos, también pueden referirse a componentes como una base de datos o como estará formada la red que componga nuestra aplicación.

Entre las ventajas de la arquitectura se encuentra que esta nos permite analizar si nuestro diseño puede cumplir los requisitos del sistema y reducir los riesgos a la hora de implementar el software.

La aplicación utilizara una arquitectura cliente-servidor.

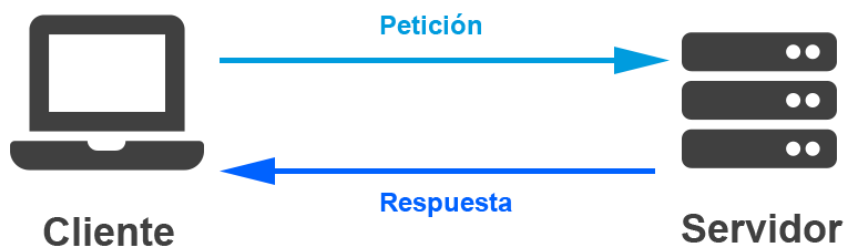


Figura 10: Arquitectura cliente-servidor

En este tipo de arquitectura los clientes demandan los recursos o datos que necesitan a los servidores, tal y como se representa en la *Figura 10*.

Los servidores por norma general son equipos que siempre se encuentran encendidos y conectados a la red esperando atender peticiones.

En este caso el programa se aloja en la máquina del cliente y estos a su vez comparten una base de datos que se encuentra en un servidor.

De esta manera muchos clientes pueden compartir datos y hacer solicitudes a la base de datos de manera concurrente.

Cuando un programa cliente hace una petición, como, por ejemplo, obtener la lista de maniobras, ésta es enviada a el servidor que contiene la base de datos y se encarga de responder con el listado de maniobras.

Se ha decidido utilizar una arquitectura de software por capas para el desarrollo de la aplicación.

Como su nombre indica, el software se divide en distintas capas, cada una de ellas independiente de la otra, su objetivo es segmentar el programa para reducir así el acoplamiento, haciendo el programa mucho más mantenible.

Normalmente esta segmentación se realiza en tres capas, como en la siguiente figura:

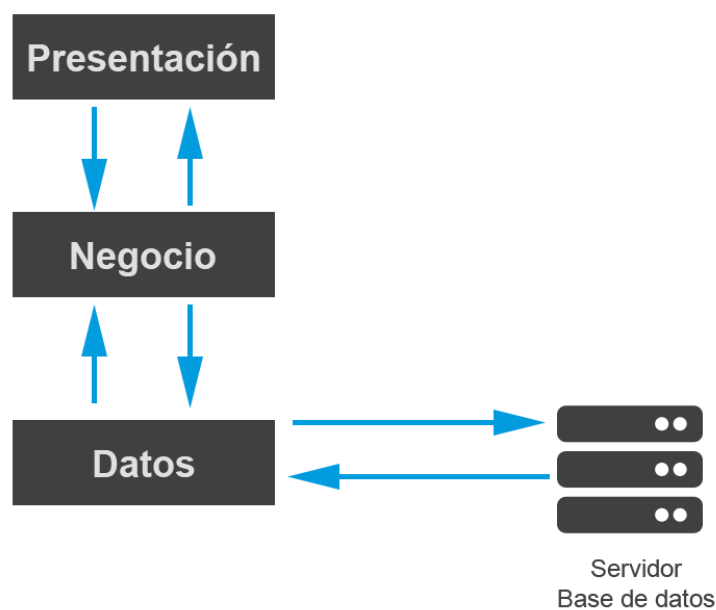


Figura 11: Arquitectura por capas

Capa de presentación: es la capa que el usuario ve, encargada de mostrar la interfaz con la que el usuario interactúa y capturar las acciones del usuario y se las comunica a la capa de lógica.

Capa de lógica o lógica de negocio: esta capa se encarga de realizar los procesos y operaciones a los datos para posteriormente transmitírselos o bien a la capa de datos o a la capa de presentación.

Capa de datos: puede ser la misma base de datos o una capa encargada de solicitar y enviar los datos a la base de datos.

Una de las ventajas de esta arquitectura es que permite a los programadores separar el trabajo, permitiéndoles así centrarse en desarrollar una sola parte y en caso de que sea

Desarrollo de un sistema software para la gestión de las operaciones de una central nuclear

necesario realizar un cambio solo es necesario modificar el nivel o niveles que afecta el cambio y no todo el programa.

Por otra parte, otra de las principales ventajas de esta arquitectura, es que torna a nuestro sistema escalable, ya que en cualquier momento podemos desacoplar las capas en varios servidores.

De esta manera si por ejemplo la capa de negocio fuese muy compleja podríamos separarla en un servidor aparte que dedique todos sus recursos a esta capa.



4.2. Diagrama de relaciones

El diagrama de relaciones muestra los componentes en los que se estructura el sistema y cómo estos están relacionados.

La siguiente figura representa el diagrama de relaciones obtenido para una primera versión de la aplicación.

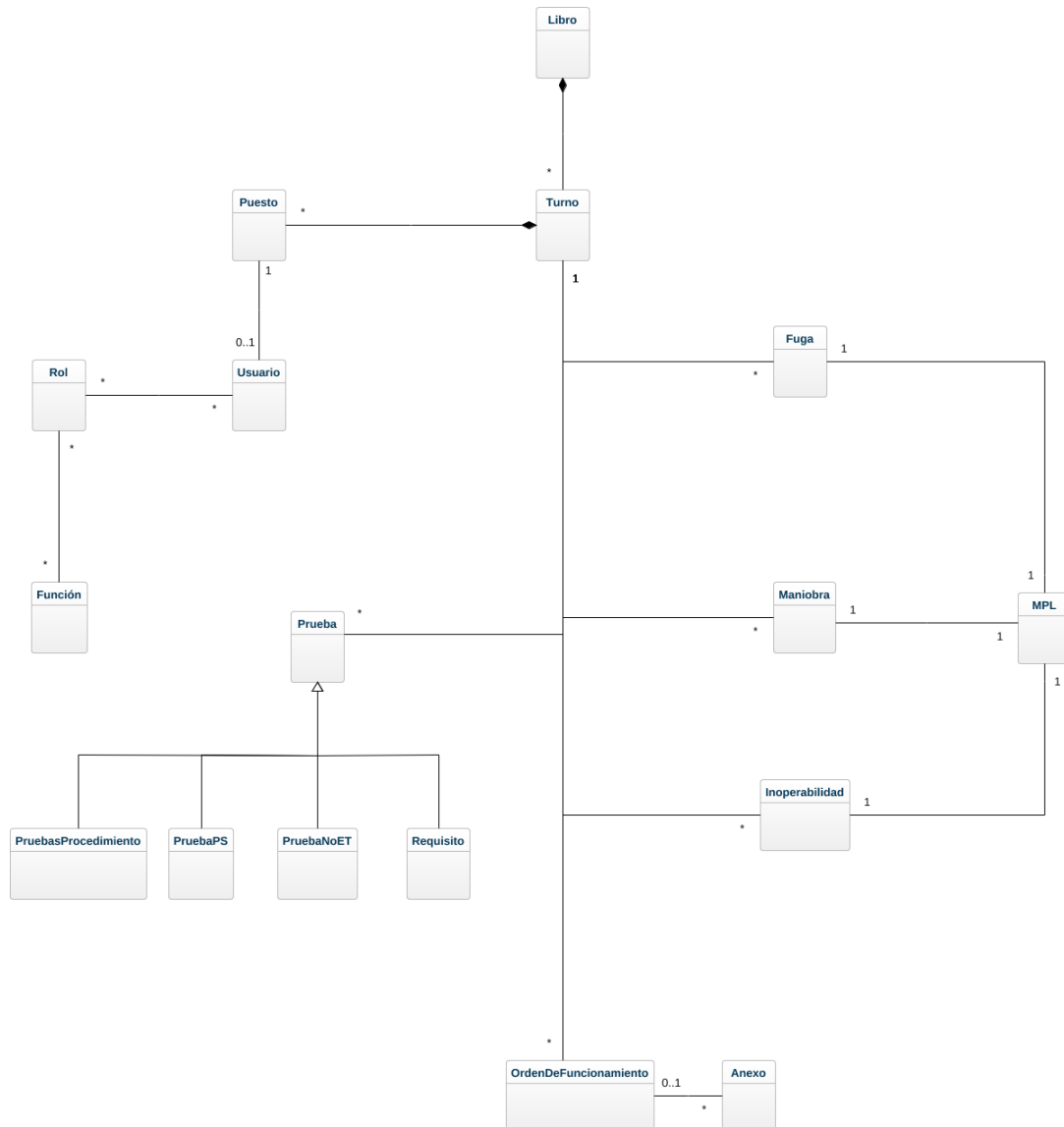


Figura 12: Diagrama de relaciones simplificado

Este diagrama de relaciones describe que cada libro esté compuesto de turnos, y cada uno de estos turnos se compone de puestos. Cada puesto puede estar ocupado por un usuario, dependiendo de si el puesto es opcional o no.

Los usuarios pueden tener varios roles asociados. Cada rol puede tener diferentes funciones asociadas a el.

A su vez, los turnos pueden estar compuestos de maniobras, inoperabilidades, ordenes de funcionamiento, fugas y pruebas.

Las inoperabilidades, fugas y maniobras tienen un MPL asociado.

Las pruebas se clasifican en: pruebas de procedimiento (dentro de la aplicación se conocerán como simplemente “pruebas”), pruebas PS, pruebas no ET y requisitos.

Las ordenes de funcionamiento podrán tener anexos adjuntos a ellas.

4.3. Esquema de base de datos

Se ha decidido utilizar para el desarrollo de este proyecto una base de datos relacional.

Las bases de datos relacionales se representan sus elementos o entidades en tablas, cada una de estas tablas está formada por filas y columnas, que contienen las propiedades y valores de los elementos.

También muestran las relaciones entre las distintas entidades, pudiendo obtener una visión más clara del sistema al comprender de una manera sencilla como se relacionan entre si.

Normalmente, cada fila es marcada con un identificador único, conocido como clave primaria.

Se realizó un esquema de la base de datos con el fin de visualizar el resultado que obtendríamos y como representaríamos la base de datos.

Para realizar el esquema se ha seguido el documento de especificación de requisitos y se ha decidió separar en distintas entidades los “objetos principales” con el fin de representar sus diferentes estados y composiciones.

Por ejemplo, un turno puede estar creado, pero no necesariamente tiene que estar en curso, para representar esto, se ha creado una tabla “TurnoIniciados” que indica si el turno se encuentra iniciado o cerrado.

También nos dimos cuenta de que una propiedad era compartida por varias entidades, los *MPL*, y se decidió hacer una tabla aparte y relacionarla con las entidades entre las que era compartida.

Por supuesto, no nos dimos cuenta de todas las posibles mejoras a añadir a nuestro esquema hasta que no nos encontramos con algunos problemas que se decidieron solucionar y mejorar por el camino.

El esquema que se ve en las figuras de a continuación es el resultado de esas mejoras.

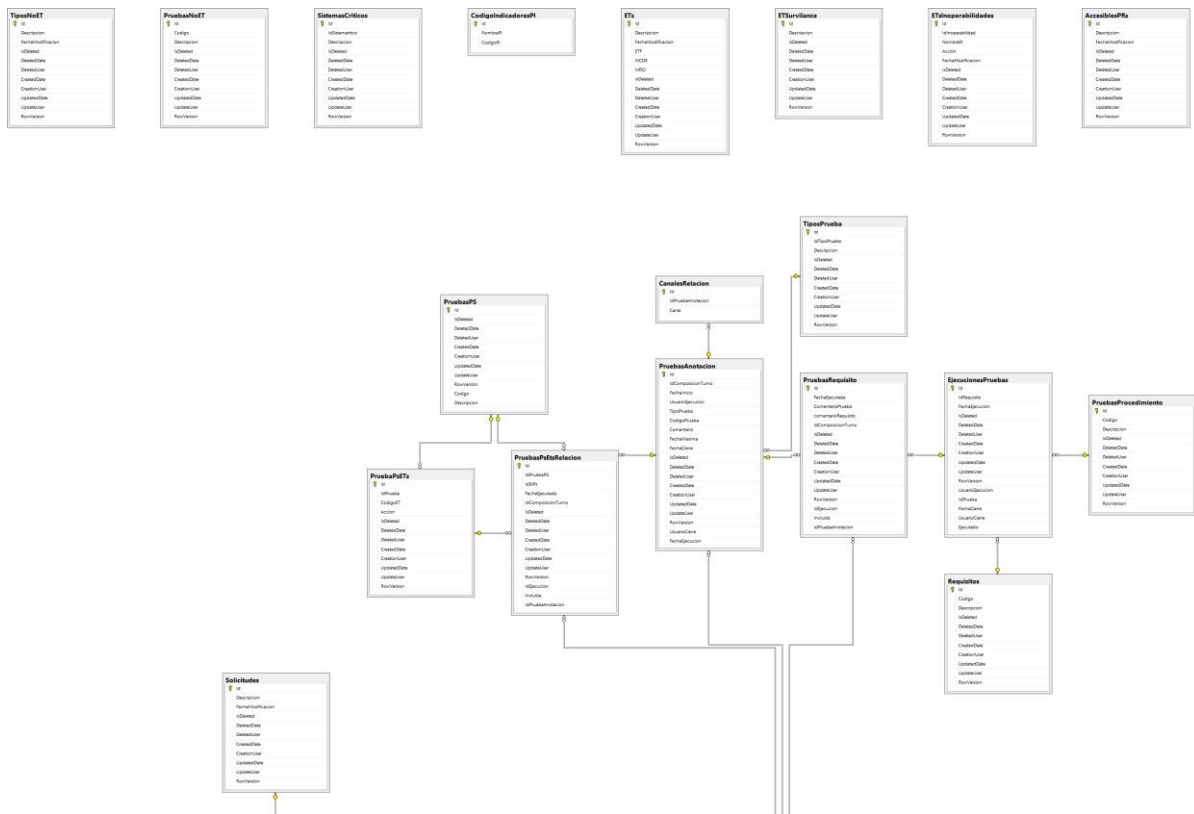


Figura 13: Esquema completo de la base de datos 1



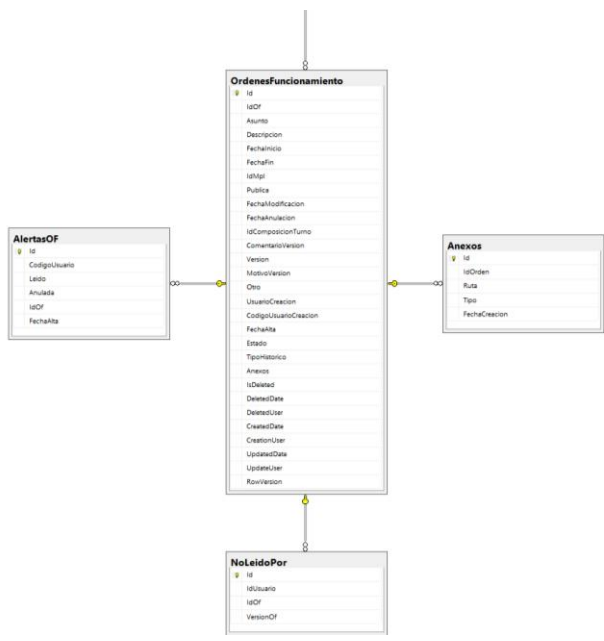


Figura 15: Esquema completo de la base de datos 3

5. Implementación

En este apartado se procede a describir las distintas tecnologías que se han usado para el desarrollo del proyecto, además de la metodología, el workflow y algunas prácticas empleadas durante su implementación.

5.1. Stack tecnológico

5.1.1. C# y .NET framework

Para la codificación se decidió utilizar el lenguaje orientado a objetos C#. C# es un lenguaje de programación compilado, fuertemente tipado y orientado a objetos creado por Microsoft, su sintaxis proviene de C++, y permite a los desarrolladores crear aplicaciones bajo el framework .NET. Un framework o marco de trabajo es un conjunto de bibliotecas, herramientas, clases, estándares y prácticas destinados a facilitar la tarea del desarrollo. .NET contiene un conjunto de componentes que facilitan el desarrollo, en los siguientes puntos se ven cuáles de estos se utilizaron para el desarrollo de la aplicación.

5.1.2. Entity Framework

Entity Framework es un ORM (Object Relational Mapping) que pertenece al framework .NET. Los ORM permiten “mapear” nuestro modelo de base de datos junto con sus relaciones a entidades en nuestro código. Entity framework se encargará registrar las modificaciones de nuestras entidades y persistirlas en la base de datos cuando nosotros se lo indiquemos. Una de las principales ventajas de utilizar un ORM es la abstracción del lenguaje de base de datos, pudiendo migrar así nuestra capa de datos a otro sistema de base de datos sin depender de su sintaxis. Entity framework, permite varias aproximaciones posibles a la hora de trabajar con una base de datos. Estas son las siguientes:

Database First

En esta aproximación se parte de que tenemos una base de datos ya creada e incluso que contiene datos.

En este caso Entity Framework se encargará de crear las entidades y relaciones en base a el esquema de nuestra base de datos.

La principal ventaja de este modo es que Entity Framework se encarga de realizar prácticamente todo el trabajo en automático, pero a cambio perdemos el control sobre el código, ya que este en su mayoría es autogenerado y puede acabar resultando innecesariamente pesado.

Model First

En esta aproximación creamos un modelo de base de datos mediante la herramienta de modelado de Visual Studio, luego Entity Framework se encarga de crear la base de datos y las entidades.



Las ventajas en este modo son similares a el anterior, Entity Framework realizara todo el trabajo, ya que genera tanto la base de datos como el código, pero a cambio tiene algo que considero una gran desventaja y es que los scripts para generar la base de datos no son incrementales sino completos, por lo tanto, por cada cambio que se realice se tendrá que regenerar la base de datos, perdiendo sus datos.

Hay maneras de solucionar esto como el uso de seeds, que son archivos que contienen los datos de nuestra base de datos y que cargamos una vez vaciamos la base de datos, pero no deja de ser añadir pasos extra al proceso.

Code First

En esta aproximación creamos nuestras entidades y sus relaciones mediante código y Entity Framework se encarga de crear la base de datos.

Este proceso suele resultar más costoso, ya que requiere codificar todas las clases y relaciones manualmente, y si nuestra base de datos es grande, puede resultar un proceso pesado. La gran ventaja es que tenemos el control total del código generado y cualquier cambio que se realice se puede sincronizar con nuestra base de datos.

Esta última fue la aproximación escogida para el desarrollo de la aplicación utilizada en la capa de datos.

En la siguiente figura se puede ver un ejemplo de los pasos realizados de cómo se implementa esta aproximación tomando la tabla MPL. Se define la entidad, utilizando la sintaxis especial de Entity Framework indicando sus propiedades y nombre de tabla.

```

using KOA.Data.Interception.Domain;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace KOA.Data.Entities.LdT
{
    [Table("MPLs")]
    public partial class MPLsEntity : KOARequiredEntity, IAuditableEntity
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
        public MPLsEntity()
        {
            Fugas = new HashSet<FugasEntity>();
            Inoperabilidades = new HashSet<InoperabilidadesEntity>();
            Maniobras = new HashSet<ManiobrasEntity>();
        }

        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int Id { get; set; }

        [StringLength(50)]
        public string Descripcion { get; set; }

        public DateTime FechaModificacion { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<FugasEntity> Fugas { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<InoperabilidadesEntity> Inoperabilidades { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<ManiobrasEntity> Maniobras { get; set; }
    }
}

```

Figura 16: Entidad MPL

Como en este caso la tabla tiene una relación de uno a muchos con las tablas Fugas, Inoperabilidades y Maniobras, se debe de incluir en la entidad una colección de estas entidades.

Y por último en el método OnModelCreating de nuestra clase encargada de crear la base de datos, que hereda de la clase DbContext añadimos las relaciones.

```

modelBuilder.Entity<MPLsEntity>()
    .HasMany(e => e.Fugas)
    .WithRequired(e => e.MPLs)
    .HasForeignKey(e => e.IdMpl)
    .WillCascadeOnDelete(false);

modelBuilder.Entity<MPLsEntity>()
    .HasMany(e => e.Inoperabilidades)
    .WithRequired(e => e.MPLs)
    .HasForeignKey(e => e.IdMpl)
    .WillCascadeOnDelete(false);

modelBuilder.Entity<MPLsEntity>()
    .HasMany(e => e.Maniobras)
    .WithOptional(e => e.MPLs)
    .HasForeignKey(e => e.IdMpl);

```

Figura 17: Relaciones MPL Entity Framework

Una buena ventaja que obtenemos al utilizar este enfoque es la posibilidad de utilizar un sistema de control de versiones para poder subir nuestros cambios a un repositorio y compartirlos con el resto de los programadores.

Posteriormente, a través de Visual Studio, se compara la versión de código, con la versión de base de datos y nos avisa si detecta algún cambio.

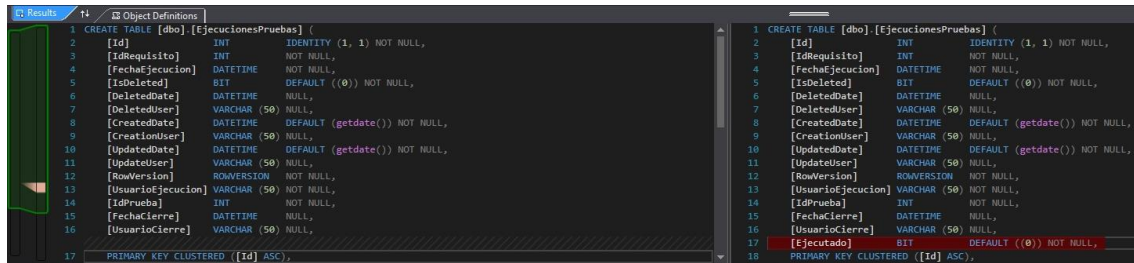


Figura 18: Comparador cambios base de datos

En la imagen se puede observar cómo Visual Studio detecta que nos falta el campo “Ejecutado” y podemos sincronizar el proyecto para contar con la última versión en nuestra base de datos.

5.1.3. LINQ

Es una extensión incluida en el framework .NET que permite realizar consultas tanto a colecciones de objetos como a una base de datos sin necesidad de utilizar SQL, mediante un conjunto de funciones predefinidas que devuelven tipos o colecciones de objetos ya definidos.

En el desarrollo de la aplicación se ha utilizado en la capa de lógica para crear una serie de servicios que serán los encargados de hacer las consultas a la base de datos.

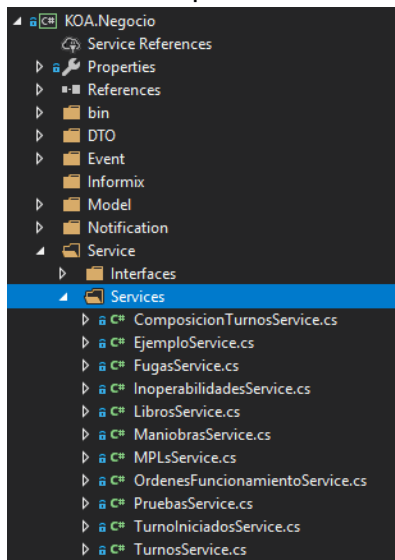


Figura 19: Servicios creados

Tomando como ejemplo el servicio de fugas se puede ver lo sencillo que resulta mediante LINQ obtener una colección con todas las entidades fuga ordenadas por su id.

```
public IQueryable<FugasEntity> GetAllFugas() => Context.Set<FugasEntity>().OrderBy(x => x.Id);
```

Figura 20: LINQ obtener todas las fugas

También resulta relativamente fácil implementar la función de búsqueda de fugas.

```
public IQueryable<FugasEntity> GetFugasFiltradas(BusquedaFugasDTO filtro)
{
    var query = Context.Set<FugasEntity>()
        .Where(x => (x.FechaFuga <= filtro.FechaFin || filtro.FechaFin == null) &&
            (x.FechaFuga >= filtro.FechaInicio || filtro.FechaInicio == null) &&
            (x.ComposicionTurnos.TurnoIniciados.IdTurno == filtro.IdTurno || filtro.IdTurno == null || filtro.IdTurno == Constantes.SIN_VALOR) &&
            (x.CodigoUsuarioCreacion == filtro.CodigoUsuarioActual || filtro.CodigoUsuarioActual == null) &&
            (x.ComposicionTurnos.TurnoIniciados.Id == filtro.IdTurnoActual || filtro.IdTurnoActual == null));
}
```

Figura 21: LINQ búsqueda fugas

Esta función recibe un DTO (Data Transfer Object), que no es más que un objeto que solo contiene las propiedades y tiene como objetivo transmitir la información de una capa a otra.

En este caso este objeto recoge los datos de la capa de presentación para traerlos a la capa de lógica.

Esta función lo utiliza para realizar las comparaciones en la cláusula *Where* y así saber que filtros utilizar dependiendo de si sus propiedades contienen datos o no y devolviendo así una colección de objetos que cumplan los filtros.

5.1.4. WPF

Es una tecnología de .NET utilizada para la creación de interfaces que utiliza el lenguaje de marcado XAML, basado en XML. Permite diseñar las interfaces de una manera similar a una aplicación web.

Se utilizó para el desarrollo de la interfaz, en la capa de presentación.

Una de sus características más importantes es el data binding que permite, utilizando propiedades de XAML, vincular los componentes visuales de la interfaz de usuario con las variables que contendrán los datos.

Esto se puede ver en las figuras de a continuación tomando como ejemplo el campo "Fecha Inicio".

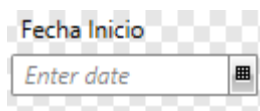


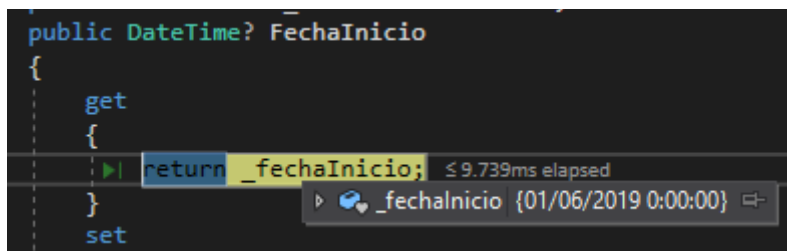
Figura 22: Fecha Inicio Vista

```
<telerik:RadDateTimePicker
  x:Name="radDateTimeFechaInicio"
  Grid.Column="4"
  Grid.Row="2"
  SelectedValue="{Binding FechaInicio}"
  IsEnabled="{Binding EnabledFieldsTurno}" >
</telerik:RadDateTimePicker>
```

Figura 23: Fecha Inicio WPF

La propiedad SelectedValue se establece al nombre de la variable donde se almacenará el dato seleccionado.

Suponiendo que seleccionamos la fecha 01/06/2019, un evento escucha cuando sucede un cambio y notifica a la propiedad del cambio, estableciendo así el nuevo valor.



```
public DateTime? FechaInicio
{
  get
  {
    return _fechaInicio;
  }
  set

```

The screenshot shows a debugger window with a tooltip displaying the return value of the `FechaInicio` property: `_fechaInicio {01/06/2019 0:00:00}`. The tooltip also indicates that 9.739ms elapsed for the return statement.

Figura 24: Depurado Fecha Inicio

5.1.5. KOA FRAMEWORK

KOA es un framework de desarrollo creado por la empresa donde se realizó las practicas tras años de diseño, ingeniería y evolución. Este consta de un conjunto de patrones de diseño, tecnologías y librerías.

Utiliza un modelo de persistencia donde a las entidades se les asigna un nombre que representa conceptos reales del dominio de la aplicación, por lo que un experto del dominio puede referirse a dichas entidades al hablar con un desarrollador.

Las entidades se crean como objetos planos (POCO, Plain Old Clr Object) donde solo se incluyen propiedades, estas no reaccionan a eventos, actuando como contenedores de datos.

Cuenta con borrado lógico o soft-delete, que consiste en establecer un flag a cada entidad que se actualiza dependiendo de si está ha sido borrada o no, evitando así el borrado de los datos de la base de datos.

Hace uso intensivo de la inyección de dependencias (que se explica en profundidad más adelante), mediante el contenedor de dependencias CastleWindsor.

5.2. Metodología

Una metodología es la manera en que diferentes procesos y técnicas son aplicadas con el objetivo de lograr un fin.

Existen diferentes metodologías a la hora de desarrollar software, cada una con sus ventajas e inconvenientes, queda fuera del alcance de este documento verlas en profundidad.

Hasta el momento podría decirse que se ha seguido una metodología tradicional en cascada, ya que los pasos se han producido de manera secuencial, uno tras otro.

Pero en la etapa de implementación se decidió seguir metodologías ágiles.

Las metodologías ágiles proponen seguir una serie de directrices recogidas en un manifiesto. Su objetivo principal es aumentar la satisfacción del cliente entregando versiones de software funcional en ciclos cortos de tiempo. Normalmente entre dos semanas y un mes.

El software va aumentando su funcionalidad de manera incremental con cada nueva versión que se entrega. De esta manera el cliente puede empezar a ver el software en funcionamiento desde una etapa temprana, reduciendo así la incertidumbre.

En estas metodologías se forman equipos pequeños y con roles variados, como, desarrolladores, diseñadores, gente de ventas y de negocio.

Se siguió la metodología ágil Scrum. Scrum propone un flujo de trabajo colaborativo e incremental, en el que se realizan entregas regulares al cliente de versiones funcionales del producto. Cada nueva versión del producto incorpora funcionalidades nuevas, mejoras y/o correcciones basadas en el *feedback* obtenido del usuario.

Scrum cuenta con los siguientes roles y artefactos:

Historias de usuario: son los requisitos expresados utilizando en un lenguaje común y fácilmente entendible.

Product Backlog: es un tablero donde se encuentran todas las historias de usuario priorizadas.

Sprint: es un conjunto de historias de usuario que se desarrollarán en un plazo tiempo determinado produciendo una nueva versión del producto

Tablero Kanban: es un tablero donde se organizan los sprint y nos permite visualizar el flujo de trabajo. Está formado por columnas, cada una de ellas corresponde al estado en el que la historia de usuario se encuentra. Un ejemplo del nombre de estas columnas podría ser ejemplo: por hacer, en desarrollo y completado.

Product owner: es una persona que cuenta con el conocimiento del producto que el usuario desea, se encarga de escribir las historias de usuario y las prioriza en producto backlog.

Scrum máster: Es una persona que se encarga de solucionar los posibles problemas que dificulten cumplir con el sprint y se encarga de que se cumpla la metodología

Desarrolladores: Son los encargados de implementar las historias de usuario, cumpliendo con los plazos de los Sprints y entregando las versiones del producto.



5.3. Workflow

El flujo de trabajo habitual del grupo de desarrollo consistía en implementar las historias de usuario que se encontraban en el tablero Kanban.

El tablero era como este:

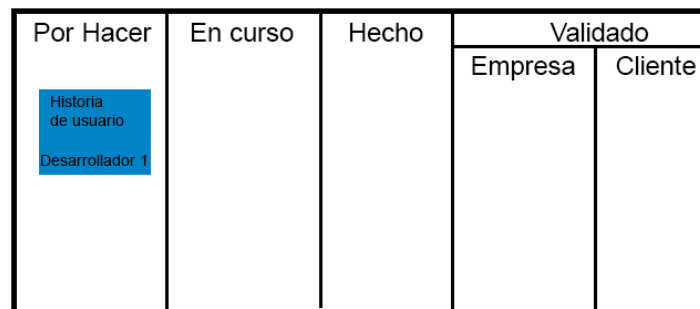


Figura 25: Kanban trabajo

Este tablero incluía una sección de validación que se dividía en dos, validado por la empresa y validado por el cliente. Donde se probaba la funcionalidad desarrollada y en caso de que uno de los dos detectase un fallo se creaba una nueva historia de usuario con el fallo detectado.

Estas tareas nos las repartíamos entre los desarrolladores, y cuando surgía algún problema con la implementación de alguna nos ayudábamos entre nosotros.

5.3.1. Control de versiones

El control de versiones es un sistema que detecta los cambios que se realizan a un archivo y permite registrarlos a lo largo del tiempo, pudiendo retroceder en los cambios realizados.

También permite trabajar en el software de manera colaborativa, de manera que dos desarrolladores pueden trabajar en versiones paralelas de la aplicación para posteriormente juntar sus modificaciones en una sola versión. Este proceso se conoce como *merge*.

Los programadores también pueden trabajar en ramas, que son copias de la aplicación en un momento determinado.

Las modificaciones son subidas a equipo o servidor conocido como repositorio.

Durante el desarrollo del software utilizábamos el sistema de control de versiones GIT junto con un cliente visual para trabajar colaborativamente.

Teníamos una estructura definida de ramas, estas se dividían en:

Master: esta rama contenía la versión de producción del software, por lo que aquí solo se subía el código una vez realizado el *testing* y después de ser validado por la empresa.

Develop: esta rama contenía una versión con un conjunto de funcionalidades completo a las que faltaba hacer testing y validar

Ramas divididas por funcionalidad: cada desarrollador abría una rama con el nombre de la funcionalidad que iba a implementar, así se podía saber de una manera sencilla quien estaba trabajando en que funcionalidad. Una vez completado el desarrollo de la funcionalidad la rama se *mergeaba* con la rama Develop.

5.4. Buenas prácticas empleadas durante el desarrollo

En este punto se explicarán las técnicas que se tuvieron en cuenta a la hora de realizar la implementación del software.

5.5. Código limpio

Puede parecer una obviedad decir que el código esté bien implementado es vital para el éxito de un proyecto software. Pero si se ha trabajado en la industria del software se sabe que esto pocas veces se llega a cumplir. Muchas veces debido a las ajustadas fechas límites con las que lidiamos los programadores nos impide darle las suficientes vueltas a la hora de encontrar una buena implementación para la solución y nos conformamos con que “funcione”.

Normalmente, los programadores noveles aterrizamos en proyectos que nos toca mantener. Código que han escrito otros programadores y que tiene una u otra calidad. Dependerá del programador que escribió ese código lo fácil o realmente difícil que se torne nuestro trabajo.

Podríamos considerar que un código está bien implementado cuando además de cumplir su propósito otro programador puede leerlo y llegar a comprenderlo sin demasiadas dudas. Para conseguir esto nos valemos de una serie de técnicas a la hora de programar.

En las siguientes líneas quiero transmitir los principios que se tuvieron en cuenta a la hora de realizar el desarrollo de este programa a través de ejemplos sacados de su código. En su gran mayoría, los principios aplicados son del libro Código Limpio de Robert C. Martin. Un libro que recomiendo encarecidamente si se desea ampliar este breve resumen que aquí se expone.

Como Robert C. Martin dice en su libro:

Puedo enseñarle la teoría de montar en bicicleta. De hecho, los conceptos matemáticos son muy sencillos. Gravedad, fricción, velocidad angular, centro de masa, etc., se pueden demostrar en menos de una página repleta de ecuaciones. [...] Pero la primera vez que se monte en una bici se caerá al suelo.



Es difícil llegar a hacer código limpio, ya que por mucho que se estudie sus conceptos, este solo se conseguirá con la práctica. Pero puestos a escribir código, intentemos hacerlo lo mejor posible.

Nombres con sentido

A la hora de programar los nombres se encuentran por todas partes, en variables, funciones, eventos, etc.

Por eso es importante que a la hora de escoger un nombre este sea lo más descriptivo y corto posible.

```
public DateTime? FechaInicio
```

Figura 26: Variable fecha inicio

Viendo esta variable queda claro que lo que contiene dentro es una fecha, y que esta es la fecha de inicio. Sin embargo, si a esta variable la llamásemos por ejemplo FI, puede que al principio tengamos claro a que nos referimos, pero más adelante cuando volviésemos a este código lo más seguro es que no recordemos su cometido.

Funciones

Las funciones deberían de estar lo más segregadas posible, encargándose cada una de ellas de una tarea que queda clara nada mas leer su nombre. Estas también deberían de ser de tamaño reducido y no contar con un excesivo número de argumentos.

```
public bool CanExportar()
{
    return this.WebService.GetPermisoFuncionNombreByUsuario(this.IdApplication, this.UserIdentity.Id, Constantes.PERMISO_CREAR_MANIOBRA);
}
```

Figura 27: Función CanExportar

El nombre CanExportar deja claro que esta función tiene como cometido comprobar si es posible realizar la acción de exportar devolviendo falso o verdadero. Para ello llama a un servicio que comprueba que el usuario tiene el permiso necesario.

Comentarios

Los comentarios solo se deberían utilizar con el propósito de arrojar aún más claridad a nuestro código.

```
// Comprobamos si existen maniobras sin publicar
var maniobrasSinPublicar = ManiobrasService.GetManiobrasTurnoActual(ComposicionTurnoActual.TurnoIniciados.Id).Where(x => x.Borrador);
if (maniobrasSinPublicar.Any())
{
    result = UtilsMessageBox.ShowMessage("Confirmación", "Existen Maniobras sin publicar en el turno, ¿está seguro que desea cerrar el turno actual?", MessageBoxButton.YesNo, MessageBoxImage.Question);
    if (result != MessageBoxResult.Yes)
    {
        return;
    }
}
```

Figura 28: Comentarios

En este fragmento de código se indica en el comentario que aquí es donde comprobamos si existen maniobras sin publicar y debajo podemos leer en el código que así es.

Los comentarios también pueden indicar tareas por hacer, son los conocidos como comentarios TODO

```
//TODO Añadir control para lanzar excepción en caso de que la Entidad seleccionada este incompleta o no se haya seleccionado con éxito
public class AutoCompleteBox : Telerik.Windows.Controls.RadAutoCompleteBox
```

Figura 29: Comentarios TODO

Regiones

En C# podemos hacer uso de la instrucción `#region` que nos permite expandir y contraer bloques de código pudiendo así organizar mejor una clase y centrarnos solo en la parte en la que estemos trabajando.

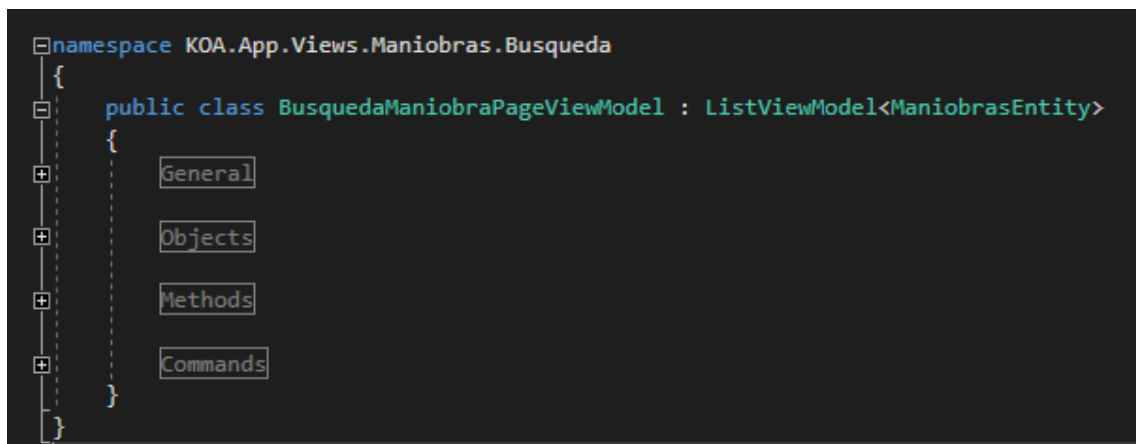


Figura 30: Regiones

En esta clase dividimos el código en varias regiones. Si por ejemplo quisiéramos añadir un nuevo comando a nuestra clase podríamos expandir la región `commands` y centrarnos en añadir nuestro nuevo comando.

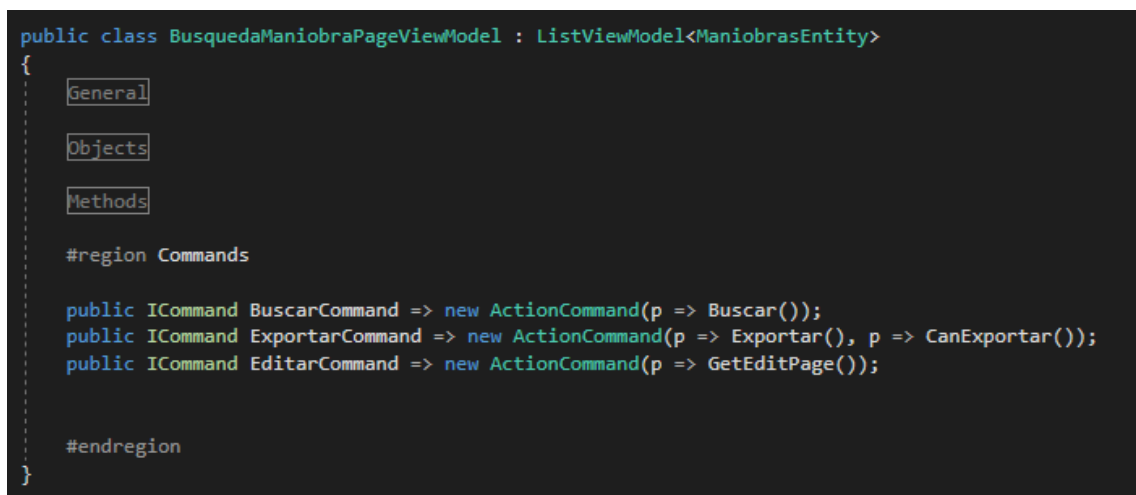


Figura 31: Región `Commands` expandida

5.6. Principios SOLID

Estos principios indican una serie de buenas prácticas a la hora de crear estructuras de datos y funciones, para que nuestro código sea reutilizable y tolere mejor los cambios a lo largo del tiempo que sin duda el cliente pedirá.

5.6.1. Principio de responsabilidad única

Este principio dice que nuestra clase solo debería de tener un cometido y no varios.

Por ejemplo, si tenemos una clase encargada de mostrar datos en pantalla, esta clase no debería de consultar datos a la base de datos ya que estamos mezclando dos cometidos diferentes.

Lo ideal sería separar la clase en dos teniendo una clase encargada de hacer las llamadas a datos en la capa de datos y otra que se encargue de presentarlos en la capa de presentación.

5.6.2. Principio abierto/cerrado

El concepto de este principio es simple, dice que las clases que escribimos deberían de estar abiertas para la extensión, pero cerradas para la modificación. Debería de ser fácil añadir una nueva funcionalidad a una clase sin tener que modificar su código original. Solamente extendiéndola.

Esto lo podemos lograr haciendo uso del polimorfismo.

5.6.3. Principio de sustitución de Liskov

Este principio indica que las clases que heredan de otra clase deben de seguir comportándose como su padre. Se podría sustituir una por otra.

Esto así escrito puede sonar un poco confuso, pero lo único que quiere decir es que si por ejemplo la clase padre tiene unos métodos que cuentan con una serie de restricciones la subclase hija también deberá contar con esos métodos y restricciones.

5.6.4. Principio de segregación de interfaz

Este principio nos dice que debemos de separar las interfaces de nuestro programa tanto como sea posible para así utilizar solo las que necesitemos en cada clase.

5.6.5. Principio de inversión de dependencias

El concepto de este principio es algo complejo, este dice que debemos eliminar las instanciaciones de nuestras clases para que no existan dependencias. Cuando una clase necesite hacer uso de un objeto, otra clase será la encargada de construir ese objeto y pasárselo.

Estos son los principios que se han tenido en cuenta a la hora de implementar el código, y se han aplicado en la medida de lo posible adaptándolos a las necesidades del proyecto.

5.7. Patrones de diseño

Los patrones de diseño son soluciones a problemas recurrentes que ocurren durante el diseño de software orientado a objetos.

De esta forma cuando nos encontramos con uno de estos problemas basta con coger uno de estos patrones y adaptarlo a nuestra solución así no tenemos que reinventar la rueda.

5.7.1. Patrón inyección de dependencias

El patrón de inyección de dependencias es una solución que utiliza el principio de inversión de dependencias.

Para explicarlo me apoyare en un ejemplo.

Supongamos que tenemos, una vista que consume dos servicios para mostrar sus datos como se representa en la *figura 32*.

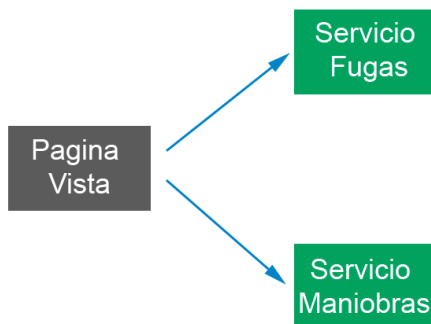


Figura 32: inyección de dependencias 1

Tenemos el siguiente código:

```
class PaginaVista
{
    private ServicioFugas _servicioFugas;
    private ServicioManiobras _servicioManiobras;

    public PaginaVista() {
        this._servicioFugas = new ServicioFugas();
        this._servicioManiobras = new ServicioManiobras();
    }

    public void ImprimirDatos() {
        this._servicioFugas.Imprimir();
        this._servicioManiobras.Imprimir();
    }
}
```

Figura 33: Inyección de Dependencias 1 código

Como vemos la página vista tiene dependencias con los servicios, creando instancias de ellos.

Ahora aplicando el patrón de inyección de dependencias veamos como quedaría.

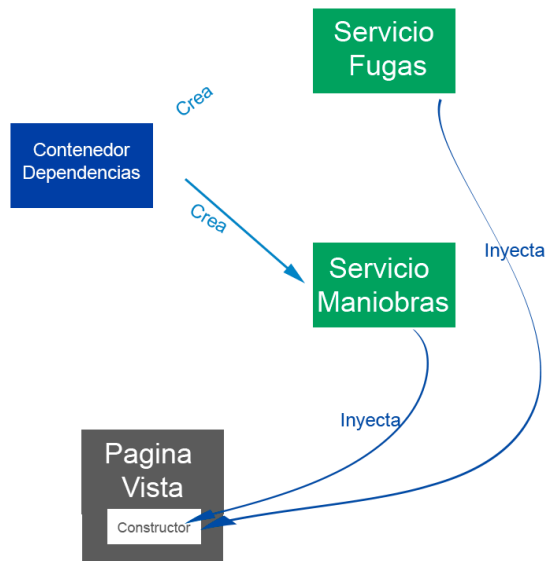


Figura 34: Inyección de Dependencias 2

Ahora hemos metido una nueva clase, ContenedorDependencias esta clase será la encargada de inyectar las dependencias al constructor de la clase PaginaVista.

Veamos el código

```
class PaginaVista
{
    private ServicioFugas __servicioFugas;
    private ServicioManiobras __servicioManiobras;

    public PaginaVista(ServicioFugas servicioFugas, ServicioManiobras servicioManiobras) {
        this.__servicioFugas = servicioFugas;
        this.__servicioManiobras = servicioManiobras;
    }

    public void ImprimirDatos() {
        this.__servicioFugas.Imprimir();
        this.__servicioManiobras.Imprimir();
    }
}
```

Figura 35 Inyección de dependencias 2 PaginaVista

```
class ContenedorDependencias
{
    static void Main(string[] args)
    {
        PaginaVista pagina = new PaginaVista(new ServicioFugas(), new ServicioManiobras());
        pagina.ImprimirDatos();
        Console.Read();
    }
}
```

Figura 36: Inyección de dependencias 2 ContenedorDependencias

El resultado puede parecer el mismo, pero ahora no es la clase `PaginaVista` la encargada de crear las instancias de las clases que utiliza si no que estas son inyectadas por la clase `ContenedorDependencias` inyectamos las dependencias desde esta clase.

Esto nos permite hacer extensible nuestro programa, por ejemplo, podríamos crear un servicio que extienda cualquiera de los que utilizamos y sobrescribir la funcionalidad de sus métodos.

Otra implementación posible sería aplicar el principio de sustitución de Liskov.

Como se ha explicado anteriormente, el objetivo de este principio es permitir que las clases que heredan de una clase padre se puedan intercambiar una por la otra sin afectar a como se comportan.

Una forma de obtener este comportamiento sería hacer que los servicios implementen los mismos métodos de una misma interfaz obteniendo un código como el siguiente.

```
public interface IServicio
{
    void Imprimir();
}

class ServicioFugas : IServicio
{
    public void Imprimir()
    {
        Console.WriteLine("Hola soy el servicio de Fugas");
    }
}

class ServicioManiobras : IServicio
{
    public void Imprimir()
    {
        Console.WriteLine("Hola soy el servicio de Maniobras");
    }
}

class ServicioInoperabilidades : IServicio
{
    public void Imprimir()
    {
        Console.WriteLine("Soy el servicio de Inoperabilidades");
    }
}
```

Figura 37: Interfaz y clases

```
class PaginaVista
{
    private IServicio _servicioFugas;
    private IServicio _servicioManiobras;

    public PaginaVista(IServicio servicioFugas, IServicio servicioManiobras) {
        this._servicioFugas = servicioFugas;
        this._servicioManiobras = servicioManiobras;
    }

    public void ImprimirDatos() {
        this._servicioFugas.Imprimir();
        this._servicioManiobras.Imprimir();
    }
}
```

Figura 38: Clase PaginaVista

De esta manera podríamos crear nuevos servicios e intercambiarlos sin problema, obteniendo un código fácil de extender.

Este patrón es utilizado por el framework KOA, haciendo uso del contenedor de inversión de dependencias *CastleWindsor*, que se encarga de inyectar las dependencias que las clases necesiten.

Para usarlo tenemos una clase *DependenciesInstaller* que hereda de la interfaz *IWindsorInstaller* e implementa el método donde se registran las dependencias a instalar.

Aquí se puede ver como lo usamos para registrar los ViewModel de las páginas de la aplicación.

```
using KOA.App.Views.OrdenesFuncionamiento.DetalleOrdenesFuncionamiento;
using KOA.App.Views.Pruebas.Busqueda;
using KOA.App.Views.Pruebas.DetallePruebas;
using KOA.Common.Windows;
using KOA.Data;
using KOA.Data.CastleWindsor;
using KOA.Windows;

namespace KOA.App
{
    public class DependenciesInstaller : IWindsorInstaller
    {
        public void Install(IWindsorContainer container, IConfigurationStore store)
        {
            #region Pages

            container.Register(Component.For<NavigationContainer>().ImplementedBy<NavigationContainer>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<MainPageViewModel>().ImplementedBy<MainPageViewModel>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<HelpContent>().ImplementedBy<HelpContent>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<BusquedaManiobraPageViewModel>().ImplementedBy<BusquedaManiobraPageViewModel>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<HelpPageViewModel>().ImplementedBy<HelpPageViewModel>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<SearchDialogWindowViewModel>().ImplementedBy<SearchDialogWindowViewModel>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<ColumnsMultiSelectionWindowViewModel>().ImplementedBy<ColumnsMultiSelectionWindowViewModel>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<ListWindowViewModel>().ImplementedBy<ListWindowViewModel>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<InformationDialogWindowViewModel>().ImplementedBy<InformationDialogWindowViewModel>().LifestyleScoped<CustomScopeAccessor>());
            container.Register(Component.For<PantallaEjemploPageViewModel>().ImplementedBy<PantallaEjemploPageViewModel>().LifestyleScoped<CustomScopeAccessor>());

            #endregion
        }
    }
}
```

Figura 39: Instalador de dependencias

5.8. Observaciones

Me gustaría comentar una serie de observaciones sobre el software aquí desarrollado que considero curiosas.

Como se ve en el gráfico de a continuación, a pesar de que el programa ha sido escrito en C#, en su mayoría está compuesto de XML. Esto es debido a que WPF hace un uso intensivo de este lenguaje para realizar la interfaz, aumentando de esta manera el peso del programa.

El resto de los lenguajes que aparecen en la tabla proceden de librerías.

3367953 Líneas de código

3127927 Líneas de código en .xml archivos (92,9 %)
106208 Líneas de código en .cs archivos (3,2 %)
96885 Líneas de código en .js archivos (2,9 %)
31998 Líneas de código en .css archivos (1,0 %)
2767 Líneas de código en .wsdl archivos (0,1 %)
1676 Líneas de código en .sql archivos (0,0 %)
360 Líneas de código en .xsd archivos (0,0 %)
132 Líneas de código en .html archivos (0,0 %)

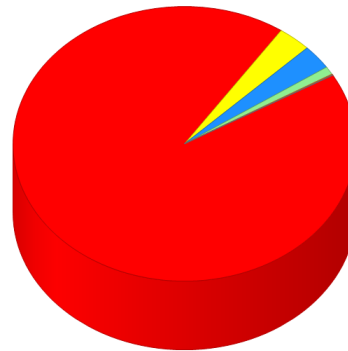


Gráfico 1: Líneas de código por lenguaje

Otra curiosidad a tener en cuenta es que los comentarios escritos son un 2% del código, esto puede parecer poco, pero si lo comparamos con el libro Don Quijote de la Mancha escrito por Cervantes, nos encontramos que este tiene aproximadamente unas veinticinco mil líneas.

3367953 Líneas de código

81549 Líneas en blanco (2,3 %)
72165 Líneas comentadas (2,0 %)
3367953 Líneas de código (95,5 %)
3328 Líneas en archivos de diseño (0,1 %)

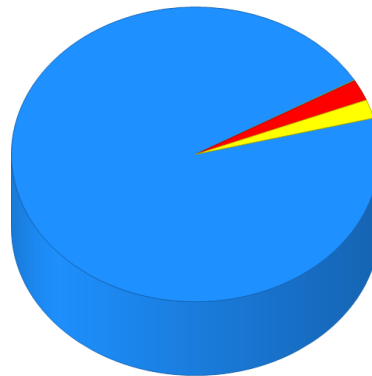


Gráfico 2: Líneas de código por tipo

Leer los comentarios de este programa equivaldría a leer casi tres veces el libro de Cervantes ya que nos encontramos con algo más de setenta y dos mil líneas de comentarios.

6. Resultado

Por último, en esta sección se va a mostrar unas cuantas capturas de pantalla de la aplicación desarrollada en funcionamiento durante el proceso de crear una maniobra.

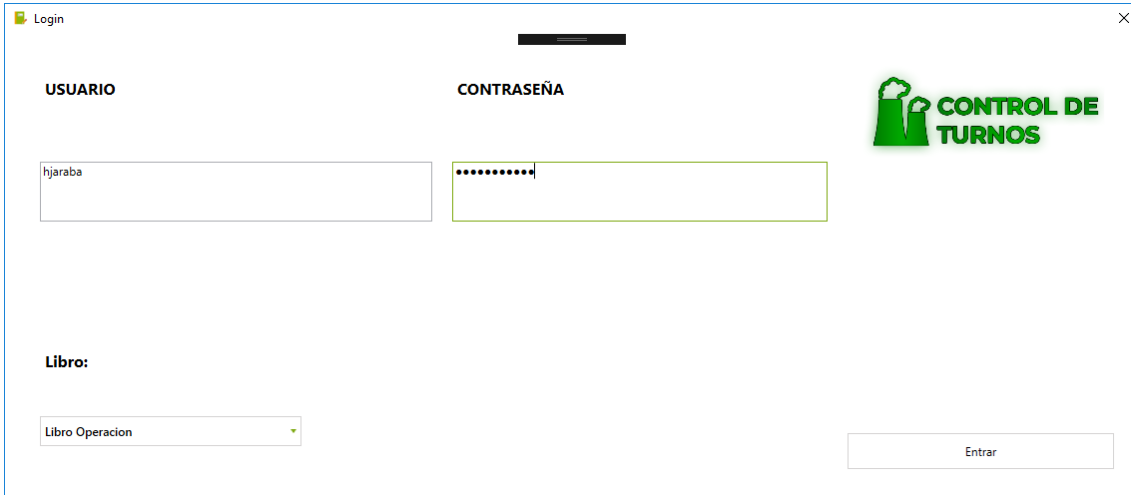


Figura 40: Pantalla login

Figura 40: Pantalla login

Esta es la pantalla de login, que valida los datos del usuario permitiéndole iniciar sesión en la aplicación.

Una vez el usuario accede, si tiene asignado el rol de Jefe de Turno y no se encuentra un turno iniciado, se le pide asignar el personal del nuevo turno.

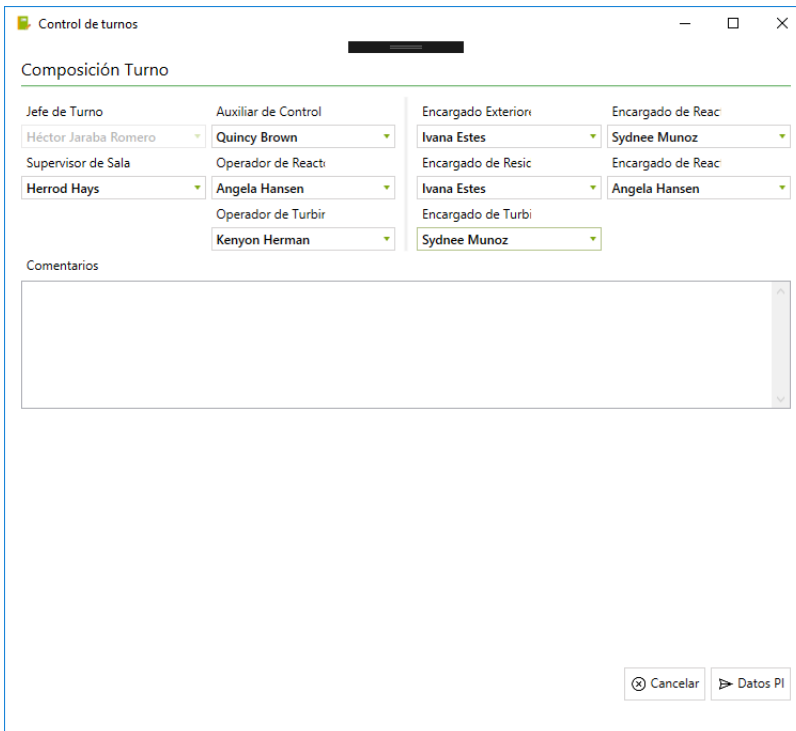


Figura 41: Pantalla composición turno

Figura 41: Pantalla composición turno

Desarrollo de un sistema software para la gestión de las operaciones de una central nuclear

Una vez se ha asignado al personal del turno se muestra una pantalla con los datos de la central y el turno que va a iniciar.

The screenshot shows the 'Control de turnos' application window. The title bar indicates the user is logged in as 'Héctor Jaraba Romero (hjaraba)'. The main content area is titled 'Inicio Turno' and contains several data entry fields and a list of critical systems.

Turno	Fecha	Act. L05	Act. P38
C	11/06/2019	Actividad L05	999,9

Below the main data fields, there are sections for 'Alineamiento Válvulas Sistemas Críticos', 'Alarmas más significativas', and a 'Recarga' dropdown menu. The 'Recarga' menu is open, showing a list of critical systems with checkboxes:

- Sistema Crítico 2
- Sistema Crítico 1
- Sistema Crítico 3
- 4654654

The bottom section of the screen displays various parameters for the Reactor, Generador, Contén., Pozo Seco, and Piscina, each with a numerical value and unit.

Reactor				Generador			
Presión	Nivel	Pot. Térr	APRM	Pot. Acti	Pot. Rea	MWE	MVAR
999,90	kg/cm3	999,90	cm	999,90	%	999,90	%

Contén.		Pozo Seco		Piscina			
Presión	Temp.	Presión	Temp.	Nivel	Temp.		
999,90	mm HG	999,90	°C	999,90	mm HG	999,90	°C

At the bottom of the screen, there are two buttons: 'Importar Datos PI' and 'Iniciar Turno'.

Figura 42: Pantalla inicio turno

Una vez iniciado el turno podemos acceder a el menú de la aplicación, en este caso como entramos con el rol de *Jefe de Turno* podemos acceder a todas las opciones de la aplicación.

Vamos a crear una maniobra. Para ello, desde el panel Alta Maniobra introducimos los datos de la nueva maniobra.

The screenshot shows the 'Control de turnos' application window with the 'Alta Maniobra' screen. The user is logged in as 'Héctor Jaraba Romero (hjaraba)'. The screen is divided into a left sidebar menu and a main content area.

The left sidebar menu is titled 'Libro Operación' and contains the following items:

- Inicio de turno
- Composición
- Maniobras
 - Maniobras Libro Operación
 - Mis maniobras
 - Crear Maniobra
 - Maniobras Exteriores
 - Maniobras Residuos
 - Maniobras Turbina
 - Maniobras Reactor Div I
 - Maniobras Reactor Div II
- Fugas
 - Búsqueda Fugas
 - Crear Fuga
- Inoperabilidad
 - Búsqueda Inoperabilidades
 - Crear Inoperabilidad
- Pruebas
 - Búsqueda
 - Crear
- Órdenes
 - Búsqueda Órdenes de funcionamiento
 - Crear Orden de funcionamiento
 - Mis Alertas
 - Abrir programa SAP
 - Cierre de turno

The main content area is titled 'Alta Maniobra' and contains a form with the following fields:

Turno	Fecha	Hora	Firma
C	11/06/2019	14:52	Héctor Jaraba Romero <input checked="" type="checkbox"/> Oficial

Below the main data fields, there are sections for 'M.P.L.' (set to 'Otro') and 'Texto de la maniobra' (containing the text 'Ajuste isotopo H-123').

At the bottom of the screen, there are two buttons: 'Guardar' and 'Guardar Borrador'.

Figura 43: Pantalla control de turnos

La maniobra puede ser guardada como un borrador, de esta manera solo será visible para el usuario que la creo y podrá elegir cuando publicarla desde el menú Mis Maniobras. *En este caso se guarda como borrador.*

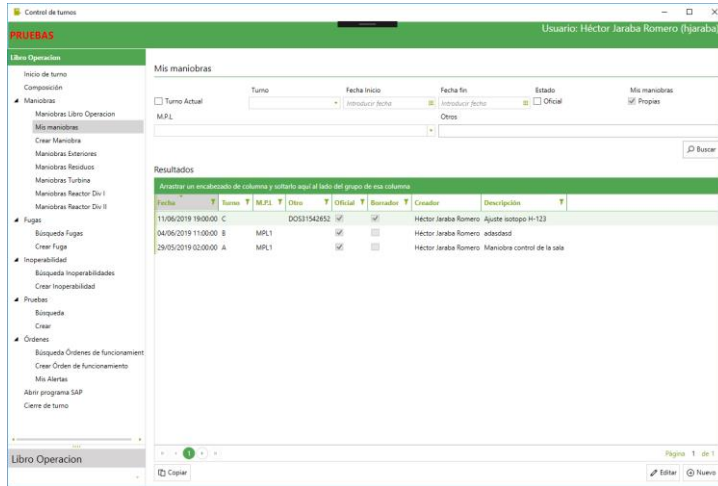


Figura 44: Pantalla mis maniobras

Desde el listado general donde se muestran todas las maniobras para todos los usuarios podemos ver que la maniobra no aparece.

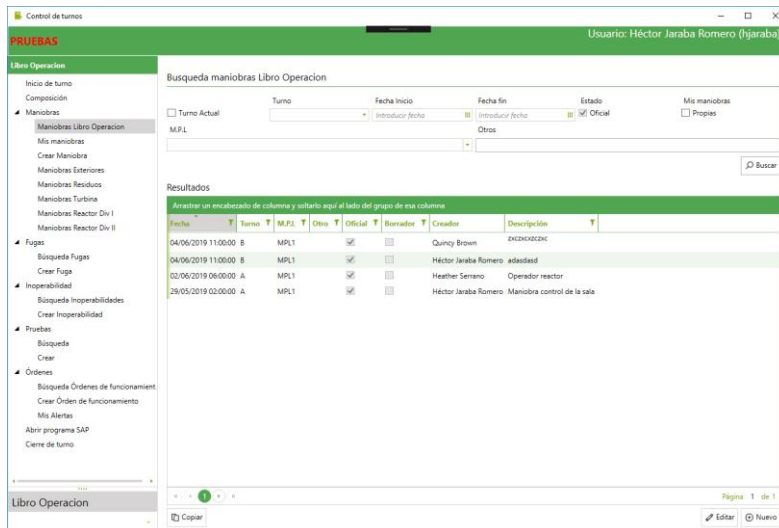


Figura 45: Pantalla listado maniobras

Podemos oficializar la maniobra, accediendo a ella y pulsando el botón publicar. Así la maniobra queda visible para el resto de los usuarios.

Desarrollo de un sistema software para la gestión de las operaciones de una central nuclear

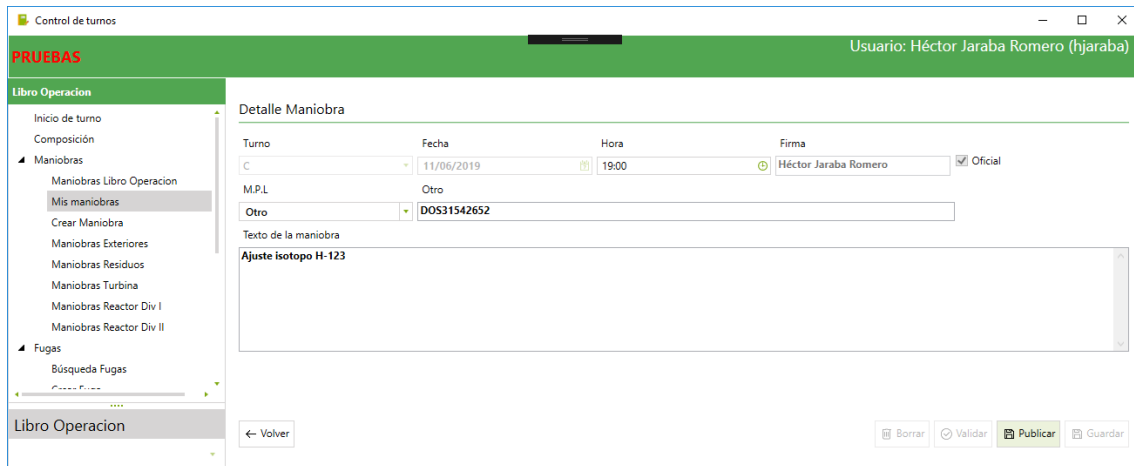


Figura 46: Pantalla detalle maniobra

Si la buscamos ahora en el listado, podemos ver que aparece. Además, se han configurado unos filtros de búsqueda para que solo se muestre la maniobra que se ha creado.

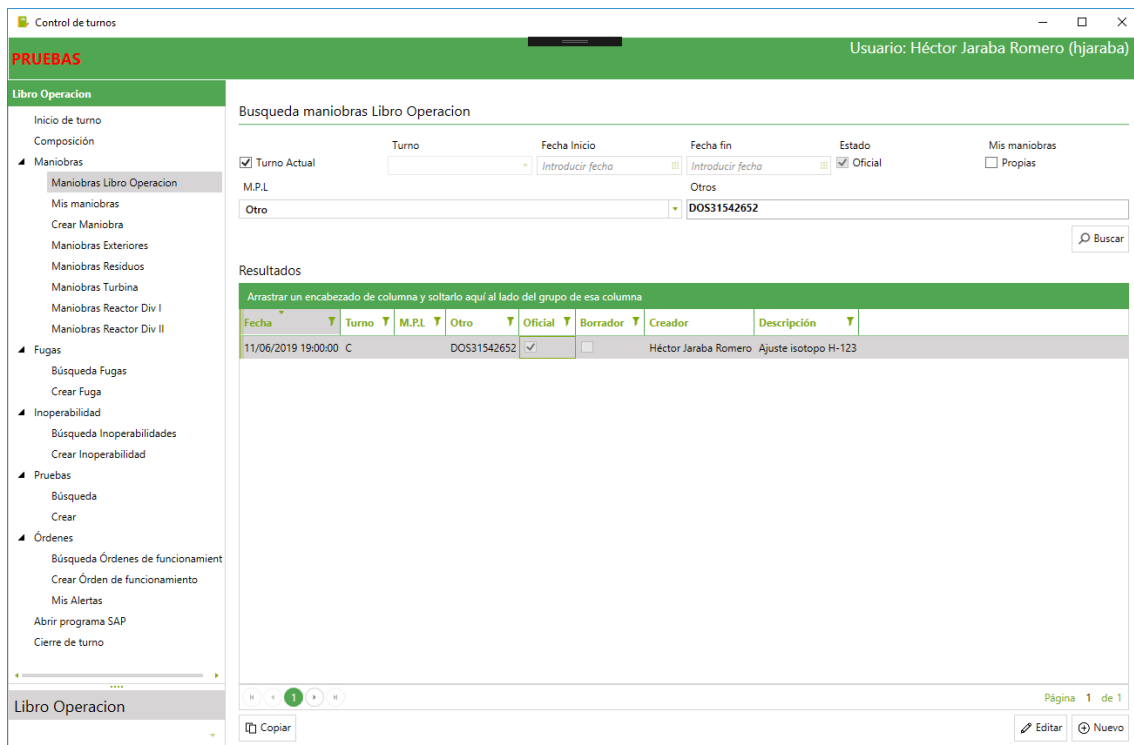


Figura 47: Pantalla Busqueda de maniobras

Por último, vamos a acceder con un usuario con el rol de *Encargado de Exteriores* para ver las diferencias entre este rol y el de *Jefe de Turno* dentro de la aplicación.

Cuando este usuario accede se le muestran si tiene alertas sin leer.

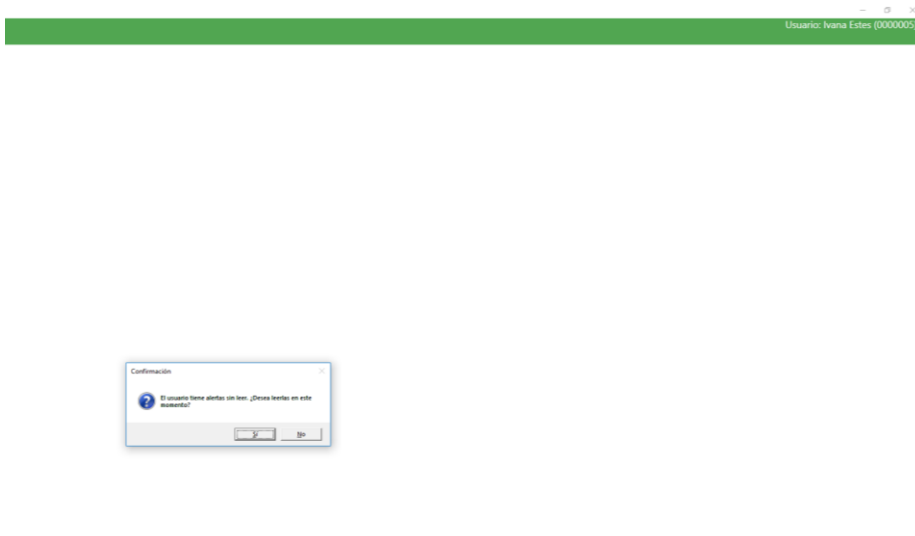


Figura 48: Pantalla aviso alertas

Se puede observar que este usuario no puede ver tantas opciones del menú de la aplicación como el usuario con el rol de *Jefe de Turno* ya que estas han sido restringidas por su rol.

Fecha	Turno	M.P.L	Otro	Oficial	Borrador	Creador	Descripción
11/06/2019 19:00:00	C		DOS31542652	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Héctor Jaraba Romero	Ajuste isotopo H-123
04/06/2019 11:00:00	B	MPL1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	Quincy Brown	zxczxczxczxc
04/06/2019 11:00:00	B	MPL1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	Héctor Jaraba Romero	adasdasd
02/06/2019 06:00:00	A	MPL1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	Heather Serrano	Operador reactor
29/05/2019 02:00:00	A	MPL1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	Héctor Jaraba Romero	Maniobra control de la sala

Figura 49: Pantalla lista maniobras rol encargado de exteriores

En el listado de maniobras aparece la maniobra que se ha creado con el usuario con el rol de *Jefe de Turno* y si la intentamos editar con este usuario vemos que no es posible ya que no dispone de los permisos suficientes para editar una *maniobra* que no le pertenece.

Desarrollo de un sistema software para la gestión de las operaciones de una central nuclear

Control de turnos

PRUEBAS

Usuario: Ivana Estes (0000005)

Libro Operacion

Inicio de turno

Maniobras

- Maniobras Libro Operacion
- Mis maniobras
- Crear Maniobra
- Maniobras Exteriores

Inoperabilidad

- Búsqueda Inoperabilidades
- Crear Inoperabilidad

Órdenes

- Búsqueda Órdenes de funcionamiento
- Crear Orden de funcionamiento
- Mis Alertas
- Abrir programa SAP

Detalle Maniobra

Turno	Fecha	Hora	Firma	
C	11/06/2019	19:00	Héctor Jaraba Romero	<input checked="" type="checkbox"/> Oficial

M.P.L. Otro

Otro D0531542652

Texto de la maniobra

Ajuste isotopo H-123

Libro Operacion

← Volver

Borrar Validar Guardar

Figura 50: Pantalla detalle maniobra rol encargado de exteriores

El resto de las opciones de la aplicación (fugas, operaciones, etc.) funcionan de manera muy similar a lo que se ha mostrado y no se añaden para no hacer excesivamente largo este trabajo de fin de grado.



7. Conclusiones

Tal y como se prometía al principio, este trabajo tenía como objetivo mostrar las distintas etapas por las que pasa un producto software.

Para conseguirlo se han aplicado la gran mayoría de conocimientos aprendidos durante el grado, sobre todo los de la rama cursada, Ingeniería del software. He de destacar que estos han sido de gran ayuda durante las practicas cursadas, ya que se ha conseguido implementar con éxito las tareas que se asignaban.

No hay duda de que el desarrollo de un producto software requiere de un proceso de ingeniería, con una serie de etapas y una estructura para poder acometerse con éxito. Esto se ha intentado transmitir a lo largo de este documento.

Aún faltarían etapas por cubrir en este documento, como el testing o la implantación del software. Durante el desarrollo de este programa el testing se hizo de una manera manual, con una hoja de Excel que definía las pruebas a realizar, y donde se indicaba si se pasaba el test o no mediante una casilla. Sin hacer uso de técnicas como test unitarios, así que no se consideró interesante incluirlo. En cuanto a la implantación no pude participar en esta etapa.

Durante estas prácticas, tuve la gran suerte de estar en proyectos que partían de cero, y trabajar junto con profesionales que me han asistido en las dudas que me surgían. También se trabajó en proyectos de mantenimiento. En estos proyectos se seguía un enfoque ágil como el que se explica en este documento, asignándonos las diferentes mejoras a implementar o fallos a solucionar entre un equipo pequeño de programadores.

Estas prácticas me han servido para mejorar como profesional, aplicando los conocimientos aprendidos a un contexto real.

8. Glosario de términos

Outsourcing: anglicismo que se refiere a la externalización de servicios. Es el proceso en el cual una organización contrata a una empresa externa para hacerse cargo de una parte de su actividad.

Mockup: es un diseño de la aplicación sin funcionalidad, que permite hacerse una idea del funcionamiento de esta.

Testing: es el proceso en el cual se prueba que el código implementado arroja el resultado esperado en diferentes escenarios.

Merge: Proceso de juntar dos fragmentos o archivos en uno solo.

Repositorio: espacio donde se almacena el código, normalmente se encuentra en un servidor.

ViewModel: En español modelo de la vista, es una clase utilizada por el patrón MVVM que se encuentra entre la vista y el modelo y contiene la lógica de la vista.

Feedback: anglicismo que se refiere a la reacción, respuesta u opinión que nos da un interlocutor en base a un asunto determinado.

SOAP: (De las siglas Simple Object Access Protocol) es un protocolo de comunicación utilizado para exponer servicios web que emplea archivos XML para el intercambio de datos.

XML: (De las siglas eXtensible Markup Language) es un meta-lenguaje que permite definir lenguajes de marcado con los que podemos estructurar los datos.

MPL: es un código asociado a un proceso dentro de la central nuclear.

9. Referencias

- [1] Roger S. Pressman. *Ingeniería del software un enfoque practico. Séptima edición*. MCGraw-Hill 2010. ISBN 978-607-15-0314-5
- [2] Robert C. Martin. *Código limpio*. Anaya 2012. ISBN: 978-84-4153-2106
- [3] Robert C. Martin. *Arquitectura limpia*. Anaya 2018. ISBN: 978-84-415-3990-7
- [4] Erich Gamma. *Arquitectura limpia*. Pearson Addison Wesley 2006. ISBN: 0-201-63361-2
- [5] Steve Krug. *Don't make me think*. New Riders Publishing 2013. ISBN: 978-03-2196-5516
- [6] Christian Nagel. *Professional C# 7 and .NET Core 2.0*. Wrox 2018. ISBN: 1119449278
- [7] Robert C. Martin. *The principles of OOD* [En línea] [Fecha de consulta: 10 de abril de 2019] Disponible en: <http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>
- [8] Cecilio Álvarez Caules. *El patrón de inyección de dependencia y su utilidad* [En línea] [Fecha de consulta: 15 de mayo de 2019] Disponible en: <https://www.arquitecturajava.com/el-patron-de-inyeccion-de-dependencia/>
- [9] Castle Project. *Windsord* [En línea] [Fecha de consulta: 10 de abril de 2019] Disponible en: <https://www.arquitecturajava.com/el-patron-de-inyeccion-de-dependencia/>
- [10] Microsoft. *Guia de C#*. [En línea] [Fecha de consulta: 24 de mayo de 2019] Disponible en: <https://docs.microsoft.com/es-es/dotnet/csharp/>

