TITULACIÓN: Ingeniería Técnica en Informática de Sistemas

Gestor de Datos de Banco de Pruebas para Generadores Síncronos

Alumno/a: D./Dña. David Pablos González

Director/a: D./Dña. German Rigau Claramunt

Proyecto Fin de Carrera, julio de 2012

Abstract

En este proyecto se desarrolla una aplicación web que permite la gestión de los datos obtenidos en las pruebas que se realizan a los generadores síncronos en Banco de Pruebas. El objetivo de este proyecto es almacenar estos datos en una base de datos de forma que se encuentren centralizados para su posterior explotación.

Para llevar a cabo esta tarea, se han utilizado tecnologías como PHP, MySQL, JavaScript y Ajax.

Palabras clave/Keywords: gestión, datos, generadores, síncronos, PHP, MySQL, JavaScript, Ajax

ÍNDICE

1. INTRODUCCIÓN 9
1.1. Presentación de la empresa 9
1.2. Antecedentes 11
2. DOCUMENTO DE OBJETIVOS DE PROYECTO 13
2.1. Descripción 13
2.2. Objetivos 13
2.3. Método de trabajo 14
2.4. Alcance 16
2.5. Planificación temporal. Diagrama Gantt 21
2.6. Riesgos. Plan de contingencia 22
2.7. Factibilidad 24
3. ARQUITECTURA 27
3.1. Arquitectura del software 27
3.2. Arquitectura del sistema 28
3.3. Elección tecnológica 29
4. CAPTURA DE REQUISITOS 37
4.1. Roles 37
4.2. Casos de Uso 39
4.2.1. Identificar Usuario 39
4.2.2. Crear 40
4.2.3. Abrir 41
4.2.4. Definición de pruebas 42
4.2.5. Copiar pruebas 43
4.2.6. Intercambiar pruebas 43
4.2.7. Definición de datos de cabecera 44
4.2.8. Entrada de datos 45
4.2.9. Configurar datos a cliente 46
4.2.10. Visualizar protocolo cliente 47
4.2.11. Editar resultados 47
4.2.12. Crear documentación cliente 48
4.2.13. Modificar responsables e implicados 49
4.2.14. Cerrar/Reabrir proyecto 50
4.2.15. Consultas 51
4.2.16. Configuración 51

4.2.17. Modificar lista master de pruebas 51
4.2.18. Añadir prueba 53
4.3. Modelo de Dominio 53
5. ANÁLISIS 59
5.1. Identificar Usuario 59
5.2. CrearProyecto 60
5.3. Abrir 62
5.4. Definir pruebas 63
5.5. Copiar Pruebas 65
5.6. Intercambiar pruebas 66
5.7. Definir datos de cabecera 68
5.8. Entrada de datos 69
5.9. Modificar responsables e implicados de proyecto 70
5.10. Cerrar/Reabrir proyecto 71
5.11. Configuración 73
5.12. Modificar lista master de pruebas 73
6. DISEÑO 77
6.1. Identificar Usuario 77
6.2. Crear proyecto 78
6.3. Abrir proyecto 83
6.4. Definir pruebas 84
6.5. Copiar pruebas 86
6.6. Intercambiar pruebas 87
6.7. Definición de datos de cabecera 89
6.8. Entrada de datos 90
6.9. Modificar responsables e implicados 91
6.10. Cerrar/Reabrir 93
6.11. Modificar lista master pruebas 95
7. IMPLEMENTACIÓN 97
7.1. Organización del código 97
7.1.1. Directorio raíz 97
7.1.2. Directorio ‘Gestion’ 100
7.2. Transacciones 103
7.3. Sesiones PHP 108
7.4. JavaScript 110
7.5. AJAX 111
7.6. jQuery 120

8. PRUEBAS 123
8.1. Conceptos generales 123
8.2. Pruebas unitarias 125
8.3. Pruebas de integración 130
8.4. Pruebas del sistema 130
9. GESTIÓN DEL PROYECTO 133
9.1. Parte de horas mensuales 133
9.2. Gantt planificado vs Gantt real 140
9.3. Conclusiones de la gestión 141
10. CONCLUSIONES 145
11. TRABAJO FUTURO 147
11.1. CSS 147
11.2. Generación de documentación 147
11.3. Consultas 148
11.4. Soporte multilenguaje 148
11.5. Soporte múltiples navegadores 149
12. BIBLIOGRAFÍA 151
13. ANEXO I. MANUAL DE USUARIO 153

1. INTRODUCCIÓN

El presente proyecto de final de carrera correspondiente a la titulación de Ingeniería Técnica en Informática de Sistemas se realizará por el alumno David Pablos durante el curso 2011/2012.

El desarrollo del mismo tendrá lugar en la empresa Indar Electric S.L., donde tendrá una duración aproximada de 5 meses y será supervisado desde Indar por Xabier Calvo y desde la Universidad del País Vasco por el profesor German Rigau.

1.1. Presentación de la empresa

Historia:

1940: Los hermanos Larrañaga y Ormazabal fundan INDAR.
1965: INDAR recibe el título de “Empresa Ejemplar”.
1997: INDAR se incorpora al GRUPO INGETEAM.
2000: Se inaugura la nueva planta en Beasain.
2003: Las distintas unidades de Negocio de INDAR experimentan un crecimiento considerable.
2006: INDAR se ha convertido en uno de los líderes mundiales en la fabricación de generadores hidráulicos, motores sumergibles de alta potencia y en el sector eólico para el que se han suministrado generadores que suponen mas del 12% de la potencia instalada en el mundo.

Descripción de INDAR:

Entre las empresas del grupo INGETEAM, INDAR con un equipo humano compuesto por mas de 650 trabajadores, destaca por su trayectoria, que viene avalada por la construcción de miles de motores y generadores para los sectores Industrial, Energético y Naval. Sus máquinas son conocidas y apreciadas por su robustez, calidad y máxima fiabilidad.

Además, INDAR Electric, S.L es una empresa que invierte de manera continuada en I+D+i, cuyo departamento está formado por más de 40 ingenieros y licenciados altamente cualificados y dotado con los medios más avanzados para el diseño y

desarrollo de máquinas con tecnología propia, garantizando la máxima flexibilidad y prestaciones y en estrecha colaboración con sus clientes.

INDAR cuenta con una planta en Beasain y posee un campo de actividad amplio que abarca desde la energía (eólica, hidroeléctrica, cogeneración y térmicas), la industria (siderurgia y metalurgia), el campo naval (propulsión, accionamientos varios, generación eléctrica, motores sumergibles) y ferroviario (Tracción, generación) hasta las infraestructuras (instalaciones de bombeo, planta desaladoras).

Unidades de Negocio:

Indar Electric está dividido en cuatro unidades de negocio especializadas:

- **Indar Wind Power**: Fabrican **Generadores Eólicos**, con potencias que van desde 850 kW hasta 6 MW y tensiones desde 690 V hasta 15 kV para turbinas de paso variable, asíncronos doblemente alimentados o síncronos de imanes permanentes. Refrigerados por aire o agua. Construcción horizontal. Según normas internacionales IEC, NEMA, etc. Servicio de Atención a Clientes: asistencia técnica, planes de mantenimiento a medida, reparaciones y reposición de elementos constructivos.

- **Indar Hydro**: Fabrican **Generadores Síncronos**, con potencias que van desde 1.250 kVA hasta 40 MVA y tensiones desde 690 V hasta 15 kV, para generación hidráulica con turbinas Pelton, Francis y Kaplan. Refrigerados por aire o agua. Construcción horizontal o vertical. Según normas internacionales IEC, NEMA, etc. Servicio de Atención a Clientes: asistencia técnica, planes de mantenimiento a medida, reparaciones y reposición de elementos constructivos.

- **Indar CIM (Cogeneration, Industry & Marine)**: Esta unidad de negocio desarrolla un gran número de productos, tales como:

  Generadores Síncronos, con potencias que van desde 1.250 kVA hasta 35 MVA y tensiones desde 690 V hasta 15 kV, para Cogeneración y Sector Naval con turbinas de vapor, gas y motores diesel. Refrigerados por aire o por agua. Construcción horizontal. Según normas internacionales IEC, NEMA, etc.

Motores Asíncronos, con potencias que van desde 400 kW hasta 15 MW, y tensiones desde 690 V hasta 15 kV, en jaula de ardilla o anillos rozantes, para todo tipo de aplicaciones en la Industria, Sector Energético, Naval, Infraestructuras y Medio Ambiente. Refrigerados por aire o por agua. Construcción horizontal o vertical. Hasta aislamiento H, de IP-23 hasta IP-56. Según normas internacionales IEC, NEMA, etc.

Motores Síncronos, con potencias que van desde 1 MW hasta 15 MW, y tensiones desde 690 V hasta 15 kV,para aplicaciones tanto a velocidad fija como variable en la Industria, Sector Energético, Naval, Infraestructuras y Medio Ambiente. Refrigerados por aire o por agua. Construcción horizontal o vertical. Según normas internacionales IEC, NEMA, etc.

Motores Corriente Continua, con potencias que van desde 400 kW hasta 4 MW en la Industria y Sector Naval.

Motores Sumergibles llenos de aceite o de aire hasta 1.000 metros de profundidad, en potencias que van desde 1.000 kW hasta 10 MW y tensiones desde 690 V hasta 15 kV para las dragas en el Sector Naval. Servicio de Atención a Clientes: asistencia técnica, planes de mantenimiento a medida, reparaciones y reposición de elementos constructivos.

- **Indar Repair**: Servicio Post-Garantía de Atención a Clientes: asistencia técnica, planes de mantenimiento a medida, reparaciones y reposición de elementos constructivos.

1.2. Antecedentes

En la actualidad, en la empresa, a la hora de registrar los resultados de las pruebas que se realizan a los generadores en el Banco de Pruebas, se utilizan un conjunto de ficheros Excel y Word.

Este método, aunque funcional, es bastante limitado, ya que los datos almacenados de esta manera no disponen de una forma de consulta realmente eficiente. Además, la utilización de una base de datos para almacenar la información conlleva otras ventajas tales como garantizar la consistencia de los datos y evita la información redundante.

La entrada de datos será más eficaz y precisa, ya que el usuario estará obligado a rellenar los campos de la forma indicada en las 'Normas de ensayo' (documentos que especifican qué se mide en cada prueba y cómo se mide) de la empresa, el registro de la información será muy completo.

El proyecto de desarrollo de un **Gestor de Datos de Banco de Pruebas para Generadores Síncronos** tiene la finalidad de proporcionar una interfaz web que sirva para gestionar toda la información generada en el Banco de Pruebas de generadores síncronos de la empresa.

Esto supone un reto ambicioso que puede ser dividido en 3 grandes bloques lógicos:

1. Entrada de datos: Será la parte de la aplicación que se encarga de solicitar al usuario los datos obtenidos en Banco de Pruebas de las pruebas realizadas a cada generador y gestionar la entrada de todos los datos necesarios.
2. Generación de documentación: En base a los datos introducidos y almacenados en la base de datos, se generarán diversa documentación.
3. Consultas: Permitirá realizar consultas sobre los datos almacenados.

Debido a la limitación temporal se ha tenido que acotar el alcance del proyecto al primer punto: entrada de datos y gestión de los mismos. Sin embargo, hay que señalar que la captura de requisitos se ha realizado para todos los casos de uso, no sólo los relativos a la entrada de datos.

2. DOCUMENTO DE OBJETIVOS DE PROYECTO

2.1. Descripción

El presente proyecto de final de carrera correspondiente a la titulación de Ingeniería Técnica en Informática de Sistemas se realizará por el alumno David Pablos durante el curso 2011/2012.

El desarrollo del mismo tendrá lugar en la empresa Indar Electric S.L., donde tendrá una duración aproximada de 5 meses y será supervisado desde Indar por Xabier Calvo y desde la Universidad del País Vasco por el profesor German Rigau.

Dicho proyecto surge de la necesidad de la unidad de negocio Hydro, extendiéndose también a la Unidad de Negocio CIM , de un **Gestor de Datos de Banco de Pruebas para Generadores Síncronos**, que sea capaz de gestionar de forma eficiente la información obtenida en el Banco de Pruebas.

La aplicación se integrará en la Intranet de Indar Electric, para que todo usuario con autorización pueda utilizarla.

2.2. Objetivos

Crear una aplicación web que gestione los datos obtenidos en el Banco de Pruebas de la empresa.

Al darse de alta un proyecto en este gestor, la aplicación automáticamente recuperará una serie de datos contenidos en diferentes repositorios de información existentes, a saber:

- Base de datos de Hojas de Diseño (Hydro)
- Base de datos de Hojas de Diseño de Máquina (CIM)
- Base de datos de Usuarios Hydro
- Base de datos de SAP.

Este gestor permitirá al usuario de Banco de Pruebas, la introducción de los datos obtenidos en las distintas pruebas realizadas a las máquinas, guardándose toda esta información en una base de datos.

El almacenamiento de los datos en una BD permitirá poder explotar, de una manera sencilla a través de consultas, toda la información introducida en este nuevo sistema. Las consultas podrán devolver desde datos de un proyecto o máquina en concreto hasta datos cruzados entre diferentes máquinas y proyectos.

Este **Gestor de Datos de Banco de Pruebas para Generadores Síncronos**, como su propio nombre indica, estará orientado a la casuística de Síncrono y se considera que aparte del conjunto de ficheros empleados en la actualidad debería considerarse y seguirse la 'Norma de Ensayos' existente actualmente a en INDAR.

Los datos a gestionar serían tanto de carácter eléctrico como mecánico.

2.3. Método de trabajo

Este proyecto se aborda con la intención de presentarlo en julio de 2012. Con el fin de lograr este objetivo se dispondrá de un mínimo de 35 horas semanales, que se podrán incrementar en el caso de que sea necesario para cumplir con los plazos o se considere conveniente.

En el caso de que no se pueda entregar el proyecto en la fecha indicada, se hará la presentación en la siguiente convocatoria.

Al tratarse de un proyecto en empresa, la elección tecnológica vendrá dada por las preferencias y la propiedad de licencias de la empresa en la que se realizará el proyecto de fin de carrera.

Se dividirá el trabajo en distintas fases, siguiendo el Proceso Unificado de Desarrollo, que es un marco de desarrollo software iterativo e incremental, y se caracteriza fundamentalmente por estar dirigido por casos de uso. Las iteraciones siguen el desarrollo en cascada, que ordena rigurosamente las etapas del ciclo de vida del software, y en la que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior.

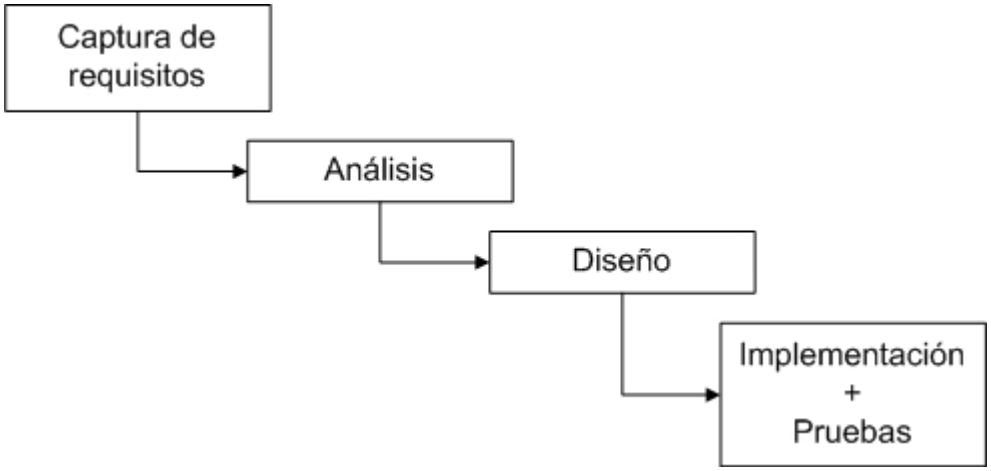


Figura 2.1. Desarrollo en cascada de cada iteración

- **Captura de requisitos**: La primera etapa en el desarrollo de un software es la extracción de los requisitos en base a lo que se desee que haga el software, y en base a ello llevar a cabo el desarrollo de la aplicación.

- **Análisis**: Establecer operaciones mediante contratos, en los cuales se fija qué hará cada función, qué entrada tendrá, qué cambios en el sistema realizará y qué salida ofrecerá.

- **Diseño**: El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación. Presenta la estructura de los datos, la arquitectura del software y la caracterización de la interfaz.

- **Implementación**: Una vez analizado y diseñado el problema, se realiza la aplicación en base a ello. Cuanto mejores sean la captura de requisitos, el análisis y el diseño, más sencilla resultará la implementación, y más fiable será la aplicación resultante.

- **Pruebas**: Una vez se ha llevado a cabo la implementación, se procede a realizar pruebas sobre el sistema o sobre uno de sus componentes, con el fin de detectar el mayor número de errores posible y corregirlos.

2.4. Alcance

2.4.1. Esquema de descomposición de trabajos

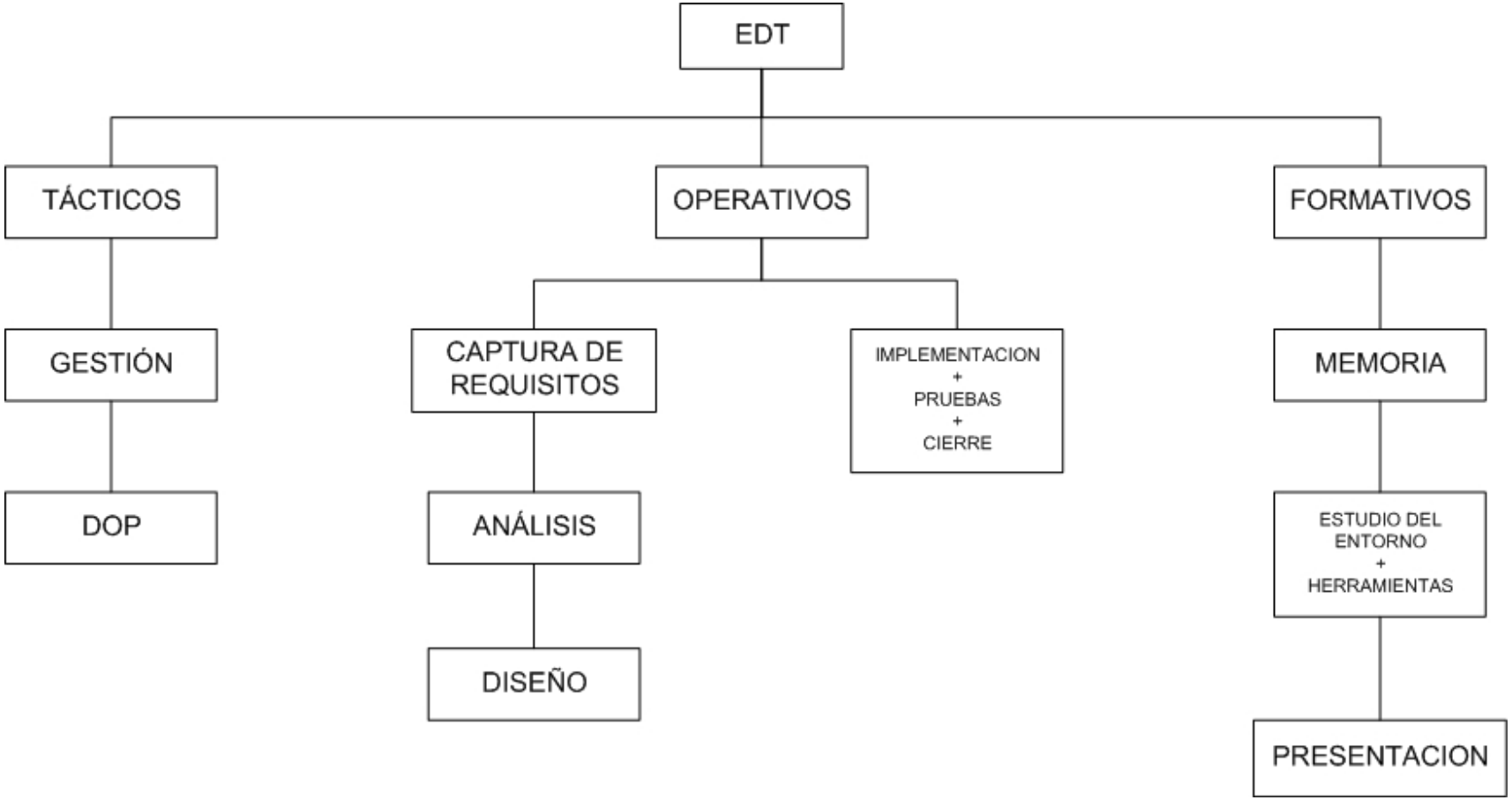


Figura 2.2. Esquema EDT

2.4.2. Estimación de esfuerzo

La dedicación total para la realización del presente proyecto se estima en aproximadamente 640 horas/persona.

2.4.2.1. Fase 1.

Estimación de esfuerzo: 3 días * 7 horas/día.

Tareas:

DOP:

- Descripción.
- Objetivos.
- Método de trabajo.
- Alcance.
- Planificación.
- Plan de contingencia.
- Factibilidad.

Gestión de proyecto.

Tipo: Informe.

2.4.2.2. Fase 2.

Estimación de esfuerzo: 2 semanas * 7 horas/día.

Tareas:

- Estudio del entorno y de las herramientas.

Gestión de proyecto.

2.4.2.3. Fase 3.

Estimación de esfuerzo: 2 semanas * 7 horas/día.

Tareas:

Captura de requisitos de la aplicación

- Roles.
- Casos de uso.
- Modelo de dominio.
- Reuniones.

- Gestión de proyecto.

Tipo: Informe.

2.4.2.4. Fase 4.

Estimación de esfuerzo: 1 semana * 7 horas/día.

Tareas:

Análisis

- Diagramas de secuencia del sistema.
- Contratos.
- Reuniones.
- Gestión de proyecto.

Tipo: Informe.

2.4.2.5. Fase 5.

Estimación de esfuerzo: 3 semanas * 7 horas/día.

Tareas:

Diseño

- Diagramas de iteración.
- Diagramas de clase.
- Reuniones.
- Gestión de proyecto.

Tipo: Informe.

2.4.2.6. Fase 6.

Estimación de esfuerzo: 2 semanas * 7 horas/día.

Tareas:

Desarrollo, primera iteración.

- Implementación.
- Pruebas.
    - Cierre.
- Reuniones.
- Gestión de proyecto.

Tipo: Informe.

2.4.2.7. Fase 7.

Estimación de esfuerzo: 2 semanas * 7 horas/día.

Tareas:

Desarrollo, segunda iteración.

- Implementación.
- Pruebas.
    - Cierre.
- Reuniones.
- Gestión de proyecto.

Tipo: Informe.

2.4.2.8. Fase 8.

Estimación de esfuerzo: 3 semanas * 7 horas/día.

Tareas:

Desarrollo, tercera iteración.

- Implementación.
- Pruebas.
    - Cierre.
- Reuniones.
- Gestión de proyecto.

Tipo: Informe.

2.4.2.9. Fase 9.

Estimación de esfuerzo: 2 semanas * 7 horas/día.

Tareas:

- Redacción de la memoria.

Tipo: Informe.

2.4.2.10. Fase 10.

Estimación de esfuerzo: 3 días * 7 horas/día.

Tareas:

Defensa del proyecto

- Redacción de las diapositivas de la presentación.
- Ensayo.

Tipo: Informe.

2.4.2.11. Total

Estimación de esfuerzo:

Tareas:

- Fase 1. DOP: 21 horas.
- Fase 2. Estudio del entorno y herramientas: 70 horas.
- Fase 3. Captura de requisitos: 70 horas.
- Fase 4. Análisis: 35 horas.
- Fase 5. Diseño: 105 horas.
- Fase 6. Desarrollo 1ª iteración: 70 horas.
- Fase 7. Desarrollo 2ª iteración: 70 horas.
- Fase 8. Desarrollo 3ª iteración: 105 horas.
- Fase 9. Memoria: 70 horas.
- Fase 10. Presentación: 21 horas.

Total: 637 horas.

2.5. Planificación temporal. Diagrama Gantt

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Comienzo del proyecto	0 días	lun 13/02/12	lun 13/02/12
2	Fase 1	8 días?	lun 13/02/12	mié 22/02/12
3	Fase 2	11 días?	jue 23/02/12	jue 08/03/12
4	Fase 3	16 días?	vie 09/03/12	vie 30/03/12
5	Fase 4	10 días?	lun 02/04/12	vie 13/04/12
6	Fase 5	10 días?	lun 16/04/12	vie 27/04/12
7	Fase 6	10 días?	lun 30/04/12	vie 11/05/12
8	Fase 7	11 días?	lun 14/05/12	lun 28/05/12
9	Fase 8	25 días?	mar 29/05/12	lun 02/07/12
10	Fase 9	25 días?	lun 04/06/12	vie 06/07/12
11	Fase 10	4 días?	lun 09/07/12	jue 12/07/12
12	Fin de proyecto	0 días	jue 12/07/12	jue 12/07/12

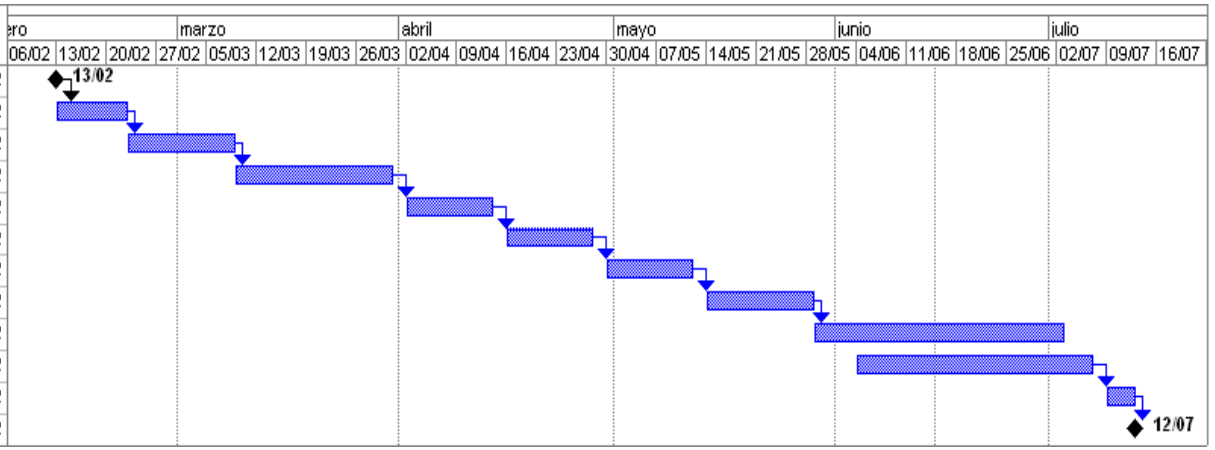


Figura 2.3. Diagrama de Gantt

Fase 1: DOP
Fase 2: Estudio del entorno y de las herramientas
Fase 3: Captura de requisitos
Fase 4: Análisis
Fase 5: Diseño
Fase 6: 1ª Iteración + Pruebas + Cierre
Fase 7: 2ª Iteración + Pruebas + Cierre
Fase 8: 3ª Iteración + Pruebas + Cierre
Fase 9: Memoria
Fase 10: Defensa

2.6. Riesgos. Plan de contingencia

Los imprevistos que pueden poner en peligro el proyecto son variados, por ello se intentará tener identificado un buen número de ellos de antemano para saber cómo actuar en el caso de que se dé esa situación, o incluso prevenir dicho escenario.

2.6.1. Pérdida de documentos

Para evitar que la pérdida de documentos suponga la pérdida de parte o incluso de la totalidad del proyecto, al final de cada sesión de trabajo se procederá a hacer una copia de seguridad tanto en el servidor donde tendrá lugar el desarrollo de la aplicación, como en el ordenador de usuario del que dispongo en Indar. Aparte de estos 2 sitios, Indar cuenta con un disco duro en red del que se hacen backups diarios en el cual se guardará tanto la información relativa a la memoria como la generada para la empresa. Una vez a la semana además de en estos 2 sitios, se guardará una copia de seguridad semanal en el correo electrónico.

Este supuesto tiene una probabilidad relativamente alta de suceder, sobre todo en el momento de la implementación, y puede tener un impacto muy alto, por lo que es importantísimo contar con copias de seguridad.

2.6.2. Retraso en los plazos establecidos

En el caso de que se dé esta situación, se procederá a aumentar el número de horas diarias de 7 a 8, además de adelantar trabajo el fin de semana si llega a ser necesario. No obstante, a la hora de hacer la planificación he intentado ser generoso con los plazos de cada tarea para evitar este problema. Debería dar tiempo suficiente para todas las fases, salvo quizás para la de implementación, pruebas y cierre, que al ser la fase más compleja y más sujeta a imprevistos, es probable que tenga un plazo más ajustado.

Este escenario tiene una probabilidad alta de suceder, pero a pesar de todo, no tendría un impacto tan grande como podría parecer. La fecha límite es la que impone la Universidad del País Vasco para entregar la memoria, en este caso el 10 de julio de 2012. A pesar de todo, se pondrán todos los medios necesarios para que tanto la

aplicación como la documentación relativa a ellos estén preparados en la fecha prevista. El mayor riesgo en este caso lo representa la coordinación de las diferentes personas con las que tendré que interactuar para llevar a cabo esta labor.

2.6.3. Problemas de diseño y/o implementación

Para cada posible funcionalidad de la aplicación, se estudiará detenidamente si dicha funcionalidad es realmente necesaria o no para el funcionamiento del programa. También se diferenciará entre funcionalidades necesarias y optimizaciones, dando prioridad a las primeras.

En este caso la probabilidad es baja, ya que contaré con el apoyo de una persona dentro de la empresa, y el impacto de este imprevisto es indeterminado, desde una gravedad muy baja, a una bastante alta.

2.6.4. Virus en el equipo de trabajo

En aras de evitar que un virus informático deje maltrecho o incluso inutilizable el ordenador sobre el que voy a trabajar, se dispondrá de un antivirus instalado de forma local y actualizado constantemente. Dicho antivirus comprobará todas las operaciones realizadas sobre los ficheros del sistema.

La probabilidad de que se dé este supuesto, es relativamente baja, ya que al tratarse de un equipo exclusivamente de trabajo, el riesgo de la infección con software malicioso es menor, y se cuenta con un antivirus completamente actualizado desde el primer momento. El impacto podría comprender un espectro muy amplio de gravedad, desde muy baja a muy alta.

2.6.5. Desconocimiento de las tecnologías empleadas

Al tratarse de un proyecto real en empresa, es posible que en algún momento surja una situación irresoluble por incompatibilidad con la herramienta o desconocimiento de la misma por parte del alumno. Para evitar este problema, se pensará en una forma

alternativa de realizar una acción en la aplicación, por ejemplo modificando el caso de uso correspondiente, en caso de no saber cómo llevarla a cabo en un inicio.

La probabilidad de que esto suceda en algún momento es alta, pero la gravedad de la situación sería baja en cierta medida, ya que generalmente existen varias formas de llevar a cabo una tarea.

2.6.6. Fallo de hardware

Para evitar que un fallo de hardware en el ordenador de trabajo impida el trabajo, poniendo en peligro los plazos de entrega, se contará con un equipo de reserva correctamente configurado para poder trabajar desde él.

El impacto de un fallo de hardware sería alto, ya que imposibilitaría el trabajo por completo si no se cuenta con una alternativa preparada. La probabilidad de que suceda este caso es baja.

2.6.7. Seguridad

Todos los equipos utilizados para llevar a cabo este proyecto estarán protegidos con contraseñas seguras, con el fin de salvaguardar la información relativa al mismo, y evitar que usuarios no autorizados o malintencionados puedan acceder a ellos.

La gravedad de un fallo en la seguridad del sistema sería baja, ya que sólo personal autorizado tiene en principio acceso físico al equipo, con lo que podría descartarse que se trate de un usuario malicioso, y la aplicación se encuentra en un servidor interno de la empresa que no tiene acceso desde el exterior. La probabilidad de que esto suceda es baja, ya que como norma general siempre se utilizan contraseñas seguras en todos los lugares donde es necesaria.

2.7. Factibilidad

Teniendo una aproximación de 637 h/persona para la realización del proyecto y teniendo en cuenta que el número de horas totales de las que podría disponer superan

las 650, además de contar con un plan de contingencia para imprevistos, considero factible la realización de esta tarea.

3. ARQUITECTURA

3.1. Arquitectura del software

La Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema. Define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos.

Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. En cuanto a patrones de arquitectura se refiere, se barajaron 2 posibles modelos:

- Modelo vista controlador (MVC):
    - Modelo: Es el encargado de la lógica de negocio y el acceso a la base de datos.
    - Vista: Interacciona con la interfaz de usuario.
    - Controlador: Responde a eventos, generalmente del usuario e invoca cambios en el modelo y en la vista.
- Arquitectura en tres capas:
    - Capa de presentación: Es la capa que ve el usuario, presenta el sistema al usuario. Interacciona únicamente con la capa de negocio. Es la encargada de presentar los resultados generados por capas inferiores.
    - Capa de negocio: Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. Se encarga de la funcionalidad de la aplicación.
    - Capa de datos: Capa encargada de la gestión de los datos, recibirá consultas y devolverá datos a la capa de negocio.

Tras las oportunas meditaciones y después de consultar con algunas personas, la elección ha sido el patrón arquitectónico en 3 capas, debido principalmente a que es el patrón que mejor conozco y más he utilizado.

3.2. Arquitectura del sistema

La aplicación **GBP** se encontrará en uno de los servidores de INDAR, y para su correcto funcionamiento deberá interactuar con múltiples bases de datos diferentes, a saber: la base de datos de Hojas de Diseño de Máquina[1] de la unidad de negocio 'Hydro', la cual utiliza MySQL. La base de datos de Hojas de Diseño de Máquina de la unidad de negocio 'CIM', la cual utiliza MS SQL Server.

Estas dos bases de datos se utilizarán principalmente para obtener de ellas la información relacionada con la cabecera del proyecto. La cabecera del proyecto es la información más importante y general de un proyecto.

Sin embargo, también tendrá que hacerse uso de otras 3 bases de datos, de forma puntual, que son: la base de datos en la que se encuentran los empleados de 'Hydro', y la sección a la que pertenecen dentro de la unidad de negocio (Comercial, Técnica eléctrica, Técnica mecánica…). Esta base de datos se utiliza de forma muy puntual, solamente para obtener los posibles responsables e implicados de un proyecto, en el caso de uso 'Crear proyecto'. Es decir, de la base de datos se obtienen los posibles responsables e implicados, y de ahí se le asignan al proyecto que queremos dar de alta en el **GBP**. El SGBD de esta base de datos es MySQL.

Otra base de datos utilizada es la de SAP[2], para obtener los números de serie de los generadores de los proyectos. El SGBD de esta base de datos es Oracle.

Finalmente, la otra base de datos de la que se obtendrá información es la base de datos del Gestor de Elementos Críticos. Este gestor, conocido como **GEC**, es una aplicación de INDAR que gestiona los elementos mecánicos más importantes de un generador. Recurriremos a su base de datos para obtener los datos relativos a los apoyos de un generador. Esta información comprende el tipo de apoyo (rodamientos o cojinetes), el tipo de lubricación (grasa o aceite), viscosidad, nombre de apoyo, caudal de aceite, etc. Esta base de datos será del tipo MySQL.

[1] Las hojas de diseño son los documentos en los que se especifican todos los parámetros del generador que se va a construir
[2] Servicio de Atención Postventa

Y por último, la base de datos propia del **GBP**, en la cual se almacenarán todos los datos obtenidos en Banco de Pruebas. Esta base de datos también será de tipo MySQL.

Para explicar esto de una forma más clara, se adjunta la siguiente imagen:

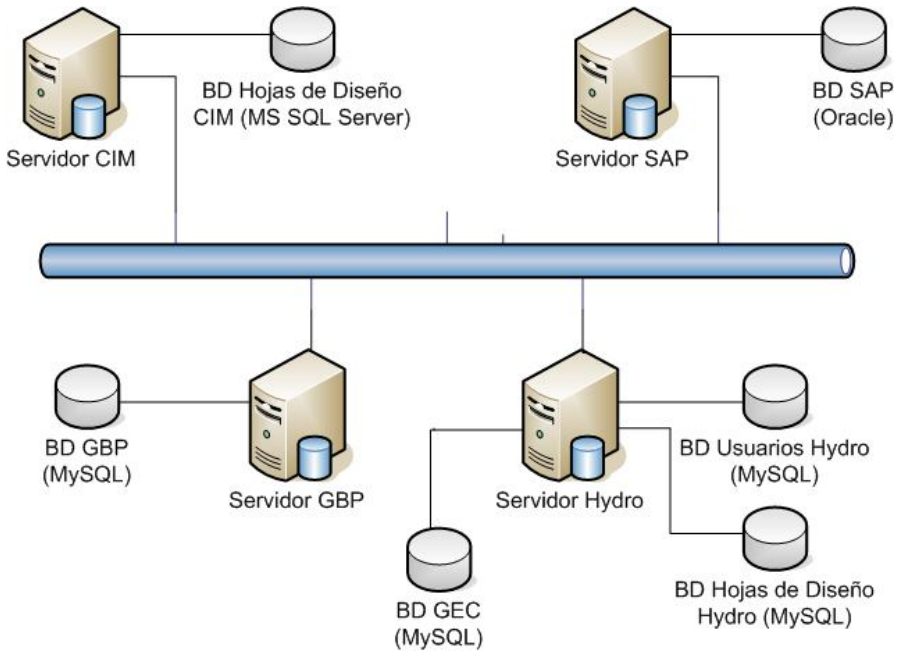


Imagen 3.1 Arquitectura del sistema

3.3. Elección tecnológica

Las tecnologías a utilizar para el desarrollo del proyecto han estado determinadas casi por completo por la empresa, por lo que en este sentido no ha estado en mi mano la decisión. Debido a que lo que se pretende es desarrollar una aplicación web, Indar Electric S.L. se decantó por la utilización de PHP5 como lenguaje de programación, ya que este es un lenguaje muy extendido a la hora de hacer este tipo de desarrollos.

Como base de datos, se utilizará MySQL, el SGBD[3] de Oracle que se usa con mayor frecuencia conjuntamente con PHP. El mecanismo de almacenamiento será InnoDB, el cual soporta transacciones de tipo ACID, además de bloqueo de registros e integridad referencial.

[3] Sistema de Gestión de Bases de Datos

La aplicación se ejecutará en un servidor corriendo sobre Linux perteneciente a la red interna de la empresa, aunque se desarrollará sobre un PC con Windows XP y Windows 7 como sistema operativo.

El navegador con el cual deben poder visualizarse los distintos apartados del gestor es Internet Explorer 8, el navegador estándar de uso interno de la empresa.

Debido al particular entorno donde se ejecutará la aplicación, y las múltiples interacciones con bases de datos de todo tipo que tiene que tener, procedemos a explicar los aspectos más relevantes de la elección tecnológica relacionados con las bases de datos.

Básicamente, la aplicación interactuará con bases de datos Oracle, para obtener los números de serie de los generadores, con bases de datos MySQL, ya que es este SGBD el que se utilizará para la propia base de datos del **GBP**, además de la base de datos MySQL del Gestor de Elementos Críticos (GEC), del cual obtendremos los datos relativos a los cojinetes, y la base de datos SQL Server de la unidad de negocio Cim, de la cual obtendremos los datos de sus proyectos, de la forma indicada por la figura 3.1.

En principio, se planteó la posibilidad de utilizar PDO para poder acceder a distintos tipos de bases de datos. PDO es una extensión de PHP 5 que define una interfaz ligera para tener acceso a bases de datos en PHP. Proporciona una capa de abstracción de acceso a datos, es decir, independientemente de la base de datos que se esté utilizando, se utiliza las mismas funciones para realizar consultas y obtener datos.
En cualquier caso, rápidamente nos dimos cuenta de que el servidor de desarrollo no disponía de los drivers necesarios, y en su lugar sí que tenía disponibles los drivers ADOdb, mssql y mysqli, por lo que son estos los drivers que finalmente he utilizado en el desarrollo de la aplicación.

Para conectar con la base de datos Oracle, se utiliza el conjunto de librerías para PHP ADOdb. La principal ventaja que aporta ADOdb es que oculta las diferencias entre cada API de base de datos, ya que en PHP las funciones de acceso a base de datos no están estandarizadas, permitiendo de esta forma una abstracción que proporciona una mayor portabilidad del código. En ese sentido es una librería similar a PDO.

Actualmente soporta un gran número de SGBD's, entre los cuales se encuentran MySQL, Oracle, Microsoft SQL Server, Sybase, Sybase SQL, Informix, Anywhere, FrontBase, PostgreSQL, SQLite, FrontBase, Interbase (versiones de Firebird y Borland), Access, FoxPro, ADO, DB2, SAP DB y ODBC.

Esto quiere decir, que de haber sido necesario, se podría haber realizado todo el proyecto utilizando ADOdb, ya que proporciona soporte para todas las bases de datos que hay que utilizar. Sin embargo, aunque esto habría sido lo más sencillo en principio, se ha optado por utilizar las extensiones de bases de datos específicas del proveedor MySQL mejorado, conocido como Mysqli, y el driver de Microsoft SQL Server para PHP conocido como SQLSRV, para la base de datos Microsoft SQL Server. Esta decisión se debe a que el rendimiento de las extensiones nativas de PHP es mucho mejor. De esta manera, se limita el uso de ADOdb a la obtención de los números de serie de un proyecto, que es para lo que realmente es necesario.

Así pues, en resumen, se dispone de ADOdb para leer los números de serie de un proyecto, sqlsrv para leer la base de datos de Cim, en la cual se encuentran los datos de las hojas de diseño de máquina, y finalmente, mysqli para leer y escribir en la base de datos del **GBP**, **GEC** y hojas de diseño de máquina de Hydro. Como en la base de datos de **GBP** es en la que más se va a trabajar, además de ser la única en la que se va a escribir, en el apartado implementación los ejemplos de transacciones irán dados preferentemente para mysqli.

Uno de los aspectos más importantes a tener en cuenta a la hora de escribir en una base de datos, es preservar la integridad y coherencia de los mismos. Por este motivo, es fundamental utilizar transacciones.

Una transacción en una base de datos es básicamente un conjunto de órdenes que se ejecutan formando una única unidad de trabajo, es decir, de forma indivisible o atómica.

Un SGBD se dice transaccional, si es capaz de mantener la integridad de los datos, haciendo que estas transacciones no puedan finalizar en un estado intermedio. Cuando por alguna causa el sistema debe cancelar la transacción, empieza a deshacer las órdenes ejecutadas hasta dejar la base de datos en su estado inicial (llamado punto de integridad), como si la orden de la transacción nunca se hubiese realizado.

En la actualidad, la mayoría de los SGBD son transaccionales. En el caso de MySQL, el mecanismo de almacenamiento InnoDB, soporta transacciones de tipo ACID, además de bloqueo de registros e integridad referencial.

Para que una transacción proporcione la garantía de que va a ser procesada correctamente, debe cumplir algunas propiedades, lo que se conoce comúnmente con el nombre de ACID. Estas propiedades son las siguientes:

- Atomicidad: Es la propiedad que requiere que cada transacción sea del tipo 'todo o nada', es decir, que si una parte de una transacción falla, entonces toda la transacción falla, y la base de datos vuelve al estado previo a iniciar la transacción. Para que un sistema proporcione atomicidad, debe garantizarla en todas las posibles situaciones, incluso en interrupción del servicio eléctrico, por ejemplo.
- Consistencia: Esta propiedad asegura que cualquier transacción llevará la base de datos de un estado válido a otro estado válido. Cualquier dato escrito en la base de datos debe cumplir ciertas reglas.
- Aislamiento: Es la propiedad que asegura que la ejecución concurrente de transacciones da como resultado un estado de la base de datos tal que podría haber sido obtenido si todas las transacciones se hubiesen realizado en serie (una después de otra).
- Durabilidad: Esta propiedad asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer, ni siquiera en un posible fallo del sistema.

Otra de las tecnologías que se han utilizado para realizar este proyecto es JavaScript. Éste es un lenguaje de programación interpretado, que se utiliza principalmente en el lado del cliente. Como parte de un navegador web, permite mejoras en la interfaz de usuario y páginas web dinámicas. Con este lenguaje se pueden crear diferentes efectos e interactuar con los usuarios.

Ajax es el acrónimo de **A**synchronous **J**avaScript **A**nd **X**ML (JavaScript asíncrono y XML). Es una técnica de desarrollo web que se ejecuta en el navegador del usuario, es decir, en el lado del cliente, mientras se mantiene una comunicación asíncrona con el servidor en segundo plano. Su principal utilidad radica en que permite realizar cambios sobre las páginas sin necesidad de recargarlas, lo cual permite aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

El proceso de una petición del cliente al servidor mediante la utilización de Ajax sigue un proceso como el siguiente:

1. En el lado del cliente, utilizando JavaScript se crea e inicializa el objeto XMLHttpRequest y se hace una petición al servidor, mediante GET o POST.
2. En el lado del servidor, se recibe la petición, se procesa, y se genera una respuesta en formato XML.
3. En el lado del cliente, se procesa la respuesta recibida del servidor, modificando mediante el DOM solamente los elementos de la página que sea necesario.

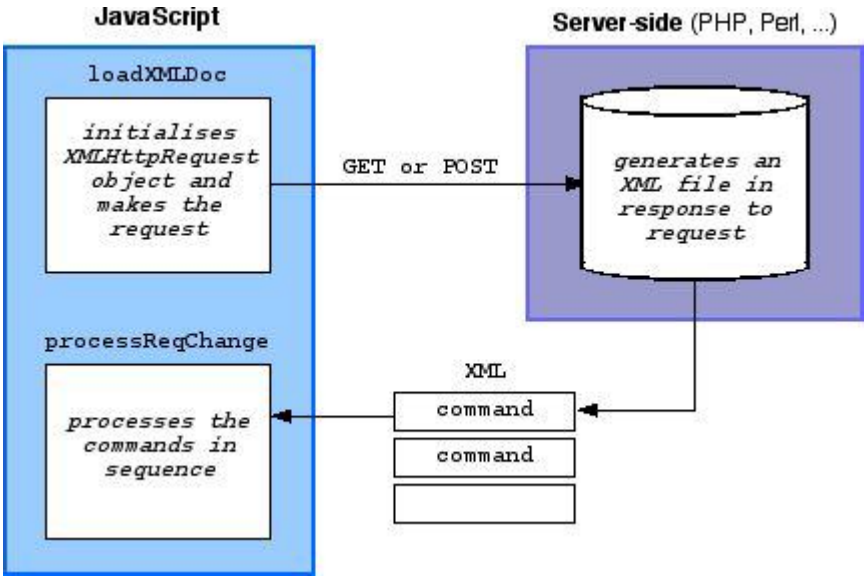


Imagen 3.2. Diagrama de flujo de una petición mediante AJAX

A continuación se explicará el proceso con más detalle, adjuntando para una mejor comprensión, el código de la aplicación que obtiene mediante AJAX la lista de números de serie asociados a un proyecto dado.

En primer lugar, se crea un objeto XMLHttpRequest. Dicho objeto es una interfaz empleada para realizar peticiones HTTP y HTTPS a servidores web, de forma asíncrona, por lo que proporciona contenido dinámico y actualizaciones asíncronas en páginas web, constituyendo la base de la tecnología AJAX.

Para los datos transferidos mediante XMLHttpRequest se utiliza cualquier codificación basada en texto, como: texto plano, XML, JSON, HTML e incluso codificaciones particulares específicas. No obstante, lo más habitual es que los datos transferidos

estén en formato XML o JSON. En el **GBP** las respuestas se devolverán en formato XML, por ser el más utilizado.

Debido a que conforme avanzaba el proyecto se hacía necesario hacer uso de efectos más complejos, se tomó la decisión de utilizar jQuery de forma puntual para realizar algunos efectos y también realizar las validaciones de los formularios de forma mucho más sencilla y escribiendo mucho menos código.

jQuery es una biblioteca o framework de JavaScript muy popular, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX a páginas web. Es utilizada por un gran número de desarrolladores y en un gran número de proyectos, muchos de los cuales son de gran envergadura, lo que demuestra la utilidad y fiabilidad de esta librería.

Probablemente, la decisión de mayor calado que he tenido que tomar ha sido la utilización o no utilización de un framework. Tras mucho meditarlo, tomé la decisión de no utilizar ninguno. Efectivamente, los frameworks facilitan el trabajo, pero tomé esta decisión porque desarrollar el proyecto sin framework, si bien es notablemente más tedioso, me ha permitido tener un control total sobre el código escrito y comprender más profundamente las tecnologías utilizadas. Analizándolo con posterioridad, creo que fue la decisión acertada, ya que de esta manera el proceso de aprendizaje es más natural, y de esta forma ahora podría comenzar a utilizar algún framework y la curva de aprendizaje sería mucho más rápida. En cualquier caso, comentar que los posibles entornos de trabajo que barajé en su día fueron ‘Zend Framework’, ‘CodeIgniter’ y ‘Yii’.

Todo el código está escrito con la aplicación Notepad++ v.6.1.2, la cual es un editor de texto de código abierto con soporte para varios lenguajes de programación, para plataformas Windows.

En cuanto a la creación de documentación para el proyecto han sido utilizadas varias aplicaciones, siempre teniendo como opciones aquellas de las cuales posee licencia la empresa. En este caso la suite ofimática utilizada ha sido Microsoft Office 2003.

Para redactar la memoria se ha utilizado Microsoft Word, para realizar los distintos diagramas se ha utilizado Microsoft Visio y para el diagrama de Gantt la aplicación

utilizada ha sido Microsoft Project. En un principio para el diagrama de Gantt se utilizó Microsoft Visio, pero con Microsoft Project se obtuvo un resultado más satisfactorio.

4. CAPTURA DE REQUISITOS

Esta sección tiene como objetivo identificar y describir los diferentes requisitos que son necesarios para llevar a cabo con éxito las tareas que se espera que haga el **Gestor de Datos de Banco de Pruebas**, de aquí en adelante **GBP**.

El objetivo final del **GBP** es facilitar la gestión de la información generada en el Banco de Pruebas de la empresa, permitiendo el tratamiento de dichos datos, y posibilitar el intercambio de información de forma ágil entre las líneas de negocio[4] CIM e Hydro. Se gestionará la entrada, edición y almacenamiento de los datos recogidos obtenidos en las pruebas realizadas a los generadores en el Banco de Pruebas. A grandes rasgos, estas serán las funciones del gestor que se va a desarrollar.

4.1. Roles

Los distintos perfiles que pueden tener los usuarios de la aplicación son 4, dependiendo de su relación con el proyecto[5]. Se presentarán en orden creciente de privilegios, ya que los perfiles con menos privilegios son contenidos por los perfiles superiores. Es decir, el perfil Implicado está contenido por el perfil Responsable, esto es, el Responsable podrá hacer todo lo que hace el Implicado y algunas cosas más, y así sucesivamente.

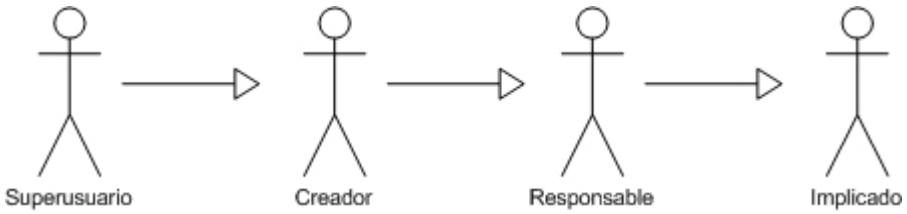


Imagen 4.1. Roles en la aplicación

[4] Línea, unidad de negocio y línea de negocio son sinónimos.
[5] Proyecto y pedido también son sinónimos.

4.1.1. Implicado

Este rol es el que tiene menos permisos, básicamente sólo podrá realizar consultas de algunos datos, y en ningún caso podrá realizar modificaciones.

4.1.2. Responsable

Este rol es el inmediatamente superior al Implicado. Los usuarios que pertenezcan a este perfil están autorizados a todo lo del perfil anterior, y además podrán abrir proyectos, e introducir/modificar datos.

4.1.3. Creador

Este rol es el inmediatamente superior al Responsable. Los usuarios que pertenezcan a este perfil están autorizados a todo lo que están autorizados los responsables, y además podrán dar de alta proyectos en el **GBP**.

4.1.4. Superusuario

Este rol es el inmediatamente superior al Creador. Los usuarios que pertenezcan a este perfil están autorizados a todo lo que están autorizados los Creadores, y además podrán acceder al apartado “Configuración”[6] de la aplicación.

[6] Área de la aplicación detallada en el apartado de casos de uso

4.2. Casos de Uso

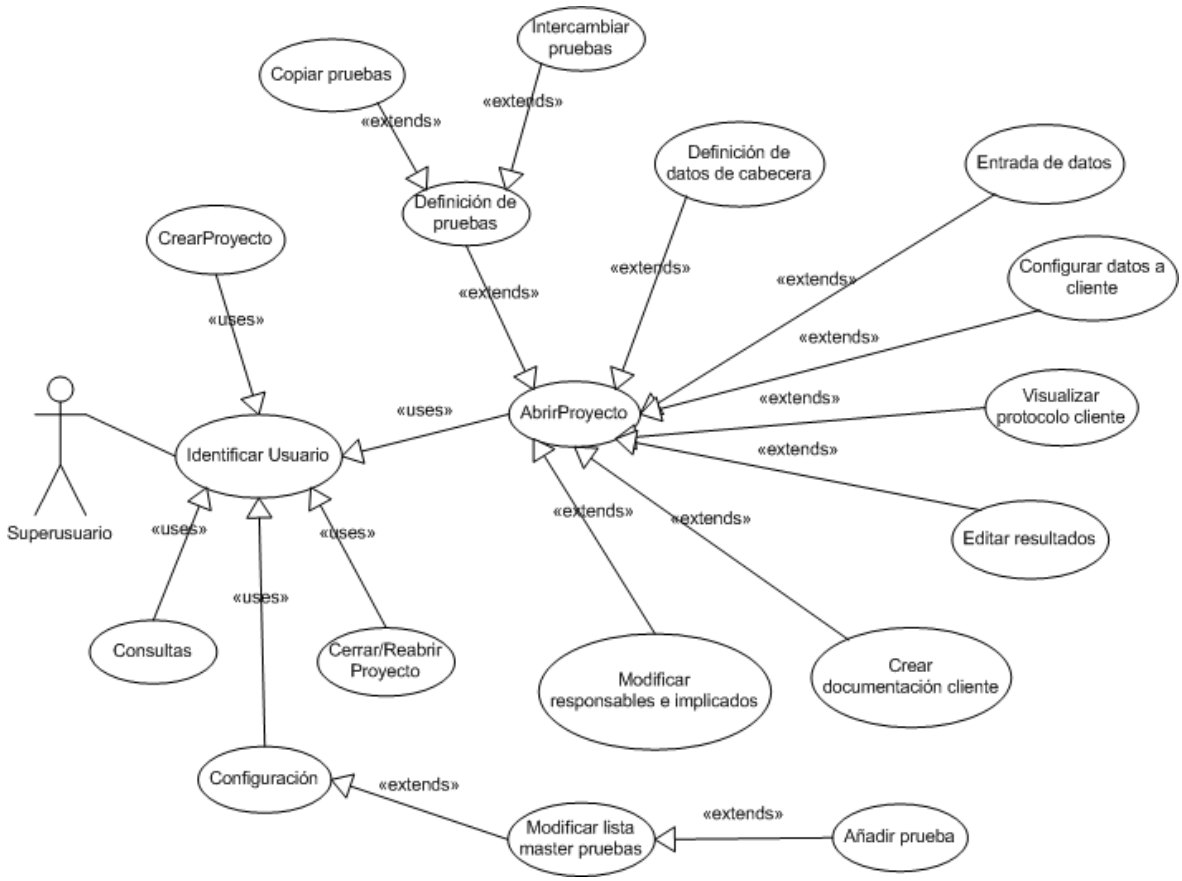


Imagen 4.2. Casos de uso del rol 'Administrador' de la aplicación.

4.2.1. Identificar Usuario

Este caso de uso no consiste en identificar usuario como se hace habitualmente, introduciendo nombre y contraseña. En este caso, al encontrarse la aplicación en la Intranet de la empresa, para acceder a ésta habrá que estar autenticado en el sistema contra el directorio activo. Por tanto, este caso de uso lo que hará será tomar el nombre de usuario del usuario que se conecte a la aplicación, y comprobar a qué unidad de negocio pertenece (Hydro o CIM), y qué permisos tiene.

- Actores: Todos los perfiles

- Resumen: Comprueba a qué unidad de negocio o línea pertenece un usuario, y los permisos de los que dispone.
- Precondición: El usuario tiene acceso a la Intranet de la empresa.
- Postcondición: Comprueba a qué unidad de negocio o línea pertenece un usuario, y los permisos de los que dispone.

1. Sistema: Consulta el nombre que tiene el usuario que se ha conectado a la aplicación, dentro del sistema, y posteriormente comprueba la unidad de negocio a la que pertenece y el rol que tiene en la aplicación.

4.2.2. Crear

Muestra los pedidos de la línea del usuario (Hydro o CIM) que no estén dados de alta en la base de datos del **GBP**. Una vez seleccionado uno de ellos, se mostrará la información relativa a dicho pedido, obteniendo una lista del personal de la empresa que podría tener relación con él. Permitirá además la asignación de una o varias de esas personas como responsables o implicados en el proyecto. Finalmente, se posibilitará dar de alta este pedido en el **GBP**, estableciendo los responsables e implicados en el pedido tal y como indicó el usuario.

- Actores: Superusuario, Creador
- Resumen: Da de alta un proyecto en el **GBP**.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y pertenece a un rol apropiado.
- Postcondición: Da de alta un proyecto de la línea del usuario que no exista todavía en la base de datos del **GBP** y le asigna distintos responsables e implicados.

Escenario principal (o curso normal de los eventos):

1. Usuario: El usuario quiere dar de alta un pedido.
2. Sistema: Muestra la lista de pedidos de la línea del usuario que aún no están dados de alta en el gestor. Para ello, se conectará a la base de datos específica de cada unidad de negocio
3. Usuario: Elige uno de esos pedidos.
4. Sistema: Muestra información relacionada con el pedido y una lista del personal que pueda ser responsable o implicado del proyecto. Por

defecto siempre habrá al menos un responsable, el usuario que crea/da de alta el proyecto.

5. Usuario: Elige los restantes responsables e implicados si los hubiera y selecciona la opción crear.
6. Sistema: Pide confirmación al usuario
7. Usuario: Confirma la operación.
8. Sistema: Añade a la base de datos el proyecto, junto con los responsables e implicados seleccionados. También almacenará los números de serie de las máquinas asociadas a ese pedido. Informa al usuario del resultado de la operación y le redirecciona al menú principal.

4.2.3. Abrir

El usuario elige un proyecto dado de alta y abierto en el gestor, para posteriormente seleccionar un número de serie[7] correspondiente a dicho pedido. Un pedido abierto en el gestor quiere decir que es un proyecto activo en el cual se pueden realizar operaciones. En un proyecto "cerrado" no se podrá realizar ninguna operación, aunque dispone de la opción "Reabrir". Esto se representará en la base de datos mediante un campo booleano de nombre "Cerrado" en la tabla de proyectos.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Selecciona un número de serie de un pedido abierto en el **GBP**.
- Precondición: El usuario tiene acceso a la Intranet de la empresa.
- Postcondición: Devuelve el número de serie de un pedido abierto en el **GBP**.

Escenario principal (o curso normal de los eventos):

1. Usuario: El usuario quiere seleccionar un número de serie de un pedido.
2. Sistema: Muestra la lista de pedidos que están dados de alta en el gestor.
3. Usuario: Selecciona un pedido.
4. Sistema: Muestra la lista de números de serie de máquinas asociadas a ese proyecto.
5. Usuario: Selecciona el número de serie deseado.

[7] Un número de proyecto o pedido, puede contener varias máquinas, cada una de las cuales tiene su número de serie.

4.2.4. Definición de pruebas

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso "Abrir". Una vez elegidos un número de proyecto y un número de serie en el caso de uso "Abrir", define las pruebas que se le quieren asignar a esa máquina.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Establece una serie de pruebas que se le realizarán a la máquina en el Banco de Pruebas.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un pedido "abierto" y un número de serie.
- Postcondición: Asigna una lista de pruebas al número de serie dado.

Escenario principal (o curso normal de los eventos):

1. Usuario: Inicia la definición de pruebas.
2. Sistema: Muestra una lista de pruebas dividiéndolas en 2 secciones, las pruebas asignadas de la máquina, y las pruebas posibles.
3. Usuario: Añade una o varias pruebas posibles a pruebas asignadas, asignándoles un tipo a cada prueba. También puede quitar pruebas asignadas. Cuando finalice el proceso y tenga exactamente las pruebas que desea asignar a la máquina, guardará.
4. Sistema: Pedirá confirmación al usuario.
5. Usuario: Confirma la operación.
6. Sistema: Crea una revisión de la lista de pruebas de la máquina y las asigna a ese número de serie. Informa al usuario del resultado de la transacción.

Extensiones (o cursos alternativos):

4. Sistema: Detecta que hay alguna prueba en la lista de pruebas a designar que no tiene tipo asignado. Informa al usuario de cuáles son esas pruebas.
5. Usuario: Asigna un tipo a las pruebas que lo requieran.
6. Sistema: Pedirá confirmación al usuario.
7. Usuario: Confirma la operación.
8. Sistema: Crea una revisión de la lista de pruebas de la máquina y las asigna a ese número de serie. Informa al usuario del resultado de la transacción.

4.2.5. Copiar pruebas

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso "Definición de pruebas". El usuario visualiza las pruebas asignadas a una máquina, y además tiene la opción de elegir otro número de serie para asignarle esas mismas pruebas con ese mismo tipo.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Asigna las mismas pruebas de un número de serie a otro número de serie distinto.
- Precondición: El usuario tiene acceso a la Intranet de la empresa, ha seleccionado previamente un pedido abierto y un número de serie y le ha asignado una serie de pruebas.
- Postcondición: Copia las pruebas de una máquina en otra máquina distinta.

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide los números de proyectos abiertos en el **GBP**.
2. Sistema: Muestra la lista de proyectos.
3. Usuario: Selecciona uno de esos proyectos.
4. Sistema: Muestra los números de serie asociados al proyecto seleccionado.
5. Usuario: Selecciona un número de serie y elige "Copiar".
6. Sistema: Comprueba las pruebas que tiene asignadas el primer número de serie, crea una revisión de la lista de pruebas de la máquina en la que se van a copiar las pruebas y las asigna a ese número de serie. Informa al usuario del resultado de la transacción.

4.2.6. Intercambiar pruebas

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso "Definición de pruebas". El usuario visualiza las pruebas asignadas a una máquina, y además tiene la opción de elegir otro número de serie para permutar las pruebas entre ellas, es decir, asignar las pruebas del generador A al generador B, y asignar las pruebas del generador B al generador A. Este caso de uso podrá realizarse siempre y cuando aún no se haya comenzado con las pruebas del generador A o B.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Asigna las pruebas de un generador A a uno B, y viceversa.
- Precondición: El usuario tiene acceso a la Intranet de la empresa, ha seleccionado previamente un pedido abierto y un número de serie y le ha asignado una serie de pruebas y éstas aún no han comenzado.
- Postcondición: Asigna al generador A las pruebas que en un inicio estaban asignadas al generador B, y asigna al generador B las pruebas que estaban asignadas al generador A.

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide los números de proyectos abiertos en el **GBP**.
2. Sistema: Muestra la lista de proyectos.
3. Usuario: Selecciona uno de esos proyectos.
4. Sistema: Muestra los números de serie asociados al proyecto seleccionado.
5. Usuario: Selecciona un número de serie y elige “Permutar”.
6. Sistema: Comprueba las pruebas que tiene asignadas el primer número de serie, crea una revisión de la lista de pruebas de la máquina en la que se van a copiar las pruebas y las asigna a ese número de serie. Comprueba las pruebas que tenía asignado el segundo número de serie en su penúltima revisión, crea una revisión de la lista de pruebas del primer número de serie y asigna las pruebas a ese número de serie. Informa al usuario del resultado de la transacción.

4.2.7. Definición de datos de cabecera

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Abrir”. Permite visualizar y editar cabeceras[8] de las pruebas de las máquinas.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Visualiza o edita los datos de las cabeceras de un proyecto.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un proyecto.
- Postcondición: Muestra o edita los datos de las cabeceras de un proyecto.

[8] Una cabecera es un resumen con las características más importantes de un proyecto

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide las cabeceras que tiene el proyecto.
2. Sistema: Muestra la lista de cabeceras del generador.
3. Usuario: Elige una cabecera.
4. Sistema: Mostrará la información relativa a la cabecera, permitiendo introducir la información necesaria para completar la información mostrada por el sistema. Para ello se mostrará un formulario.
5. Usuario: Rellena el formulario y pulsa guardar.
6. Sistema: Pide confirmación al usuario.
7. Usuario: Confirma la operación.
8. Sistema: Define la cabecera con los datos introducidos.

Extensiones (o cursos alternativos):

8. Sistema: Detecta que hay campos mal introducidos en el formulario e informa al usuario de cuáles son.
9. Usuario: Corrige los datos incorrectos.
10. Sistema: Pide confirmación al usuario.
11. Usuario: Confirma la operación.
12. Sistema: Define la cabecera con los datos introducidos.

4.2.8. Entrada de datos

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Abrir”. Permite introducir los datos relativos a cada prueba obtenidos en el Banco de Pruebas.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Asigna una serie de valores a una prueba, que serán los datos obtenidos en el Banco de Pruebas.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un pedido, un número de serie y el generador tiene pruebas y cabeceras definidas.
- Postcondición: Muestra o edita los datos de las pruebas de un número de serie concreto.

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide las pruebas que tiene asignadas la máquina.
2. Sistema: Muestra la lista de pruebas del número de serie.
3. Usuario: Elige una prueba.
4. Sistema: Dará la opción al usuario de realizar una nueva entrada de datos de la prueba, o sobreescribir una prueba anterior realizada.
5. Usuario: Elige una de las opciones.
6. Sistema: Muestra el formulario de entrada de datos.
7. Usuario: Rellena el formulario y pulsa guardar.
8. Sistema: Pide confirmación al usuario.
9. Usuario: Confirma la operación.
10. Sistema: Introduce en la base de datos los datos introducidos e informa al usuario del resultado de la transacción.

Extensiones (o cursos alternativos):

10. Sistema: El sistema detecta campos incorrectos en el formulario de entrada e informa al usuario.
11. Usuario: Corrige los datos incorrectos.
12. Sistema: Pide confirmación al usuario.
13. Usuario: Confirma la operación.
14. Sistema: Introduce en la base de datos los datos introducidos e informa al usuario del resultado de la operación.

4.2.9. Configurar datos a cliente

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Abrir”. Decidir qué resultados de pruebas se van a enviar al cliente.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Dado el total de pruebas realizadas, se seleccionarán aquellas de las que se quiera enviar los resultados al cliente.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un pedido, un número de serie y el generador tiene pruebas y cabeceras definidas.
- Postcondición: Establece qué datos resultantes de las pruebas se le enviarán al cliente.

Escenario principal (o curso normal de los eventos):

1. Usuario: El usuario solicita las pruebas que se le han realizado al generador.
2. Sistema: Muestra las pruebas que se le han realizado al generador.
3. Usuario: Elige una o varias de las opciones ofrecidas y pulsa “Guardar”.
4. Sistema: Pide confirmación de la operación al usuario.
5. Usuario: Confirma la acción.
6. Sistema: Introduce en la base de datos los cambios deseados e informa al usuario del resultado de la transacción.

4.2.10. Visualizar protocolo cliente

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Abrir”. Se visualizan los datos que se le van a enviar al cliente. Es decir, se visualiza lo que se ha configurado en el caso de uso anterior.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Se visualizan los datos que se le van a enviar al cliente.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un pedido, un número de serie y el generador tiene pruebas y cabeceras definidas.
- Postcondición: Muestra por pantalla los resultados de las pruebas que se desean enviar al cliente.

Escenario principal (o curso normal de los eventos):

1. Usuario: El usuario solicita la información que se ha configurado como la que se va a enviar al cliente.
2. Sistema: Muestra dicha información por pantalla.

4.2.11. Editar resultados

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Abrir”. Permite editar los resultados de las pruebas realizadas al generador en el Banco de Pruebas.

- Actores: Superusuario, Creador, Responsable.
- Resumen: Se modifican los datos de las pruebas.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un pedido, un número de serie y el generador tiene pruebas asignadas.
- Postcondición: Se modifican los datos de las pruebas que desee el usuario.

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide las pruebas que tiene asignadas la máquina.
2. Sistema: Muestra la lista de pruebas del número de serie.
3. Usuario: Elige un número de serie.
4. Sistema: Mostrará la fecha y hora de la prueba a optimizar.
5. Usuario: Elige una de las opciones.
6. Sistema: Muestra el formulario de entrada de datos.
7. Usuario: Rellena el formulario y pulsa guardar.
8. Sistema: Pide confirmación al usuario.
9. Usuario: Confirma la operación.
10. Sistema: Actualiza la base de datos con los datos introducidos e informa al usuario de la transacción.

Extensiones (o cursos alternativos):

8. Sistema: El sistema detecta campos incorrectos en el formulario de entrada e informa al usuario.
9. Usuario: Corrige los datos incorrectos.
10. Sistema: Pide confirmación al usuario.
11. Usuario: Confirma la operación.
12. Sistema: Actualiza la base de datos con los datos introducidos e informa al usuario de la transacción.

4.2.12. Crear documentación cliente

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Abrir”. Se encargará de generar los documentos que se entregarán al cliente.

- Actores: Superusuario, Creador, Responsable.
- Resumen: El usuario elige una serie de documentos a generar y el sistema los genera.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un pedido, un número de serie y el generador tiene pruebas y cabeceras asignadas.
- Postcondición: Generará los documentos seleccionados por el usuario que se enviarán al cliente, y mostrará un enlace a la carpeta que los contiene.

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide el conjunto de documentos que pueden ser generados.
2. Sistema: Muestra la lista de documentos.
3. Usuario: Elige uno o varios documentos a generar y selecciona “Crear”.
4. Sistema: Pide confirmación al usuario.
5. Usuario: Confirma la operación.
6. Sistema: Genera los documentos seleccionados, informa al usuario del resultado de la operación y, en caso de éxito, también mostrará un enlace a la carpeta que los contiene.

4.2.13. Modificar responsables e implicados

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Abrir”. Permite modificar los responsables e implicados de un pedido.

- Actores: Superusuario, Creador, Responsable.
- Resumen: El usuario edita los responsables e implicados de un proyecto y guarda las modificaciones.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y ha seleccionado previamente un pedido.
- Postcondición: Modifica el conjunto de responsables e implicados que tiene un proyecto.

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide el conjunto de responsables e implicados que tienen relación con el proyecto seleccionado.
2. Sistema: Muestra la lista de responsables e implicados que pertenecen a dicho pedido.

3. Usuario: Pide la lista del personal que puede tener este estado en los proyectos.
4. Sistema: Muestra la lista total del personal que pueda ser responsable o implicado.
5. Usuario: Modifica a su elección el personal asignado al proyecto y selecciona “Guardar”.
6. Sistema: Pide confirmación al usuario.
7. Usuario: Confirma la operación.
8. Sistema: Actualiza los responsables e implicados del proyecto e informa al usuario del resultado de la transacción.

4.2.14. Cerrar/Reabrir proyecto

Muestra los pedidos de la línea del usuario (Hydro o CIM) en 2 conjuntos: por un lado los que estén abiertos, y por otro lado los que estén cerrados. El usuario podrá elegir un proyecto y cambiarle el estado, es decir, si elige uno de los abiertos, podrá cerrarlo, y si elige uno de los cerrados, podrá reabrirlo.

- Actores: Superusuario, Creador.
- Resumen: El usuario cierra o reabre proyectos.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y hay pedidos abiertos y/o cerrados.
- Postcondición: Cambia el estado del proyecto seleccionado.

Escenario principal (o curso normal de los eventos):

1. Usuario: Pide el conjunto de proyectos de su línea dados de alta en el **GBP**.
2. Sistema: Muestra la lista de proyectos divididos en: abiertos y cerrados.
3. Usuario: Elige uno de los proyectos y selecciona cambiar estado.
4. Sistema: Pide confirmación al usuario.
5. Usuario: Confirma la operación.
6. Sistema: Actualiza el estado del pedido en la base de datos del **GBP** e informa al usuario del resultado de la transacción.

4.2.15. Consultas

Este caso de uso está todavía pendiente de definición. No está claro qué tipo de consultas se realizarán. De momento, la única consulta definida es visualizar las pruebas asignadas a generadores pertenecientes a otra unidad de negocio, es decir, que un usuario de Hydro pueda visualizar la configuración de pruebas que tiene un generador perteneciente a CIM, y viceversa, eso sí, en modo de sólo lectura. La idea es facilitar el intercambio de información entre líneas, pero en ningún caso un usuario de una línea podrá modificar nada relativo a otra. De todas formas, en el futuro la idea es que se puedan realizar un buen número de consultas, de una forma sencilla.

- Actores: Todos los perfiles.
- Resumen: El usuario visualiza información referente a las pruebas asignadas a un generador de otra línea de negocio.
- Precondición: El usuario tiene acceso a la Intranet de la empresa, hay pedidos abiertos y los números de serie asociados a él tienen números de serie asignados.
- Postcondición: Muestra las pruebas asignadas a un generador perteneciente a otra línea.

Escenario principal (o curso normal de los eventos):

1. Usuario: El usuario solicita la lista de proyectos.
2. Sistema: El sistema muestra la lista de proyectos.
3. Usuario: Solicita la lista de generadores pertenecientes a dicho pedido.
4. Sistema: Muestra la lista de generadores.
5. Usuario: Selecciona uno de ellos.
6. Sistema: Muestra las pruebas asignadas a este número de serie.

4.2.16. Configuración

Este caso de uso también está pendiente de definición. De momento solamente se contempla la modificación de la lista master de pruebas.

4.2.17. Modificar lista master de pruebas

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso “Configuración”. Este caso de uso crea, o en caso de que exista, modifica la lista

master de pruebas. En otras palabras, el usuario de la línea define una lista de pruebas y una configuración por defecto para su línea. Para ello seleccionará una serie de pruebas de las definidas por el master, y les asignará un tipo[9]. Además, permitirá visualizar las configuraciones anteriores.

- Actores: Superusuario.
- Resumen: El usuario modifica la información referente a la revisión de la master de pruebas de la línea.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y rol Superusuario.
- Postcondición: Crea una nueva revisión de la master de pruebas para la línea del usuario.

Escenario principal (o curso normal de los eventos):

1. Usuario: El usuario solicita la revisión de la master de pruebas de su línea.
2. Sistema: Muestra la revisión de la lista de pruebas. También indicará qué número de revisión es el último.
3. Usuario: Modifica la revisión y pulsa “Guardar”.
4. Sistema: Pide confirmación al usuario.
5. Usuario: Confirma la operación
6. Sistema: Crea una nueva revisión del master de pruebas de la línea e informa al usuario del resultado de la transacción.

Extensiones (o cursos alternativos):

3. Usuario: Pide al sistema el número de todas las revisiones anteriores.
4. Sistema: Muestra todas las revisiones anteriores.
5. Usuario: Elige una de ellas.
6. Sistema: Muestra la configuración de pruebas de esa revisión, de forma editable.

[9] El tipo de prueba determina si es prueba relativa al cliente o a la empresa, y la situación en la que se hará esta prueba a un generador

4.2.18. Añadir prueba

Este caso de uso comienza cuando el usuario se encuentra dentro del caso de uso "Modificar lista master de pruebas". Añade una prueba al master de pruebas.

- Actores: Superusuario.
- Resumen: El usuario añade una nueva prueba al master de pruebas del **GBP**.
- Precondición: El usuario tiene acceso a la Intranet de la empresa y rol Superusuario.
- Postcondición: Añade una nueva prueba al master de pruebas.

Escenario principal (o curso normal de los eventos):

1. Usuario: El usuario solicita añadir prueba.
2. Sistema: Muestra un formulario en el que se le pedirá el nombre de la nueva prueba.
3. Usuario: Introduce el nombre de la prueba y pulsa "Guardar".
4. Sistema: Añade la nueva prueba al master de pruebas e informa al usuario del resultado de la transacción.

Extensiones (o cursos alternativos):

4. Sistema: El sistema detecta que el nombre de la prueba es vacío e informa al usuario.
5. Usuario: Corrige el error.
6. Sistema: Añade la nueva prueba al master de pruebas e informa al usuario del resultado de la transacción.

4.3. Modelo de Dominio

Por razones de claridad y mejor comprensión, el modelo de dominio que presentamos en la Figura es una versión muy simplificada del modelo real, recogiendo solamente las entidades fundamentales de la aplicación.

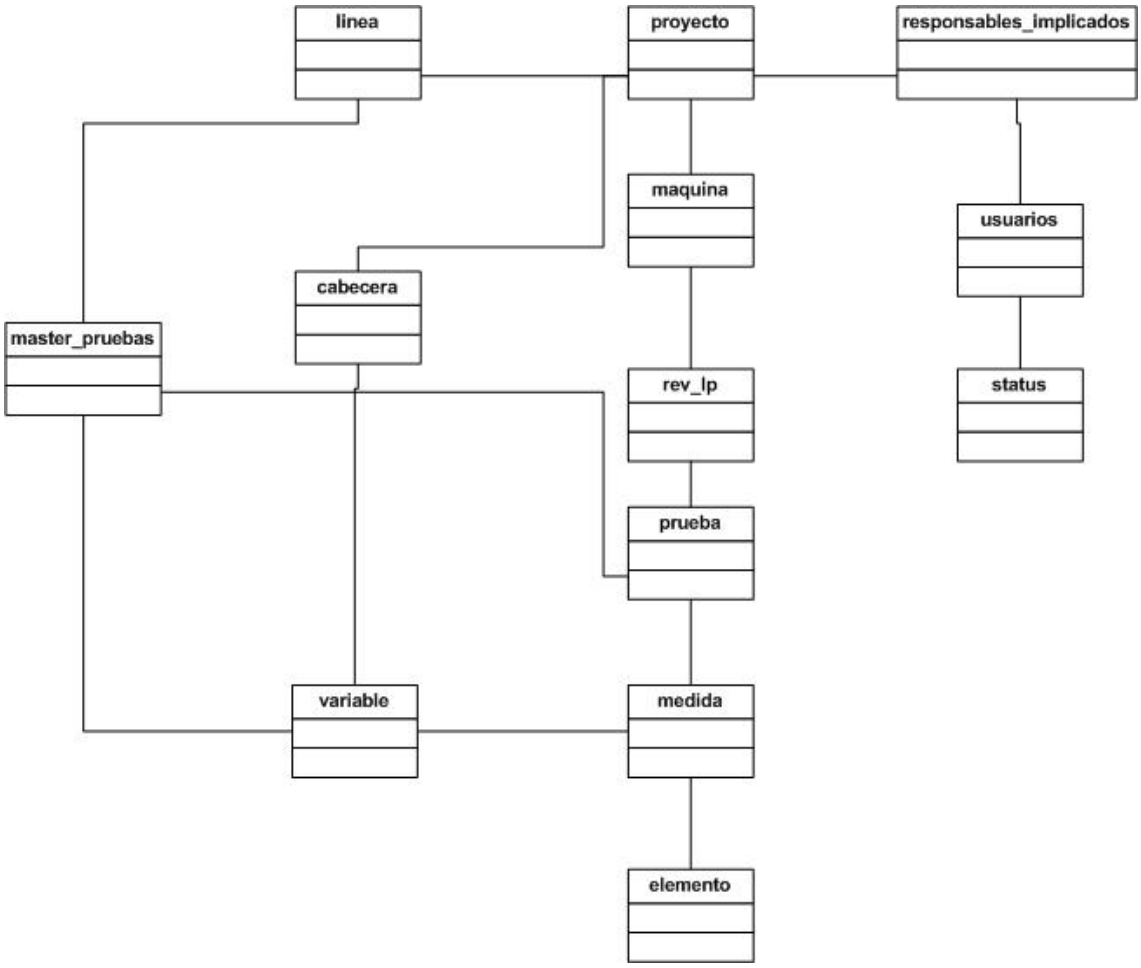


Imagen 4.3. Modelo de dominio

En el siguiente punto procederé a explicar el modelo de dominio. Preservando la privacidad de empresa, mostraré algunos de los atributos más importantes de cada entidad, y cual es su función en el proceso.

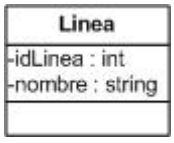


Imagen 4.4. Atributos de 'Linea'

La imagen 4.4 representa las distintas líneas o unidades de negocio que utilizarán la aplicación. En el momento actual son solamente 'Hydro' y 'CIM' pero en el futuro la intención es que también utilicen esta aplicación tanto 'Wind Power' como 'I+D'.

Imagen 4.5. Atributos de 'Proyecto'

Es una de las tablas más importantes de toda la aplicación. En ella almacenaremos el string que identificará de forma unívoca el proyecto, en los campos nProyecto y posición. El campo 'nRevHDCreacion' almacenará la última revisión que tenía la Hoja de Diseño del proyecto cuando se da de alta el proyecto en el **GBP**. En este punto conviene aclarar que las Hojas de Diseño son los documentos que contienen todas las especificaciones eléctricas y mecánicas del generador. El campo 'idLinea' determinará a qué línea pertenece el proyecto.

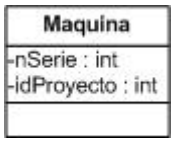


Imagen 4.6. Atributos de 'Maquina'

En la tabla 'Maquina' se almacenarán los números de serie de todos los generadores que pertenezcan a un proyecto. El atributo 'nSerie' corresponde al número de serie de un generador concreto, y el campo 'idProyecto' indica a qué proyecto pertenece dicho generador.

Imagen 4.7. Atributos de 'master_pruebas'

Esta tabla también es muy importante, ya que en ella se almacenarán todas las pruebas que se pueden realizar a las máquinas en Indar Electric S.L. En el campo nombre se almacena el nombre de la prueba, por ejemplo 'Calentamientos en corto a X%'. En el campo activa se almacenará un booleano que indica si dicha prueba está activa o inactiva. Una prueba está inactiva cuando ya no se realiza nunca, y está activa en caso contrario. El campo 'idLinea' determinará a qué línea pertenece la

prueba, ya que puede darse el caso de que una prueba se realice solamente en una unidad de negocio, o que una prueba esté activa para una unidad de negocio e inactiva para otra.

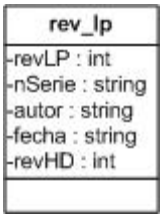


Imagen 4.8. Atributos de ‘rev_lp’

Esta tabla se utiliza para tener el control de las pruebas que se le asignan a un número de serie. Cada vez que se asignen pruebas a un número de serie, se creará una entrada en esta tabla con la información que se indica en los campos mostrados. El campo ‘revLP’ es el número de revisión de lista de pruebas que tiene un número de serie. El campo ‘nSerie’ representa el número de serie al cual se le han asignado pruebas. El campo ‘autor’ corresponde al usuario de la aplicación que ha asignado una lista de pruebas a un generador. La fecha indica la fecha y la hora en la que tuvo lugar dicha asignación, y el campo revHD indicará la última revisión de la Hoja de Diseño del proyecto en el momento de asignarse las pruebas. Obsérvese que el campo ‘revHD’ no tiene por qué coincidir con el campo ‘nRevHDCreacion’ de la tabla ‘Proyecto’, ya que si se da de alta un proyecto en el **GBP** y posteriormente se modifica la Hoja de Diseño, cuando se asignen pruebas a un número de serie de ese proyecto, el campo ‘nRevHDCreacion’ será distinto (y menor) que el campo ‘revHD’.

De esta forma, se mantiene un registro de todas las pruebas que se le han asignado a un número de serie y de las condiciones en las que se asignaron. Por simplicidad en el modelo de dominio se ha mostrado solamente esta tabla, para ilustrar el procedimiento que se ha utilizado para permitir una correcta trazabilidad y futuras consultas sobre los datos almacenados, pero en realidad este procedimiento u otros similares se utilizan en prácticamente todas las operaciones que se realicen sobre la base de datos.

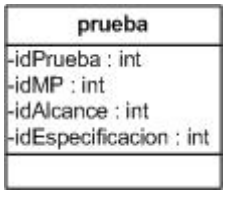


Imagen 4.9. Atributos de ‘prueba’

En esta tabla se almacenarán todas las pruebas que formen parte de una determinada lista de pruebas de un número de serie. El campo ‘idPrueba’ es un identificador único para cada prueba, el campo ‘idMP’ indica qué prueba de la tabla master de pruebas es, el campo ‘idAlcance’ indica qué tipo de prueba es, si ‘Interna’ o ‘Cliente’. Las pruebas internas son aquellas que se realizan a nivel interno en la empresa para garantizar la propia calidad del producto comprado, y las pruebas cliente son aquellas en las que el cliente puede estar presente cuando se realizan. El campo ‘idEspecificacion’ indica qué tipo de especificación tiene la prueba. Hay 3 tipos posibles: ‘Rutina’, ‘Tipo’ y ‘Especial’.

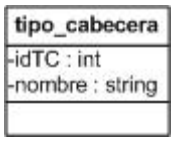


Imagen 4.10. Atributos ‘tipo_cabecera’

Una cabecera de proyecto es la información general más relevante de dicho proyecto, y puede ser de distintos tipos: ‘General’, ‘Apoyos’, ‘Electroventiladores’, ‘Refrigeración’, ‘Excitación’ o ‘Varios’, En esta última categoría se encontrarán aquellos datos de cabecera que no pertenezcan a ninguno de los anteriores apartados.

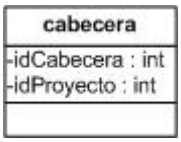


Imagen 4.11. Atributos cabecera

Relaciona una cabecera con un proyecto.

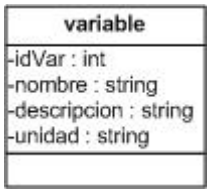


Imagen 4.12. Atributos ‘variable’.

En esta tabla se encuentran todas las variables que forman parte de la aplicación. Se utiliza para generar todos los formularios, de forma que se soliciten los datos adecuados en cada caso. La razón de ser de esta lógica, es que uno de los requisitos

de la empresa consiste en que cuando se quiera modificar algún formulario de forma que se añadan o quiten variables, solamente haya que añadirla en la base de datos, y la aplicación siga funcionando sin necesidad de modificar el código. Este ha sido uno de los puntos de mayor dificultad del desarrollo.

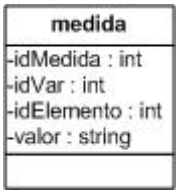


Imagen 4.13. Atributos ‘medida’

En esta tabla se almacenarán todos los valores introducidos por el usuario relativos a las cabeceras y a las pruebas. Cada medida llevará asociada la variable a la que corresponde y el elemento del que forma parte.

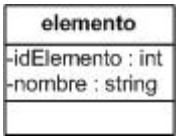


Imagen 4.14. Atributos ‘elemento’

Un elemento es cualquier parte de un generador del cual se introduzcan datos, por ejemplo, elementos son ‘Rotor’, ‘Estator’, ‘Rotor-excitatriz’, ‘Estator-excitatriz’, ‘PT-100’, faseUV, faseVW, faseWU, etc…

5. ANÁLISIS

A continuación se detallará el análisis de cada uno de los casos de uso presentados en la sección anterior.

5.1. Identificar Usuario

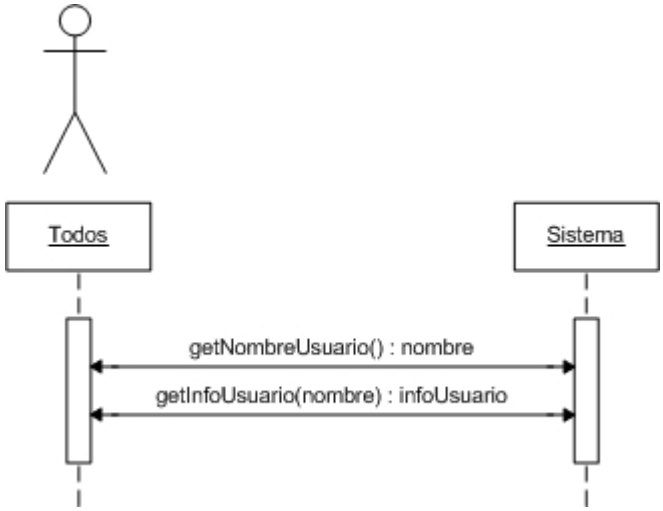


Imagen 5.1. Diagrama de secuencia de “Identificar Usuario”

5.1.1. Contrato de getNombreUsuario

Name: getNombreUsuario()

Responsabilities: Obtiene del sistema el login del usuario que se ha conectado a la aplicación.

Preconditions: True.

Postconditions:

Salida: Devuelve el nombre del usuario.

5.1.2. Contrato de getInfoUsuario

Name: getInfoUsuario(nombre) : infoUsuario

Responsabilities: Obtiene la información más relevante del usuario pasado como parámetro (nombre, línea, status)

Preconditions: El usuario tiene un login asociado.

Postconditions:

Salida: Devuelve información perteneciente al usuario.

5.2. CrearProyecto

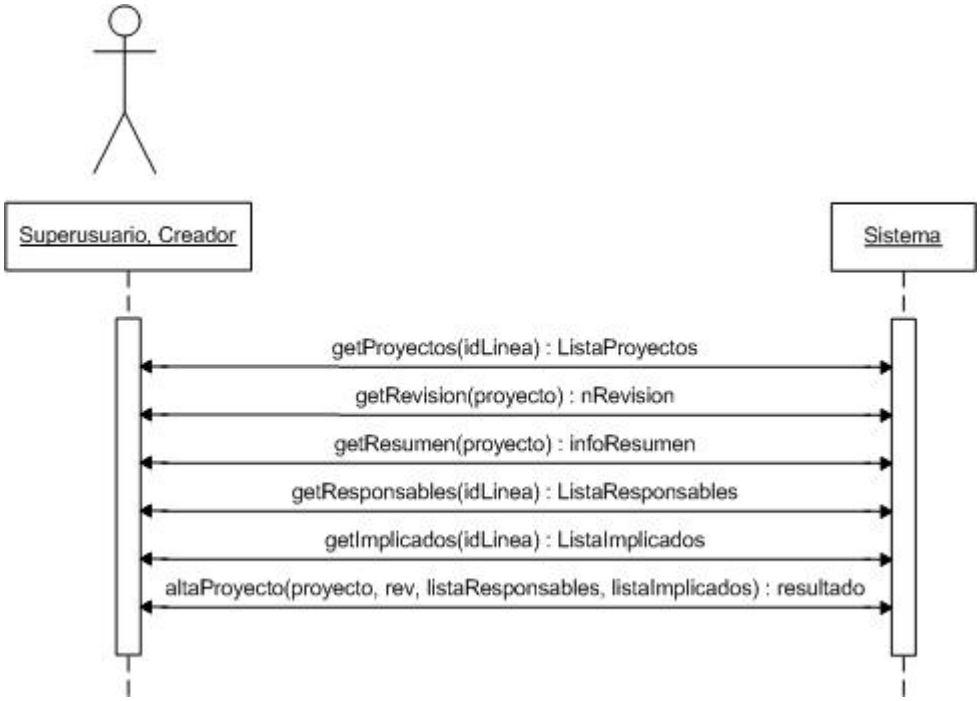


Imagen 5.2. Diagrama de secuencia de “CrearProyecto”

5.2.1. Contrato de getProyectos

Name: getProyectos(idLinea) : ListaProyectos.

Responsabilities: Obtiene la lista de proyectos de la unidad de negocio que aún no están dados de alta en el **GBP**.

Preconditions: idLinea es un idLinea válido.

Postconditions:

Salida: Devuelve los proyectos de la unidad de negocio que aún no están dados de alta en el **GBP**.

5.2.2. Contrato de getRevision

Name: getRevision(proyecto) : nRevision.

Responsabilities: Obtiene última revisión de un proyecto.

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve un número entero que es la última revisión del proyecto pasado como parámetro.

5.2.3. Contrato de getResumen

Name: getResumen(proyecto) : infoResumen.

Responsabilities: Obtiene información resumen del proyecto seleccionado. Esta información consistirá en el tipo de máquina, el nombre de la central, el número de unidades y la configuración de la máquina.

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve la información resumen del proyecto.

5.2.4. Contrato de getResponsables

Name: getResponsables(proyecto) : ListaResponsables.

Responsabilities: Obtiene la lista de usuarios que pueden ser responsables de un proyecto.

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve una lista de responsables.

5.2.5. Contrato de getImplicados

Name: getImplicados(proyecto) : ListaImplicados.

Responsabilities: Obtiene la lista de usuarios que pueden estar implicados en un proyecto.

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve una lista de implicados.

5.2.6. Contrato de altaProyecto

Name: altaProyecto(proyecto, rev, listaResponsables, listaImplicados) : resultado.

Responsabilities: Da de alta un proyecto en el **GBP**.

Preconditions: Los parámetros existen en el sistema.

Postconditions: Introduce en la base de datos del **GBP** el proyecto seleccionado, junto con la revisión en la que se encuentra, y las listas de responsables e implicados seleccionados por el usuario. Además, también dará de alta las máquinas pertenecientes a dicho proyecto.

Salida: Devuelve un mensaje indicando cuál ha sido el resultado de la operación.

5.3. Abrir

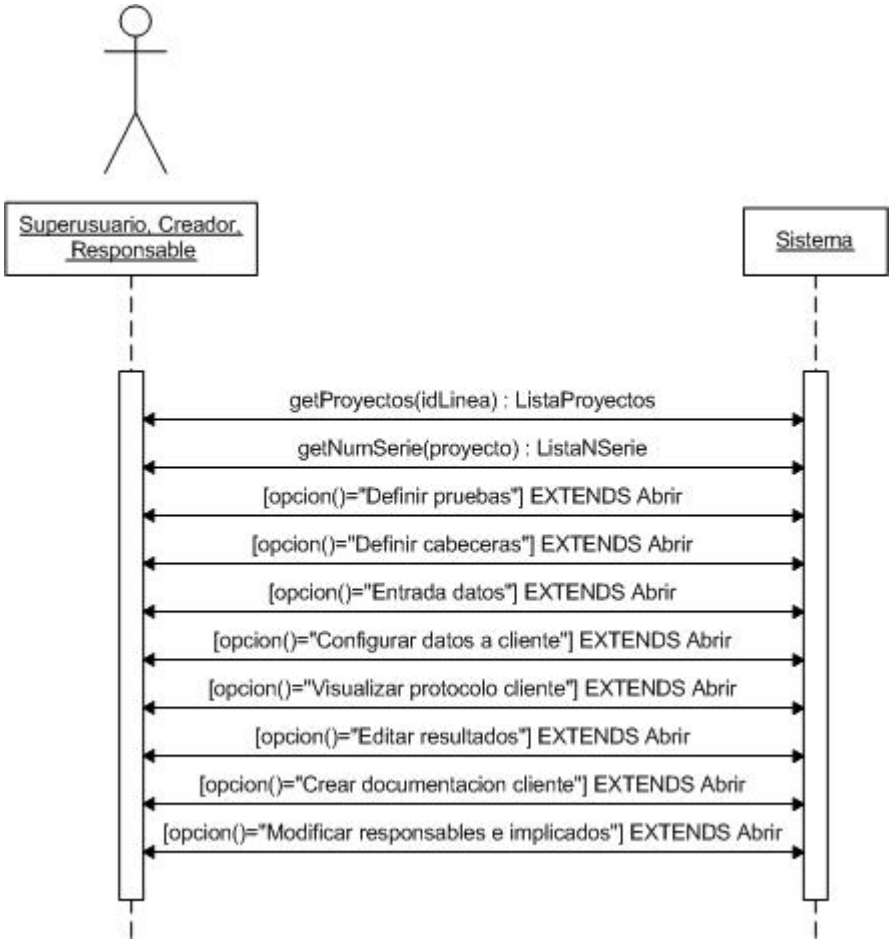


Imagen 5.3. Diagrama de secuencia de “AbrirProyecto”

5.3.1. Contrato de getProyectos

Name: getProyectos(idLinea) : ListaProyectos.

Responsabilities: Obtiene la lista de proyectos de la línea del usuario que están dados de alta en el gestor.

Preconditions: El parámetro idLinea es un identificador de línea válido.

Postconditions:

Salida: Devuelve una lista de proyectos relativos a la línea del usuario que están dados de alta en el gestor.

5.3.2. Contrato de getNumSerie

Name: getNumSerie(proyecto) : ListaNSerie.

Responsabilities: Obtiene la lista de los números de serie de los generadores que pertenecen al proyecto seleccionado.

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve una lista de números de serie.

5.4. Definir pruebas

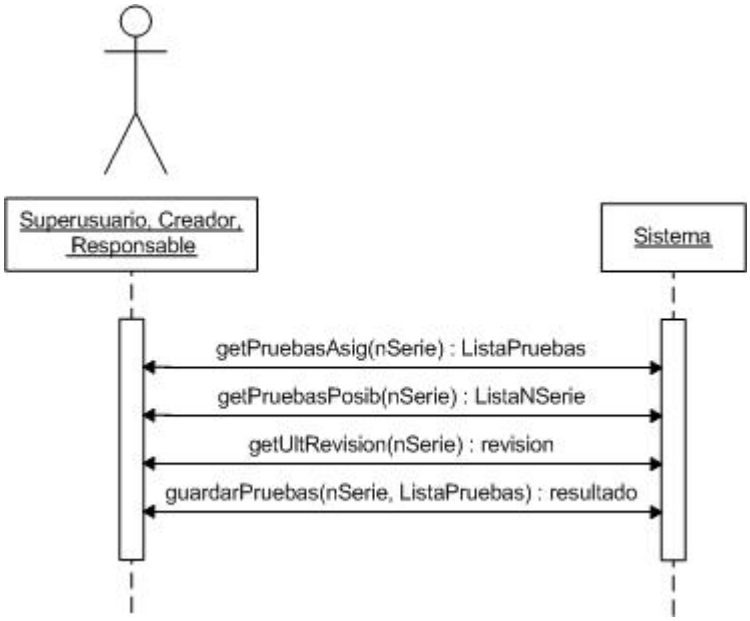


Imagen 5.4. Diagrama de secuencia de “Definir pruebas”

5.4.1. Contrato de getPruebasAsig

Name: getPruebasAsig(nSerie) : ListaPruebas.

Responsabilities: Obtiene la lista de pruebas asignadas que tiene ese número de serie.

Preconditions: El parámetro nSerie es un número de serie válido.

Postconditions:

Salida: Devuelve una lista de pruebas que tiene asignada la máquina con el número de serie nSerie.

5.4.2. Contrato de getPruebasPosib

Name: getPruebasPosib(nSerie) : ListaPruebas.

Responsabilities: Obtiene la lista de pruebas posibles que se le pueden hacer a un generador.

Preconditions: El parámetro nSerie es un número de serie válido.

Postconditions:

Salida: Devuelve una lista de pruebas que se le pueden hacer a una máquina con el número de serie nSerie.

5.4.3. Contrato de getUltRevision

Name: getUltRevision(nSerie) : revision.

Responsabilities: Obtiene la última revisión de la lista de pruebas que tiene una máquina.

Preconditions: El parámetro nSerie es un número de serie válido.

Postconditions:

Salida: Devuelve la última revisión de la lista de pruebas que tiene asignada la máquina con el número de serie nSerie.

5.4.4. Contrato de guardarPruebas

Name: guardarPruebas(nSerie, listaPruebas) : resultado.

Responsabilities: Asigna una lista de pruebas seleccionadas por el usuario, al número de serie nSerie.

Preconditions: El parámetro nSerie es un número de serie válido y la lista de pruebas la componen pruebas válidas.

Postconditions: Crea una nueva revisión de la lista de pruebas para el número de serie dado, y le asigna todas las pruebas de listaPruebas.

Salida: Asigna una lista de pruebas seleccionadas por el usuario, al número de serie nSerie, y muestra el resultado de la operación.

5.5. Copiar Pruebas

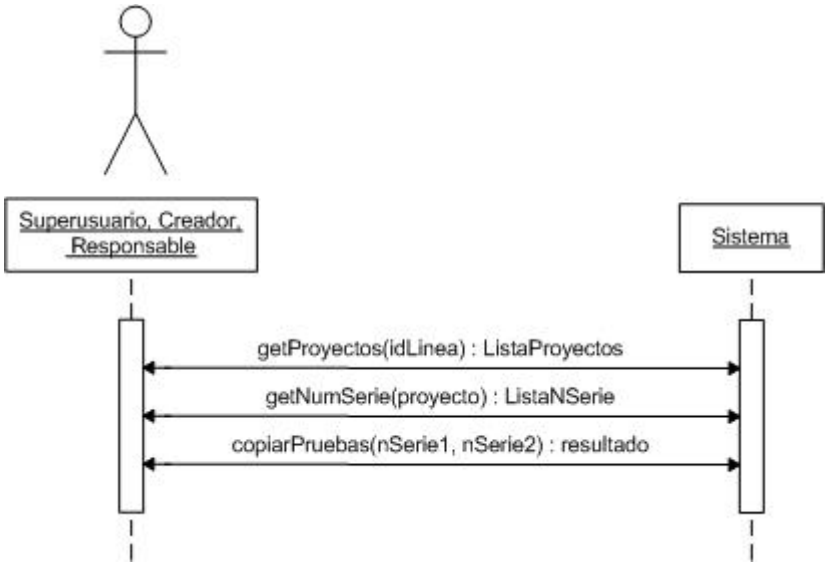


Imagen 5.5. Diagrama de secuencia de “Copiar pruebas”

5.5.1. Contrato de getProyectos

Name: getProyectos(idLinea) : ListaProyectos.

Responsabilities: Obtiene la lista de proyectos de la línea del usuario que están dados de alta en el gestor.

Preconditions: El parámetro idLinea es un identificador de línea válido.

Postconditions:

Salida: Devuelve una lista de proyectos relativos a la línea del usuario que están dados de alta en el gestor.

5.5.2. Contrato de getNumSerie

Name: getNumSerie(proyecto) : ListaNSerie.

Responsabilities: Obtiene la lista de los números de serie de los generadores que pertenecen al proyecto seleccionado, salvo el número de serie que ya está seleccionado (no se pueden copiar las pruebas de una máquina en esa misma máquina, no tiene sentido).

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve una lista de números de serie en los cuales se pueden copiar las pruebas.

5.5.3. Contrato de copiarPruebas

Name: copiarPruebas(nSerie1, nSerie2) : resultado.

Responsabilities: Copia la lista de pruebas asignadas del generador con nSerie1, en el generador con nSerie2.

Preconditions: Los parámetros nSerie1 y nSerie2 son números de serie válidos.

Postconditions: Obtiene la lista de pruebas asignadas de nSerie1, crea una nueva revisión de la lista de pruebas para nSerie2, y le asigna todas las pruebas de listaPruebas.

Salida: Devuelve un mensaje con el resultado de la operación.

5.6. Intercambiar pruebas

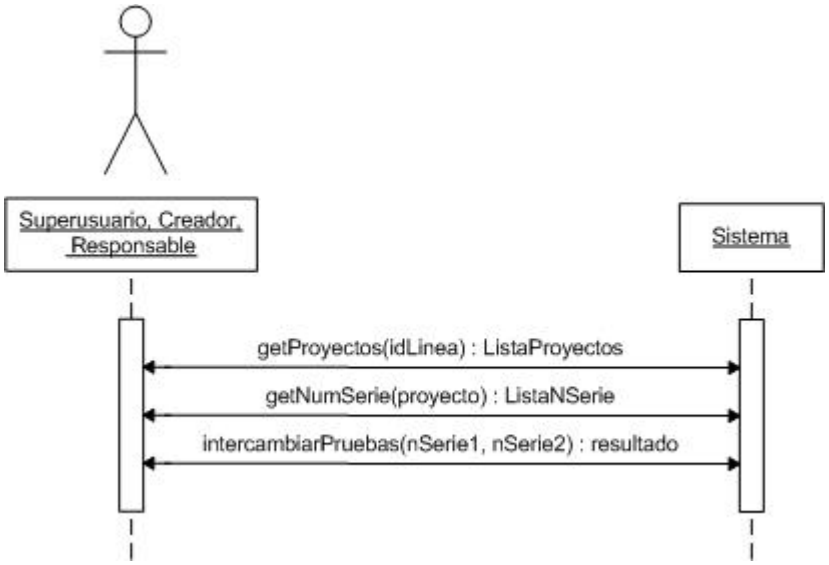


Imagen 5.6. Diagrama de secuencia de “Intercambiar pruebas”

5.6.1. Contrato de getProyectos

Name: obtProyectos(idLinea) : ListaProyectos.

Responsabilities: Obtiene la lista de proyectos de la línea del usuario que están dados de alta en el gestor.

Preconditions: El parámetro idLinea es un identificador de línea válido.

Postconditions:

Salida: Devuelve una lista de proyectos relativos a la línea del usuario que están dados de alta en el gestor.

5.6.2. Contrato de getNumSerie

Name: getNumSerie(proyecto) : ListaNSerie.

Responsabilities: Obtiene la lista de los números de serie de los generadores que pertenecen al proyecto seleccionado, salvo el número de serie que ya está seleccionado (no se pueden permutar las pruebas de una máquina en esa misma máquina, no tiene sentido).

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve una lista de números de serie con los cuales se pueden permutar las pruebas.

5.6.3. Contrato de intercambiarPruebas

Name: intercambiarPruebas(nSerie1, nSerie2) : resultado.

Responsabilities: Intercambia la lista de pruebas asignadas del generador con nSerie1 y el generador con nSerie2.

Preconditions: Los parámetros nSerie1 y nSerie2 son números de serie válidos. La entrada de datos de las pruebas del generador todavía no ha comenzado.

Postconditions: Obtiene la última lista de pruebas asignadas de nSerie1, obtiene la última lista de pruebas asignadas a nSerie2, crea una nueva revisión de la lista de pruebas para nSerie2, y le asigna todas las pruebas de la última revisión de pruebas de nSerie1. Finalmente, crea una nueva revisión de la lista de pruebas para nSerie1, y le asigna todas las pruebas de la revisión anteriormente obtenida para nSerie2 (hay que tener en cuenta que la última revisión de nSerie2 en este momento será la copia de las pruebas de nSerie1, por lo que realmente queremos copiar las pruebas de la penúltima revisión de la lista de pruebas)

Salida: Devuelve un mensaje con el resultado de la operación.

5.7. Definir datos de cabecera

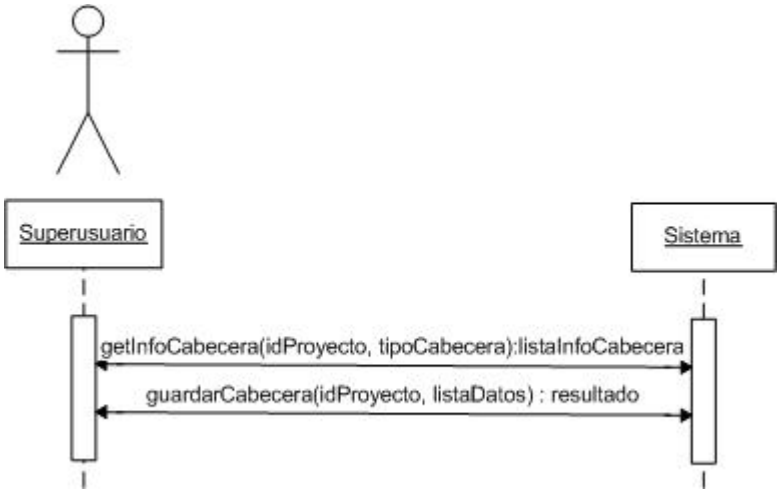


Imagen 5.7. Diagrama de secuencia de “Definir datos de cabecera”

5.7.1. Contrato de getInfoCabecera

Name: getInfoCabecera(idProyecto, tipoCabecera) : listaInfoCabecera

Responsabilities: Obtiene las variables relacionadas con un tipo de cabecera dado para un proyecto concreto. En el caso de que estas variables tengan medidas asociadas, también obtendrá esa información.

Preconditions: Los parámetros idProyecto y tipoCabecera son identificadores válidos.

Postconditions:

Salida: Devuelve una lista con toda la información relacionada con un tipo de cabecera de un proyecto.

5.7.2. Contrato de guardarCabecera

Name: guardarCabecera(idProyecto, listaDatos) : resultado

Responsabilities: Guarda una lista de datos relacionados con un tipo de cabecera.

Preconditions: Los datos encapsulados en el array pasado como parámetro ‘listaDatos’ son válidos.

Postconditions: Crea una nueva revisión de la lista de cabeceras para el proyecto dado, y le asigna los datos contenidos en el array ‘listaDatos’.

Salida: Muestra el resultado de la operación.

5.8. Entrada de datos

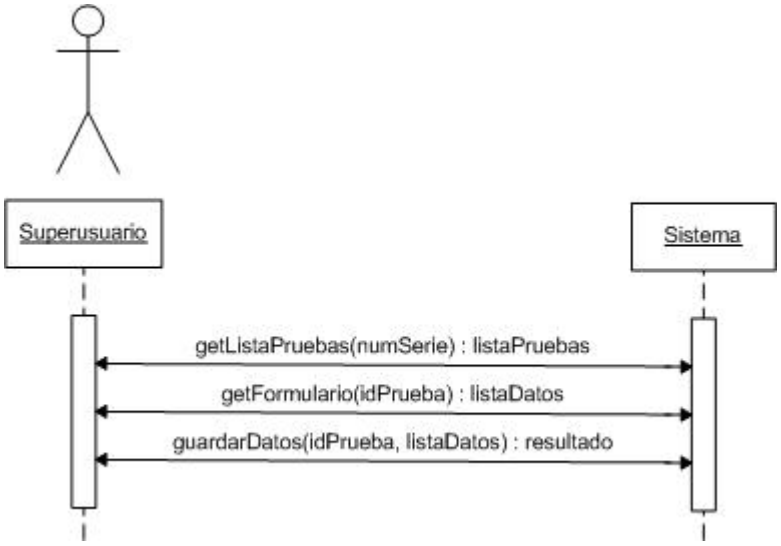


Imagen 5.8. Diagrama de secuencia de “Entrada de datos”

5.8.1. Contrato de getListaPruebas

Name: getListaPruebas(numSerie) : listaPruebas

Responsabilities: Obtiene la lista de pruebas asignadas que tiene ese número de serie.

Preconditions: El parámetro ‘numSerie’ es un número de serie válido.

Postconditions:

Salida: Devuelve una lista de pruebas que tiene asignada la máquina con el número de serie numSerie.

5.8.2. Contrato de getFormulario

Name: getFormulario(idPrueba) : listaDatos

Responsabilities: Obtiene las variables relacionadas con un tipo de prueba dado. En el caso de que estas variables tengan medidas asociadas, también obtendrá esa información.

Preconditions: El parámetro ‘idPrueba’ es un identificador de prueba válido.

Postconditions:

Salida: Devuelve una lista con toda la información (variables y medidas, en el caso de que las haya) relacionada con una prueba.

5.8.3. Contrato de guardarDatos

Name: guardarDatos(idPrueba, listaDatos) : resultado

Responsabilities: Guarda una lista de medidas relacionadas con una prueba dada.

Preconditions: Los datos encapsulados en el array pasado como parámetro 'listaDatos' son válidos.

Postconditions: Crea una nueva revisión de la lista de medidas para la prueba dada, y le asigna los datos contenidos en la lista 'listaDatos'.

Salida: Devuelve un mensaje indicando cuál ha sido el resultado de la operación.

5.9. Modificar responsables e implicados de proyecto

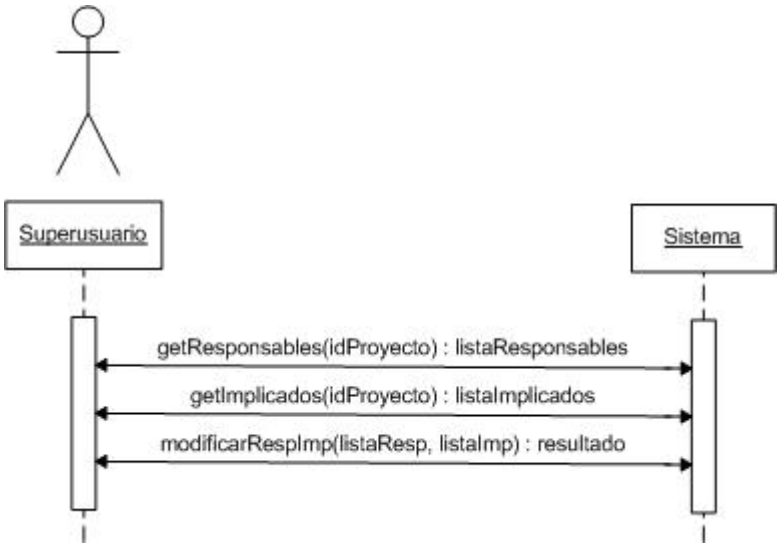


Imagen 5.9. Diagrama de secuencia de "Modificar responsables e implicados"

5.9.1. Contrato de getResponsables

Name: getResponsables(proyecto) : listaResponsables.

Responsabilities: Obtiene la lista de empleados que pueden ser responsables de un proyecto.

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve una lista de responsables.

5.9.2. Contrato de getImplicados

Name: getImplicados(proyecto) : listaImplicados.

Responsabilities: Obtiene la lista de empleados que pueden estar implicados en un proyecto.

Preconditions: El parámetro proyecto es un proyecto válido.

Postconditions:

Salida: Devuelve una lista de implicados.

5.9.3. Contrato de modificarRespImp

Name: guardarDatos(idPrueba, listaDatos) : resultado

Responsabilities: Guarda una lista de medidas relacionadas con una prueba dada.

Preconditions: Los datos encapsulados en el array pasado como parámetro 'listaDatos' son válidos.

Postconditions: Crea una nueva revisión de la lista de medidas para la prueba dada, y le asigna los datos contenidos en la lista 'listaDatos'.

Salida: Devuelve un mensaje indicando cuál ha sido el resultado de la operación.

5.10. Cerrar/Reabrir proyecto

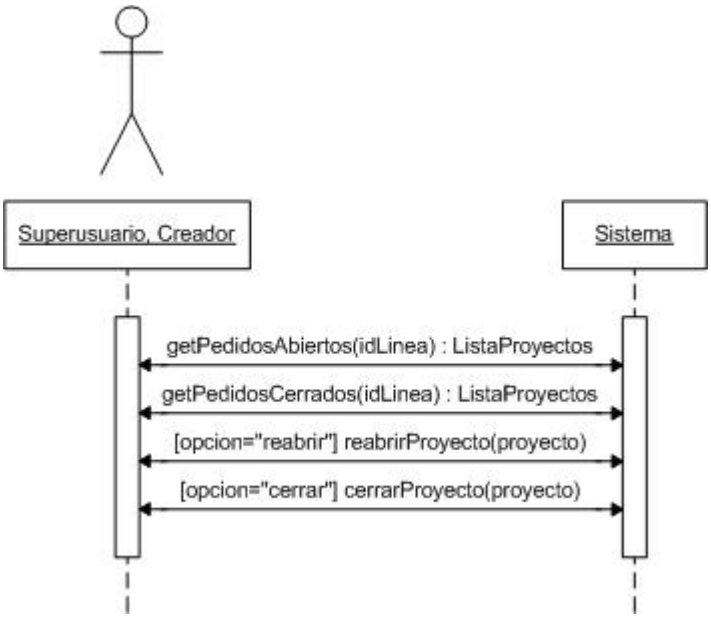


Imagen 5.10. Diagrama de secuencia de "Cerrar/Reabrir proyecto"

5.10.1. Contrato de getPedidosAbiertos

Name: getPedidosAbiertos(idLinea) : ListaProyectos.

Responsabilities: Obtiene la lista de los proyectos que están dados de alta en el **GBP**, y están abiertos, pertenecientes a la línea del usuario.

Preconditions: El parámetro idLinea es un identificador de línea válido.

Postconditions:

Salida: Devuelve una lista de proyectos abiertos en el **GBP** pertenecientes a la línea del usuario.

5.10.2. Contrato de getPedidosCerrados

Name: getPedidosCerrados(idLinea) : ListaProyectos.

Responsabilities: Obtiene la lista de los proyectos que están dados de alta en el **GBP**, y están cerrados, pertenecientes a la línea del usuario.

Preconditions: El parámetro idLinea es un identificador de línea válido.

Postconditions:

Salida: Devuelve una lista de proyectos cerrados en el **GBP** pertenecientes a la línea del usuario.

5.10.3. Contrato de reabrirProyecto

Name: reabrirProyecto(proyecto) : resultado.

Responsabilities: Dado un proyecto cerrado en el **GBP** para la línea del usuario, lo reabre.

Preconditions: El parámetro proyecto corresponde a un proyecto cerrado, y opción = “reabrir”.

Postconditions: Cambia el estado del proyecto seleccionado, y lo establece como ‘abierto’.

Salida: Devuelve un mensaje con el resultado de la operación.

5.10.4. Contrato de cerrarProyecto

Name: cerrarProyecto(proyecto) : resultado.

Responsabilities: Dado un proyecto abierto en el **GBP** para la línea del usuario, lo cierra.

Preconditions: El parámetro proyecto corresponde a un proyecto abierto, y opción = “cerrar”.

Postconditions: Cambia el estado del proyecto seleccionado, y lo establece como “cerrado”.

Salida: Devuelve un mensaje con el resultado de la operación.

5.11. Configuración

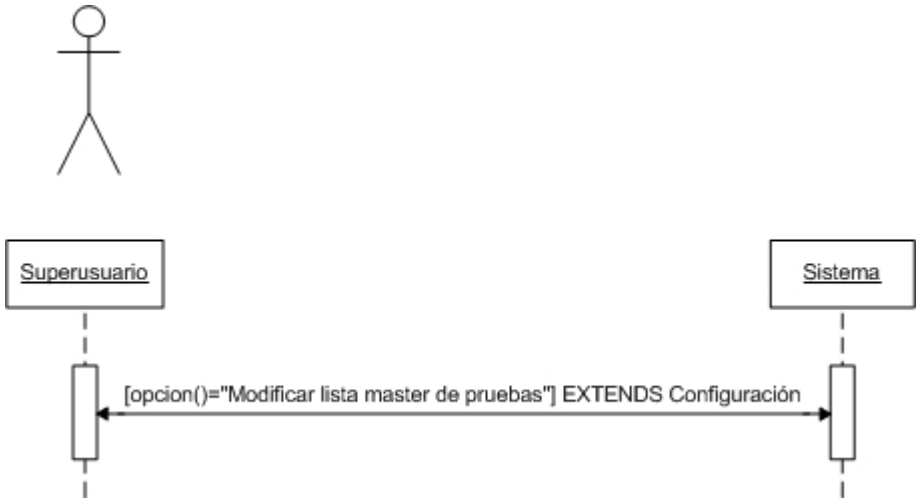


Imagen 5.11. Diagrama de secuencia de “Configuración”

5.12. Modificar lista master de pruebas

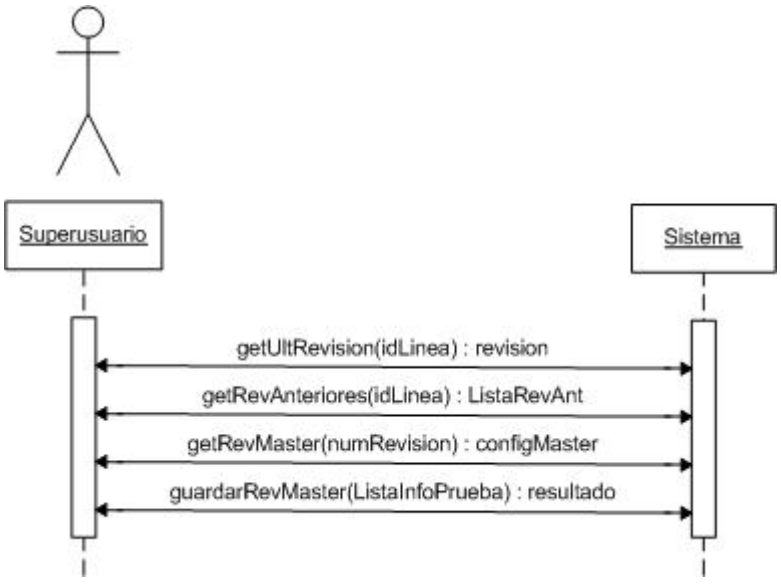


Imagen 5.12. Diagrama de secuencia de “Modificar lista master de pruebas”

5.12.1. Contrato de getUltRevision

Name: getUltRevision(idLinea) : revision.

Responsabilities: Obtiene última revisión de la master de pruebas de la línea del usuario.

Preconditions: El parámetro idLinea corresponde a un identificador de línea válido.

Postconditions:

Salida: Devuelve un número entero que es la última revisión del master de pruebas de la línea del usuario.

5.12.2. Contrato de getRevAnteriores

Name: getRevMaster(numRevision) : configMaster

Responsabilities: Obtiene las anteriores revisiones del master de pruebas de la línea del usuario.

Preconditions: El parámetro idLinea corresponde a un identificador de línea válido.

Postconditions:

Salida: Devuelve una lista de enteros que corresponden a las anteriores revisiones del master de pruebas de la línea del usuario.

5.12.3. Contrato de getRevMaster

Name: getRevMaster(numRevision) : configMaster.

Responsabilities: Obtiene la configuración de la tabla master de pruebas de la línea del usuario para la revisión pasada como parámetro. La configuración se compone una serie de pruebas, junto con el tipo que tienen en esa revisión.

Preconditions: El parámetro numRevision es un número de revisión válido.

Postconditions:

Salida: Devuelve la configuración de la tabla master de pruebas de la línea del usuario para la revisión pasada como parámetro.

5.12.4. Contrato de guardarRevMaster

Name: guardarRevMaster(listaInfoPrueba) : resultado.

Responsabilities: Crea una nueva revisión del master de pruebas de la línea del usuario, y guarda la configuración seleccionada por éste.

Preconditions: El parámetro listaInfoPrueba lo componen una lista de pruebas válidas con tipos asignados válidos.

Postconditions: Crea una nueva revisión del master de pruebas para la línea del usuario, y guarda todas las pruebas que componen esta revisión, junto con los tipos que tienen.

Salida: Devuelve un mensaje comunicando el resultado de la operación al usuario.

6. DISEÑO

A continuación se detallará el diseño de cada uno de los contratos de los casos de uso presentados en la sección anterior. Para ello, se mostrará el algoritmo en pseudocódigo que tendrá cada uno de los métodos especificados en el análisis del proyecto.

6.1. Identificar Usuario

6.1.1. Diseño de getNombreUsuario

Como información preliminar, hay que decir que los usuarios acceden al Gestor de Datos de Banco de Pruebas desde otra máquina. Por lo tanto, para conseguir el nombre de usuario lo que se hace es obtener el nombre en la máquina desde la que se va a acceder a la aplicación mediante la instrucción de asp Request.ServerVariables('LOGON_USER'), escribir el valor en un campo hidden y enviarla por POST al menú principal de la aplicación. El algoritmo tendría el siguiente aspecto:

Descripción: Identifica un usuario.

Algoritmo en pseudocódigo:

```
getNombreUsuario(){
      usuario = Request.ServerVariables('LOGON_USER');
      Introducir el nombre obtenido en un campo
      input de tipo hidden.
      Pasar mediante POST el valor a la página
      de inicio de la aplicación GBP.
      Obtener el valor de la variable $_POST mediante PHP.
}
```

6.1.2. Diseño de getInfoUsuario

Descripción: Método que asigna un rol (status) al usuario, que determinará los casos de uso a los que tiene acceso. En el caso de no tener permisos para utilizar la aplicación, se mostrará un mensaje notificándoselo y se le redireccionará al menú de la intranet de la empresa correspondiente a la línea de negocio del usuario (punto desde el cual se accede al **GBP**).

Algoritmo en pseudocódigo:

```
getInfoUsuario(usuario){
      infoUsuario = new InfoUsuario();
      base_datos = new BDMySQL();
      resultado = ejecutar->base_datos(consulta)
      si resultado no es vacío entonces
            infoUsuario.Usuario = resultado.Usuario;
            infoUsuario.Linea = resultado.Linea;
            infoUsuario.Status = resultado.Status;
            infoUsuario.idLinea = resultado.idLinea;
      fin si;
      devolver infoUsuario;
}
```

6.2. Crear proyecto

6.2.1. Diseño de getProyectos

Descripción: Muestra los pedidos de la línea del usuario (Hydro o CIM) que no estén cerrados ni tampoco estén dados de alta en la base de datos del **GBP**. En el caso de 'Hydro', simplemente obtiene los proyectos que no están cerrados en las Hojas de Diseño (HD) de la unidad de negocio y los introduce en listaProyectosAbiertos.

En el caso de 'CIM', obtiene los proyectos correspondientes a esta unidad de negocio que sean de generadores síncronos y los introduce en listaProyectosAbiertos.

En definitiva, obtiene los proyectos de las unidades de negocio que podemos dar de alta en el **GBP**.

Algoritmo en pseudocódigo:

```
getProyectos(idLinea){
      Si idLinea = 'Hydro' entonces
            listaProyectosPosibles=logicaHydro.getProyectosPosibles();
      si no
            listaProyectosPosibles = logicaCim.getProyectosPosibles();
      fin si.
      devolver listaProyectosPosibles;
}
```

Lógica de negocio Hydro:

Obtiene los proyectos que no están cerrados en las HD de la unidad de negocio y tampoco están dados de alta en el **GBP** y los introduce en listaProyectosPosibles.

```
getProyectosPosibles(){
      bd = new MySQL();
      pedidos_abiertos_hydro = bd->ejecuta(consultaHydro);
      pedidos _gbp = bd->ejecuta(consultaGBP);
      mientras indice<pedidos_abiertos_hydro hacer
            mientras indice2<pedidos_gbp y no encontrado hacer
                  si          pedidos_abiertos_hydro[indice]          =
                  pedidos_gbp[indice]  entonces
                        encontrado = true;
                  fin si;
            fin mientras;
            si no encontrado entonces
                  listaProyectosPosibles                              =
                  pedidos_abiertos_hydro[indice];
            fin si;
            encontrado = false;
      fin mientras;
      devolver listaProyectosPosibles;
}
```

Lógica de negocio CIM:

Análoga a la lógica de ‘Hydro’, salvo que las consultas se realizan a una base de datos SQL Server y dicha consulta tendrá en cuenta solamente los pedidos de generadores síncronos de ‘CIM’, ya que a diferencia de ‘Hydro’, en esta unidad de negocio hay proyectos que no son de ese tipo (motores, cogeneradores, generadores asíncronos…).

```
getProyectosPosibles(){
      bd = new SQLServer();
      pedidos_cim = bd->ejecuta(consultaCim);
      pedidos _gbp = bd->ejecuta(consultaGBP);
      mientras indice<pedidos_cim hacer
            mientras indice2<pedidos_gbp y no encontrado hacer
                  si pedidos_cim[indice]=pedidos_gbp[indice]  entonces
                        encontrado = true;
                  fin si;
```

```
            fin mientras;
            si no encontrado entonces
                  listaProyectosPosibles = pedidos_cim[indice];
            fin si;
            encontrado = false;
      fin mientras;
      devolver listaProyectosPosibles;
}
```

6.2.2. Diseño de getRevision

Descripción: Obtiene revisión del proyecto pasado como parámetro, donde revisión se corresponde con la última revisión existente para dicho proyecto. Esta operación sólo se realiza para la UDN[10] 'Hydro', ya que en 'Cim' no se lleva el control de las revisiones de los proyectos, se machacan los datos de una versión a otra.

Algoritmo en pseudocódigo:

```
getRevision(numProyecto){
      bd = new MySQL();
      ultima_rev = bd->ejecuta(consulta);
      devolver ultima_rev;
}
```

6.2.3. Diseño de getResumen

Descripción: Obtiene el tipo de máquina, nombre de la central, número de unidades y configuración de la máquina y guarda los datos en resumen. Elimina los caracteres especiales del campo 'NomCentral' (nombre de la central), debido a que esta función se ejecuta mediante AJAX y en ocasiones el nombre de la central contiene caracteres reservados de XML, y hay que tratarlos previamente para que funcione correctamente.

[10] Unidad De Negocio

Algoritmo en pseudocódigo:

```
getResumen(numProyecto){
      bd = new MySQL();
      resumen = bd->ejecuta(consulta);
      si resumen no es null entonces
            eliminar_caracteres_especiales(resumen.NomCentral);
      fin si;
      devolver resumen;
}
```

6.2.4. Diseño de getResponsables

Descripción: Obtiene los usuarios que pueden ser responsables de un proyecto de la UDN del usuario. Como en la base de datos de Cim no se especifican los empleados, se ha tenido que introducir en la base de datos MySQL del **GBP** de forma manual una tabla en la que se especifiquen los responsables e implicados de esta unidad de negocio. A esto se debe que hagamos una consulta a una base de datos MySQL y no SQL Server.

Algoritmo en pseudocódigo:

```
getResponsables(){
      Si idLinea = 'Hydro' entonces
            consulta = consultaHydro;
      si no
            consulta = consultaCim;
      fin si;
      bd = new MySQL();
      listaResponsables = bd->ejecuta(consulta);
      devolver listaResponsables;
}
```

6.2.5. Diseño de getImplicados

Descripción: Obtiene los usuarios que pueden estar implicados en un proyecto de la UDN del usuario. Análogo al caso de getResponsables.

Algoritmo en pseudocódigo:

```
getImplicados(){
      Si idLinea = ‘Hydro’ entonces
            consulta = ‘obtener implicados hydro’;
      si no
            consulta = ‘obtener implicados cim’;
      fin si;
      bd = new MySQL();
      listaImplicados = bd->ejecuta(consulta);
      devolver listaImplicados;
}
```

6.2.6. Diseño de altaProyecto

Descripción: Da de alta un proyecto en el Gestor de Datos de Banco de Pruebas.

Algoritmo en pseudocódigo:

```
altaProyecto(numProyecto, listaResponsables, listaImplicados){
      bd = new MySQL();
      bd->iniciarTransaccion();
      consulta = ‘insertar proyecto en tabla proyecto’
      bd->ejecuta(consulta);
      insertarMaquinas(numProyecto, posicion);
      insertarResponsables(idProyecto, listaResponsables);
      insertarImplicados(idProyecto, listaImplicados);
      bd->cerrarTransaccion();.
}
```

Algoritmo insertar máquinas:

```
insertarMaquinas(numProyecto, posicion){
      idProyecto = getIdProyecto(numProyecto, posicion);
      numeros_serie = getNumSerie(numProyecto+posicion);
      para cada numero en numeros_serie hacer
            insertarEnBD(numero);
      Fin para.
}
```

Algoritmo insertar responsables:

```
insertarResponsables(idProyecto, listaResponsables){
      consulta = ‘insertar en tabla’;
      para cada responsable en listaResponsables hacer
            consulta = consulta+responsable;
      Fin para.
      insertarEnBD(consulta);
}
```

Algoritmo insertar implicados:

```
insertarImplicados(idProyecto, listaImplicados){
      consulta = ‘insertar en tabla’;
      para cada implicado en listaImplicados hacer
            consulta = consulta+implicado;
      fin para.
      insertarEnBD(consulta);
}
```

6.3. Abrir proyecto

6.3.1. Diseño de getProyectos

Descripción: Obtiene los proyectos asociados a la idLinea pasada como parámetro y los almacena en listaProyectos.

Algoritmo en pseudocódigo:

```
getProyectos(idLinea){
      base_datos = new MySQL();
      listaProyectos = ejecuta->base_datos(consulta);
      devolver listaProyectos;
}
```

6.3.2. Diseño de getNumSerie

Descripción: Obtiene los números de serie asociados al proyecto dado y los almacena en la variable listaNSerie. Para obtener los números de serie asociados a un proyecto, la consulta habrá que hacerla en una base de datos Oracle. A nivel de diseño esto no es relevante, pero conviene puntualizarlo. Los diferentes tipos de BD se

tratan de forma más extendida en las secciones 'Elección tecnológica' e 'Implementación'.

Algoritmo en pseudocódigo:

```
getNumSerie(numProyecto){
      bd = conectar();
      // Elimina los guiones de numProyecto
      numProyecto = str_replace("-", "", numProyecto);
      listaNSerie = ejecuta->bd(consulta);
      devolver listaNSerie;
}
```

6.4. Definir pruebas

6.4.1. Diseño de getPruebasAsig

Descripción: Obtiene las pruebas asignadas en la última revisión de la lista de pruebas del número de serie pasado como parámetro y lo almacena en ListaPruebas.

Algoritmo en pseudocódigo:

```
getPruebasAsig(numSerie){
      bd = new MySQL();
      consulta = 'obtener la última revisión de pruebas de
                  numSerie'
      ultima_revision = bd->ejecuta(consulta);
      consulta = obtener las pruebas de ultima_revision de
                  numSerie
      pruebas_asig = bd->ejecuta(consulta);
      devolver pruebas_asig;
}
```

6.4.2. Diseño de getPruebasPosib

Descripción: Obtiene todas las pruebas del master de pruebas cuyo estado sea 'activa' y que no estén asignadas al número de serie pasado como parámetro. Almacena el resultado en listaPruebas.

Algoritmo en pseudocódigo:

```
getPruebasPosib(numSerie){
      bd = new MySQL();
      consulta = total_pruebas-pruebas_asig;
      pruebas_posib = bd->ejecuta(consulta);
      devolver pruebas_posib;
}
```

6.4.3. Diseño de getUltRevision

Descripción: Obtiene el número que indica la última revisión de la lista de pruebas de la máquina.

Algoritmo en pseudocódigo:

```
getUltRevision(numSerie){
      bd = new MySQL();
      consulta = MAX(revisión de numSerie)
      ultima_revision = bd->ejecuta(consulta);
      devolver ultima_revision;
}
```

6.4.4. Diseño de guardarPruebas

Descripción: Guarda las pruebas almacenadas en el parámetro listaPruebas, en un número de serie concreto de un número de proyecto dado.

Algoritmo en pseudocódigo:

```
guardarPruebas(nProyecto, numSerie, listaPruebas){
      bd = new MySQL();
      bd->iniciarTransaccion();
      ultima_revision = getUltRevision(numSerie);
      nueva_revision = ultima_revision+1;
      consulta = insertar nueva_revision;
      bd->ejecuta(consulta);
      para cada prueba en listaPruebas hacer
            consulta = insertar prueba;
            bd->ejecuta(consulta);
      fin para;
```

```
      bd->cerrarTransaccion();
}
```

6.5. Copiar pruebas

6.5.1.Diseño de getProyectos

Descripción: Obtiene los proyectos asociados a la idLinea pasada como parámetro y los almacena en listaProyectos.

Algoritmo en pseudocódigo:

```
getProyectos(idLinea){
      base_datos = new MySQL();
      listaProyectos = ejecuta->base_datos(consulta);
      devolver listaProyectos;
}
```

6.5.2. Diseño de getNumSerie

Descripción: Obtiene los números de serie asociados al proyecto dado y los almacena en la variable listaNSerie, excepto el número de serie del generador actual (no se permite la copia de pruebas de un generador a él mismo).

Algoritmo en pseudocódigo:

```
getNumSerie(numProyecto){
      bd = conectar();
      // Elimina los guiones de numProyecto
      numProyecto = str_replace("-", "", numProyecto);
      consulta = 'números de serie de un proyecto dado, excepto
                  el número de serie del generador actual'
      listaNSerie = bd->ejecuta(consulta);
      devolver listaNSerie;
}
```

6.5.3. Diseño de copiarPruebas

Descripción: Copia las pruebas de un generador a otro generador distinto (perteneciente al mismo proyecto o a otro diferente).

Algoritmo en pseudocódigo:

```
copiarPruebas(numSerieFuente, numSerieDestino){
      bd = new MySQL();
      bd->iniciarTransaccion();
      listaPruebasCopiar = getPruebasAsig(numSerieFuente);
      ultima_revision = getUltRevision(numSerieDestino);
      nueva_revision = ultima_revision+1;
      consulta = insertar nueva_revision;
      bd->ejecuta(consulta);
      para cada prueba en listaPruebasCopiar hacer
            consulta = ‘insertar prueba’;
            bd->ejecuta(consulta);
      fin para;
      bd->cerrarTransaccion();
}
```

6.6. Intercambiar pruebas

6.6.1. Diseño de getProyectos

Descripción: Obtiene los proyectos asociados a la idLinea pasada como parámetro y los almacena en listaProyectos.

Algoritmo en pseudocódigo:

```
getProyectos(idLinea){
      base_datos = new MySQL();
      consulta = ‘obtener proyectos de idLinea’;
      listaProyectos = base_datos->ejecuta(consulta);
      devolver listaProyectos;
}
```

6.6.2. Diseño de getNumSerie

Descripción: Obtiene los números de serie asociados al proyecto dado y los almacena en la variable listaNSerie, excepto el número de serie del generador actual (no está permitido el intercambio de pruebas de un generador con él mismo).

Algoritmo en pseudocódigo:

```
getNumSerie(numProyecto){
      bd = conectar();
      // Elimina los guiones de numProyecto
      numProyecto = str_replace(“-“, “”, numProyecto);
      consulta = ‘números de serie de un proyecto dado, excepto
                  el número de serie del generador actual’
      listaNSerie = bd->ejecuta(consulta);
      devolver listaNSerie;
}
```

6.6.3. Diseño de intercambiarPruebas

Descripción: Intercambia las pruebas de un generador con otro generador distinto. Es decir, si un generador llamado ‘A’ tiene las pruebas 1, 2 y 3, y otro generador llamado ‘B’ tiene las pruebas 2, 3, y 4, y las intercambiamos, el generador A pasará a tener una nueva revisión de pruebas formada por las 2, 3 y 4, y el generador B pasará a tener una nueva revisión de pruebas formada por las pruebas 1, 2 y 3. No se permite la copia de pruebas de un generador con él mismo.

Algoritmo en pseudocódigo:

```
intercambiarPruebas(numSerie1, numSerie2){
      bd = new MySQL();
      bd->iniciarTransaccion();
      ultima_revision_1 = getUltRevision(numSerie1);
      ultima_revision_2 = getUltRevision(numSerie2);
      consulta = obtener la lista de pruebas correspondiente
                  a la última revisión de numSerie1
      lista_1 = bd->ejecuta(consulta);
      consulta = obtener la lista de pruebas correspondiente
                  a la última revisión de numSerie2
      lista_2 = bd->ejecuta(consulta);
      nueva_revision_2 = ultima_revision_2+1;
      consulta = insertar nueva_revision;
      bd->ejecuta(consulta);
```

```
      para cada prueba en lista_1 hacer
            consulta = insertar prueba en numSerie2;
            bd->ejecuta(consulta);
      fin para;

      nueva_revision_1 = ultima_revision_1+1;
      consulta = insertar nueva_revision;
      bd->ejecuta(consulta);
      para cada prueba en lista_2 hacer
            consulta = insertar prueba en numSerie1;
            bd->ejecuta(consulta);
      fin para;
      resultado = bd->cerrarTransaccion();
      devolver resultado;
}
```

6.7. Definición de datos de cabecera

6.7.1. Diseño de getInfoCabecera

Descripción: Obtiene las variables relacionadas con un tipo de cabecera dado para un proyecto concreto. En el caso de que estas variables tengan medidas asociadas, también obtendrá esa información.

Algoritmo en pseudocódigo:

```
getInfoCabecera(idProyecto, tipoCabecera){
      bd = new MySQL();
      consulta = 'obtener variables de prueba nombrePrueba';
      listaDatos = bd->ejecuta(consulta);
      listaInfoCabecera = completarDatos(listaDatos);
      devolver listaDatos;
}
```

6.7.2. Diseño de guardarCabecera

Descripción: Guarda los datos de cabeceras introducidos en el formulario correspondiente.

Algoritmo en pseudocódigo:

```
guardarCabecera(nProyecto, listaDatos){
      bd = new MySQL();
      bd->iniciarTransaccion();
      consulta = 'obtener id de proyecto nProyecto';
      idProyecto = bd->ejecuta(consulta);
      consulta = 'insertar nueva revisión de proyecto idProyecto';
      bd->ejecuta(consulta);
      consulta = 'insertar ';
      mientras i<listaDatos.length hacer
            consulta = consulta+listaDatos[i];
      fin mientras;
      resultado = bd->ejecuta(consulta);
      devolver resultado;
}
```

6.8. Entrada de datos

6.8.1. Diseño de getListaPruebas

Descripción: Obtiene la lista de pruebas asignadas a un número de serie dado.

Algoritmo en pseudocódigo:

```
getListaPruebas(nSerie){
      bd = new MySQL();
      consulta = 'obtener pruebas asignadas
                  al número de serie nSerie';
      listaPruebas = bd->ejecuta(consulta);
      devolver listaPruebas;
}
```

6.8.2. Diseño de getFormulario

Descripción: Obtiene el formulario correspondiente a la entrada de datos de la prueba dada.

Algoritmo en pseudocódigo:

```
getFormulario(nombrePrueba){
      bd = new MySQL();
      consulta = ‘obtener variables de prueba nombrePrueba’;
      listaDatos = bd->ejecuta(consulta);
      listaDatos = completarDatos(listaDatos);
      devolver listaDatos;
}
```

6.8.3. Diseño de guardarDatos

Descripción: Guarda los datos introducidos en los formularios de entrada de datos para una prueba de un proyecto.

Algoritmo en pseudocódigo:

```
guardarDatos(idPrueba, listaDatos){
      bd = new MySQL();
      bd->iniciarTransaccion();
      consulta = ‘insertar ’;
      mientras i<listaDatos.length hacer
            consulta = consulta+listaDatos[i];
      fin mientras;
      resultado = bd->ejecuta(consulta);
      devolver resultado;
}
```

6.9. Modificar responsables e implicados

6.9.1. Diseño de getResponsables

Descripción: Obtiene los responsables asociados a un proyecto.

Algoritmo en pseudocódigo:

```
getResponsables(idProyecto){
      bd = new MySQL();
      consulta = ‘obtener responsables del proyecto idProyecto’
      listaResponsables = bd->ejecuta(consulta);
      devolver listaResponsables;
}
```

6.9.2. Diseño de getImplicados

Descripción: Obtiene los implicados asociados a un proyecto.

Algoritmo en pseudocódigo:

```
getImplicados(idProyecto){
      bd = new MySQL();
      consulta = ‘obtener implicados del proyecto idProyecto’;
      listaImplicados = bd->ejecuta(consulta);
      devolver listaImplicados;
}
```

6.9.3. Diseño de modificarRespImp

Descripción: Modifica los responsables e implicados de un proyecto.

Algoritmo en pseudocódigo:

```
modificarRespImp(nProyecto, responsables, implicados){
      bd = new MySQL();
      bd->iniciarTransaccion();
      consulta = ‘obtener id proyecto nProyecto’;
      idProyecto = bd->ejecuta(consulta);
      consulta = ‘borrar responsables e implicados del proyecto’;
      bd->ejecuta(consulta);
      insertarResponsables(idProyecto, responsables);
      insertarImplicados(idProyecto, implicados);
      resultado = bd->cerrarTransaccion();
      devolver resultado;
}
```

```
insertarResponsables(idProyecto, responsables){
      consulta = 'insertar';
      mientras (i<responsables.length) hacer
            consulta = 'consulta'+responsables[i];
      fin mientras;
      bd->ejecuta(consulta);
}

insertarImplicados(idProyecto, implicados){
      consulta = 'insertar';
      mientras (i<implicados.length) hacer
            consulta = 'consulta'+implicados[i];
      fin mientras;
      bd->ejecuta(consulta);
}
```

6.10. Cerrar/Reabrir

6.10.1. Diseño de getPedidosAbiertos

Descripción: Obtiene los proyectos del **GBP** asociados al idLinea pasado como parámetro, cuyo campo 'cerrado' sea igual a 0 (es decir, que el proyecto no esté cerrado).

Algoritmo en pseudocódigo:

```
getPedidosAbiertos(idLineaP, numProyecto){
      consulta = 'proyectos con cerrado='0' e idLinea=idLineaP'
      pedidosAbiertos = bd->ejecuta(consulta);
      devolver pedidosAbiertos;
}
```

6.10.2. Diseño de getPedidosCerrados

Descripción: Obtiene los proyectos del **GBP** asociados al idLinea pasado como parámetro, cuyo campo 'cerrado' sea igual a 1 (es decir, que el proyecto esté cerrado).

Algoritmo en pseudocódigo:

```
getPedidosCerrados(idLineaP, numProyecto){
      consulta = ‘obtener pedidos cerrados’
      pedidosCerrados = bd->ejecuta(consulta);
      devolver pedidosCerrados;
}
```

6.10.3. Diseño de reabrirProyecto

Descripción: Actualiza el campo ‘cerrado’ de la tabla ‘proyecto’ correspondiente al proyecto pasado como parámetro, cambiando el valor del campo de ‘1’ a ‘0’.

Algoritmo en pseudocódigo:

```
reabrirProyecto(idLinea, numProyecto, posicion){
      bd = new MySQL();
      bd->iniciarTransaccion();
      consulta='actualizar estado de proyecto
                  marcándolo como abierto';
      bd->ejecuta(consulta);
      bd->cerrarTransaccion();
}
```

6.10.4. Diseño de cerrarProyecto

Descripción: Actualiza el campo ‘cerrado’ de la tabla ‘proyecto’ correspondiente al proyecto pasado como parámetro, cambiando el valor del campo de ‘0’ a ‘1’.

Algoritmo en pseudocódigo:

```
cerrarProyecto(idLinea, numProyecto, posicion){
      bd = new MySQL();
      bd->iniciarTransaccion();
      consulta='actualizar estado de proyecto
                  marcándolo como cerrado';
      bd->ejecuta(consulta);
      bd->cerrarTransaccion();
}
```

6.11. Modificar lista master pruebas

6.11.1. Diseño de getUltRevision

Descripción: Obtiene el número de la última revisión de la master pruebas de la línea del usuario.

Algoritmo en pseudocódigo:

```
getUltRevision(idLinea){
      bd = new MySQL();
      consulta = "obtener última revision de master de pruebas"
      ultima_revision = bd->ejecuta(consulta);
      devolver ultima_revision;
}
```

6.11.2. Diseño de getRevAnteriores

Descripción: Devuelve los números de las anteriores revisiones de la master de pruebas para la línea del usuario.

Algoritmo en pseudocódigo:

```
getRevAnteriores(idLinea){
      bd = new MySQL();
      consulta = "obtener revisiones anteriores"
      anteriores = bd->ejecuta(consulta);
      devolver anteriores;
}
```

6.11.3. Diseño de getRevMaster

Descripción: Para una prueba, devuelve el tipo de la prueba que tiene dicha prueba en una revisión determinada del master de pruebas si numRevMP es distinto de null y si no, devuelve el tipo de prueba que tiene en la última revisión del master de pruebas para un usuario de una línea. En otras palabras, mediante esta función se carga el tipo que tenía una prueba en la revisión ‘numRevMP’, y si este parámetro es null, se carga el tipo que tiene en la última revisión del master de pruebas de la línea del usuario.

Algoritmo en pseudocódigo:

```
getRevMaster(idMP, numRevMP){
      bd = new MySQL();
      configMaster = new ConfigMaster();
      Si numRevMP es distinto de null entonces
            consulta = "obtener idAlcance e idEspecifiacion
                        de la prueba idMP en el número de revisión
                        de la master de pruebas numRevMP"
      Si no
            consulta = "obtener idAlcance e idEspecifiacion
                        de la prueba idMP en la última revisión
                        de la master de pruebas"
      Fin si;
      configMaster = bd->ejecuta(consulta);
      devolver configMaster;
}
```

6.11.4. Diseño de guardarRevMaster

Descripción: Guarda la lista de pruebas creando una nueva revisión de la master de pruebas para la línea del usuario.

Algoritmo en pseudocódigo:

```
guardarRevMaster(idLinea, lista_pruebas){
      bd = new MySQL();
      bd->iniciarTransaccion();
      consulta = "obtener última revision de master de pruebas";
      ultima_rev = bd->ejecuta(consulta);

      si ultima_rev no es null entonces
            ultima_rev = ultima_rev+1;
      si no
            ultima_rev = 1;
      fin si;
      mientras indice<lista_pruebas hacer
            consulta = "inserter prueba en la nueva revisión";
            resul = bd->ejecuta(consulta);
      fin mientras;
      bd->cerrarTransaccion();
      devolver resul;
}
```

7. IMPLEMENTACIÓN

Por política de privacidad de la empresa en la que he desarrollado el proyecto, no se ha podido adjuntar todo el código de la aplicación. Por ello explicaré a grandes rasgos cómo se ha realizado la implementación del proyecto. He adjuntado parte del código, con la finalidad de exponer de una forma más clara los principales aspectos de la implementación.

7.1. Organización del código

En esta sección se pretende explicar la forma en la que se encuentra organizado el código en el servidor, indicando dónde se encuentra cada parte del código y la función que cumple dentro de la aplicación. En otras palabras, lo que se pretende explicar es la forma en la que se organiza el código, de forma que si hay que cambiar algo en la aplicación que suponga modificar el código, se sepa cuáles son los ficheros implicados que hay que modificar. A la hora de tomar la decisión de cómo organizar el código, se ha intentado que dicha organización fuese lo más natural e intuitiva posible

7.1.1. Directorio raíz

Comenzaremos por el directorio raíz de la aplicación, el cual presenta el siguiente aspecto:

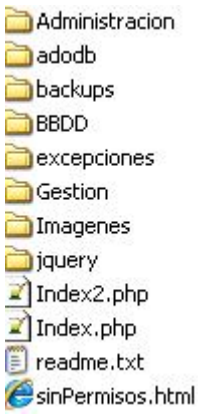


Imagen 7.1. Contenido del directorio raíz

En la carpeta ‘Administración’ se encuentran los casos de uso relativos a la administración de la aplicación. De momento en esta carpeta solamente se encuentra

el caso de uso ‘Modificar master pruebas’, aunque en el futuro lo más probable es que se añadan otros casos de uso como ‘Añadir prueba a master de pruebas’ o ‘Modificar prueba’.

En la carpeta ‘adodb’ se encuentra ADOdb, la cual es un conjunto de bibliotecas de bases de datos para PHP. En este proyecto se utiliza esta librería solamente a la hora de conectarse a la base de datos Oracle de SAP, mientras que para el resto de operaciones con bases de datos, tanto en MySQL como en SQL Server, se utilizarán sus API’s de PHP correspondientes.

La carpeta ‘backups’ simplemente es el lugar en el servidor de desarrollo donde se almacenan las copias de seguridad, no tiene mayor relevancia en la implementación.

En el directorio ‘BBDD’ es donde se encuentra la capa de datos de la aplicación, en este directorio podemos encontrar las clases utilizadas para conectar tanto con la base de datos Oracle de SAP, como con las bases de datos MySQL del **GBP** y del Gestor de Elementos Críticos (GEC), y con la base de datos SQL Server de Cim.

En el directorio ‘excepciones’ se encuentra un fichero php en el que se definen 2 tipos de excepciones propias, ExcepcionConectarBD y ExcepcionEjecutarSQL, las cuales se utilizan a la hora de capturar las posibles excepciones que puedan darse en la aplicación con una mayor eficacia. El fichero en sí es extremadamente simple, ya que contiene sólo el siguiente código:

```
<?php
      class ExcepcionConectarBD extends Exception { }
      class ExcepcionEjecutarSQL extends Exception { }
?>
```

Sin embargo, para cumplir su cometido no hace falta más. De esta forma, la captura de las excepciones que se den al conectar con la base de datos o al ejecutar alguna sentencia SQL se capturan de la siguiente manera:

```
try{
      // Código de la aplicación
}catch (ExcepcionConectarBD $e){
      // Código de tratamiento de excepción al conectar a BD
}catch (ExcepcionEjecutarSQL $e){
      // Código de tratamiento de excepción al ejecutar SQL
}catch (Exception $e){
      // Código de tratamiento de excepción de otro tipo
}
```

La carpeta ‘Gestion’ es la carpeta fundamental de este proyecto, ya que en ella se encuentra la mayor parte del código de casi todos los casos de uso que se han implementado. Por este motivo, se analizará el contenido de este directorio con mayor profundidad en la siguiente subsección.

El directorio ‘Imagenes’ contiene las imágenes que se muestran en la aplicación, del cual el ejemplo más destacable es el logo de INDAR S.L., el cual se muestra en la cabecera de todas las páginas de la aplicación.

Imagen 7.2. Logo de Indar

El directorio ‘jquery’ contiene los ficheros de jQuery que se utilizan en la aplicación, que son comunes a todo el proyecto. jQuery es una biblioteca de JavaScript extremadamente popular y de reconocida solvencia. Se hablará de ella con mayor profundidad en uno de los próximos apartados de esta sección.

El fichero ‘Index.php’ es simplemente el menú principal de la aplicación.

El fichero ‘readme.txt’ se utiliza solamente para comentar los últimos cambios realizados a la aplicación. Esto es útil a la hora de hacer las copias de seguridad, para saber exactamente qué está hecho y qué no en determinada copia de seguridad.

El fichero ‘sinPermisos.html’ es el fichero al cual te redirecciona la aplicación en el caso de no tener permisos para ver una página. Cuando la aplicación esté en producción, este fichero sobrará ya que la aplicación debe redireccionar al menú

principal de la unidad de negocio del usuario sin permisos. Sin embargo, a la hora de desarrollar, me resulta útil, ya que yo no formo parte de ninguna de estas unidades de negocio y por lo tanto no estoy autorizado para acceder a dichos menús.

7.1.2.Directorio ‘Gestion’

En este directorio nos encontramos los siguientes subdirectorios:

Imagen 7.3. Directorio ‘Gestion’

Comprobamos que son 3: ‘Crear’, ‘Cerrar’ y ‘Abrir’. Si observamos con atención en el apartado de casos de uso el diagrama correspondiente, comprobamos que son los 3 casos de uso ‘principales’. Por tanto, en la carpeta ‘Crear’ encontraremos el caso de uso ‘Crear proyecto’, en la carpeta ‘Cerrar’ encontraremos el caso de uso ‘Cerrar/Reabrir proyecto’ y en la carpeta ‘Abrir’ encontraremos el caso de uso ‘Abrir proyecto’, y además, todos los casos de uso que dependen de éste. Para verlo de forma clara, se adjunta la siguiente imagen que muestra el contenido del directorio ‘Abrir’:

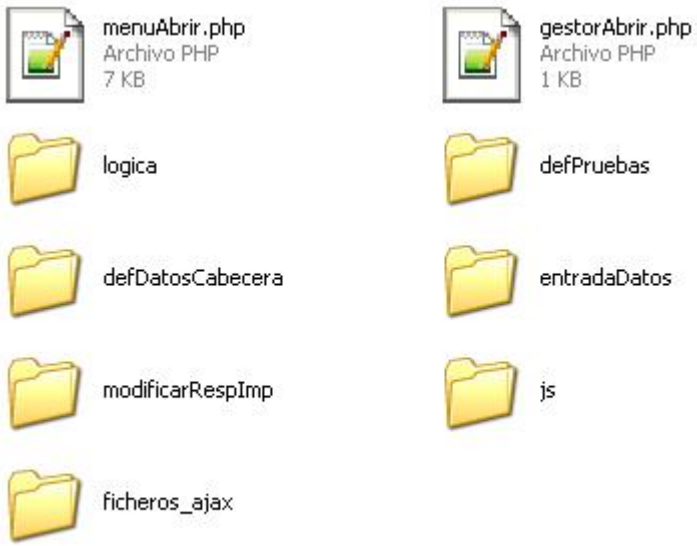


Imagen 7.4. Imagen de contenido de directorio ‘Abrir’

Ahora se procederá a explicar cada uno de los elementos que se pueden observar. En primer lugar, indicar que los ficheros ‘menuAbrir.php’ y ‘gestorAbrir.php’ se corresponden con la capa de presentación de la aplicación. En el fichero ‘menuAbrir.php’ se encuentra todo el HTML estático de la página, y mediante ‘gestorAbrir.php’ se crea todo el HTML dinámico que tenga la página, mediante PHP. El motivo de tener 2 ficheros distintos que se encarguen de la capa de presentación es porque la empresa deseaba que en el fichero que abre el usuario haya el menor código posible de PHP. De esta forma el único código PHP que contienen las páginas son las llamadas a las funciones pertinentes.

En la carpeta ‘js’ se encuentra el código JavaScript propio del caso de uso. Cada uno de los casos de uso dispondrá de un fichero como este.

En la carpeta ‘ficheros_ajax’ se encuentran los ficheros utilizados por las llamadas AJAX que se realicen en el caso de uso correspondiente. En caso de utilizar AJAX, los ficheros relacionados se encontrarán aquí.

En la carpeta ‘logica’ se encuentran los ficheros correspondientes a la lógica de negocio del caso de uso en cuestión. Así pues, los ficheros de la capa de presentación del caso de uso disponen de la lógica indicada en los ficheros de este directorio, y a su vez, los ficheros de lógica de negocio hacen uso de los que se encuentran en el subdirectorio ‘BBDD’ del directorio raíz para acceder a la base de datos.

El resto de carpetas, se corresponden con los casos de uso ‘Definir pruebas’, ‘Definir datos de cabecera’, ‘Entrada de datos’ y ‘Modificar responsables e implicados de proyecto’. La estructura de estos casos de uso es análoga a la analizada. Con una imagen que muestre la estructura de alguno de los casos de uso anteriores, como por ejemplo, ‘Definir pruebas’, se espera que quede clara la analogía entre ambos, y se comprenda con claridad la estructura del código del proyecto.

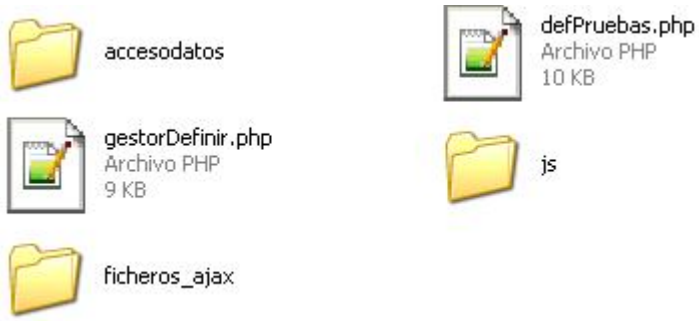


Imagen 7.5 Imagen de contenido de directorio ‘defPruebas’

En este caso, los ficheros ‘defPruebas.php’ y ‘gestorDefinir.php’, se corresponden con la capa de presentación, y el resto de los directorios son análogos al caso anterior. En este caso no hay más carpetas porque no existe ningún caso de uso que dependa de este para realizarse.

En resumen, para un caso de uso cualquiera, la estructura es la siguiente:

1. 2 ficheros de capa de presentación.
2. Directorio ‘ficheros_ajax’, si se utiliza AJAX en el caso de uso, donde se encuentran los ficheros relativos a AJAX.
3. Directorio ‘js’, donde se encuentra el código JavaScript relativo al caso de uso.
4. Directorio ‘logica’, donde se encuentran los ficheros de lógica de negocio del caso de uso.
5. Habrá un directorio más por cada caso de uso que dependa del caso de uso en cuestión.

Esta es la estructura que se ha elegido para el proyecto. La otra estructura que se barajó fue la consistente en tener cada capa de la aplicación en un directorio, en lugar de dividir por módulos basados en casos de uso. Es decir, un directorio con todo lo relativo a la capa de presentación, otro directorio con todo lo relativo a la lógica de negocio, y otro directorio con lo relativo a la capa de datos. La decisión se tomó teniendo en cuenta que probablemente se identifiquen más fácilmente los ficheros relativos a cada caso de uso de esta forma. En cualquier caso, la conversión de esta estructura a la otra estructura planteada no supondría apenas esfuerzo.

7.2. Transacciones

Como ya se ha comentado con anterioridad, la consistencia de los datos es fundamental en la aplicación. Por este motivo es imprescindible utilizar transacciones siempre que se escriba en la base de datos (en el caso de que solamente se realicen lecturas, el uso de transacciones es opcional).

En el caso del **GBP**, todas las escrituras se realizan sobre una base de datos MySQL, por eso se explicará el uso de transacciones exclusivamente para este SGBD.

Para interactuar con la base de datos, se ha utilizado el driver nativo de PHP mysqli. A continuación se pone un ejemplo de uso de transacciones, utilizando para ello el caso de uso 'Modificar responsables e implicados'.

Se ha elegido este caso de uso porque es de los casos de uso más sencillos que se pueden encontrar en la aplicación. También conviene reseñar que el código ha sido modificado convenientemente para simplificarlo y poder mostrar de una forma más clara y sencilla todos los pasos que se deben dar a la hora de utilizar transacciones.
A grandes rasgos, en el método modificar se obtiene el identificador del proyecto, para posteriormente borrar los responsables e implicados que estén asociados a dicho proyecto, y finalmente, se introducen los nuevos responsables e implicados. La introducción de los nuevos datos se realiza mediante 2 funciones, de las cuales se ha explicado la que introduce los nuevos responsables en la tabla correspondiente. El código del método que introduce los nuevos implicados no se ha adjuntado, pero es análoga a la función 'insertResponsablesProyecto()'.

```
function modificar($nProyecto, $responsables, $implicados){
      try{
            // Dividimos $nProyecto en los campos pedido y posición
            $nProyecto = explode("-", $nProyecto);
            $posicion = $nProyecto[2];
            $nProyecto = $nProyecto[0]."-".$nProyecto[1];

            $mysqli = new mysqli($servidor, $usuario, $password, $bd);

            if ($mysqli->connect_errno()) {
                  printf("Error al conectar: %s\n",
                                          $mysqli->connect_error);
```

```
      exit();
}

/*Comienza la transacción estableciendo autocommit=false*/
$mysqli->autocommit(FALSE);

// Obtenemos el id del proyecto actual
$sql = "SELECT p.idProyecto
        FROM proyecto p
        WHERE p.nProyecto='".$nProyecto."' AND
                  p.posicion='".$posicion."'";

$result = $mysqli->query($sql);

if ($result !== TRUE) {
      // En caso de error, hacemos rollback
      $mysqli->rollback();
}

// En caso de existir dicho proyecto

// Borramos los responsables e implicados del proyecto
if($row = $result->fetch_array(MYSQLI_ASSOC)){
      $sql = "DELETE
              FROM responsables_implicados
              WHERE idProyecto='".$row['idProyecto']."'";

      $result = $mysqli->query($sql);

      if ($result !== TRUE) {
            // En caso de error, hacemos rollback
            $mysqli->rollback();
      }

      // Introducimos los nuevos responsables e implicados
      $this->insertResponsablesProyecto($idProyecto,
                              $responsables, $mysqli);
      $this->insertImplicadosProyecto($idProyecto,
                              $implicados, $mysqli);
}
```

```
            /* liberar la serie de resultados */
            $result->free();

            /* Finalmente, se hace commit de la transacción
               Si ha habido algún error antes, éste commit dejará
               la base de datos igual que al comienzo */
            $mysqli->commit();

            /* Cerramos la conexión */
            $mysqli->close();

      }catch (Exception $e){
            throw new Exception($e->getMessage());
      }
}

/* Guardar responsables del proyecto */
function insertResponsablesProyecto($idProyecto,
                                    $responsables, $mysqli){
      try{
            $insertSQL ="INSERT INTO responsables_implicados
                              (dni, idProyecto, idStatus) VALUES ";

            /* Función que devuelve el id del status 'responsable' */
               $idStatus = $this->getIdStatus("responsable");

            /* Construye el insert hasta el penúltimo elemento,
               en este momento la consulta termina con coma */
            for ($i = 0; $i < count($responsables)-1; $i++) {
               $insertSQL = $insertSQL."('".$responsables[$i]."',
                              '".$idProyecto."','".$idStatus."'),";
            }

            /* Termina el insert añadiendo el último elemento
               a la consulta.
               En caso de haber un sólo responsable seleccionado,
               el bucle for anterior no se ejecutará y solamente se
               ejecutara el if,
               con lo que la sentencia sql será correcta en todos los
               casos */
            if (count($responsables[$i])>0){
```

```
                $insertSQL = $insertSQL."('".$responsables[$i]."',
                            '".$idProyecto."','".$idStatus."')";

                $result = $mysqli->query($insertSQL);

                if ($result !== TRUE) {
                      // En caso de error, hacemos rollback
                      $mysqli->rollback();
                }
          }
      }catch (Exception $e){
            throw new Exception($e->getMessage());
      }
}
```

Ahora se comentará el código para explicar de forma exhaustiva los pasos más importantes que se dan, relacionados con las operaciones sobre la base de datos. El primer fragmento interesante es este:

```
$mysqli = new mysqli($servidor, $usuario, $password, $bd);

if ($mysqli->connect_errno()) {
      printf("Error al conectar: %s\n", $mysqli->connect_error);
      exit();
}
```

La primera instrucción crea una nueva conexión con la base de datos. Para ello se pasarán 4 parámetros, que determinan la conexión. Dichos parámetros son el servidor en el que se encuentra la base de datos, el usuario con el cual se va a realizar la conexión, la contraseña, y finalmente, la base de datos a la que se va a conectar.

Las siguientes instrucciones se utilizan para el tratamiento de posibles errores, '`$mysqli->connect_errno()`' devuelve el código de error de la última función llamada, y '`$mysqli->connect_error`' devuelve una cadena con la descripción del último error.

A continuación, se inicia la transacción con la instrucción:

```
$mysqli->autocommit(FALSE);
```

Este código actualiza el campo autocommit de la base de datos y lo establece como falso. Dicho campo forma parte de los datos de configuración de la base de datos, y establece el modo de operación que va a tener lugar en una conexión con ella. Si el campo tiene un valor ‘verdadero’ (true), cada operación que se realice sobre la base de datos, se ejecutará como si fuese una transacción, es decir, se realiza un commit implícito con cada operación. Una operación ejecutada con este modo activado, no podrá ser deshecha con la instrucción rollback, por eso todas las transacciones deben comenzar desactivando este modo de operación.

A continuación, se obtiene el identificador del proyecto pasado como parámetro, el cual se utiliza para poder borrar los responsables e implicados de dicho proyecto. Si se produce un error al ejecutar la instrucción, se realizará un rollback que deshaga los cambios realizados. El código que implementa lo explicado es el siguiente:

```
$sql = "DELETE
        FROM responsables_implicados
        WHERE idProyecto='".$row['idProyecto']."'";

$result = $mysqli->query($sql);

if ($result !== TRUE) {
      // En caso de error, hacemos rollback
      $mysqli->rollback();
}
```

Una vez borrados los responsables e implicados, se introducirán los nuevos. La inserción de datos en la base de datos es análoga al borrado. Es decir, se construye la sentencia SQL, se ejecuta y en caso de error se hace rollback.

Para terminar, se realiza el commit mediante la instrucción:

```
$mysqli->commit();
```

La cual guarda las modificaciones realizadas a la base de datos de forma permanente.

Para finalizar, se cierra la conexión mediante la instrucción:

```
$mysqli->close();
```

Esta instrucción es opcional, ya que el servidor Apache cierra automáticamente las conexiones una vez finaliza la ejecución del script.

En resumen, las transacciones constan de 3 partes fundamentales:

- Se comienza la transacción estableciendo el modo autocommit como 'false'.
- Se ejecutan las sentencias SQL sobre la base de datos, y si hay error, se ejecuta la instrucción rollback.
- Finalmente, se ejecuta la instrucción commit.

7.3. Sesiones PHP

Las sesiones, en el desarrollo aplicaciones web, sirven para almacenar alguna información que será accesible durante toda la visita de un usuario a una página web. Es decir, un usuario puede visitar varias páginas durante su paso por un sitio web, y con sesiones es posible almacenar variables que el usuario podrá acceder en cualquiera de esas páginas.

De esta forma, podría decirse que las sesiones almacenan información específica para cada usuario, durante toda su visita a un sitio web. Las sesiones son independientes entre sí, es decir, cada usuario abre su propia sesión al acceder a un sitio web. En la sesión de un usuario se almacenan datos a los que es interesante poder acceder en cualquier momento, como por ejemplo, el nombre, historial de páginas, productos del carrito de la compra, preferencias de visualización, preferencias del idioma, etc. Toda esta información se guarda en las variables de sesión.

Una sesión se crea en el momento en el que un usuario accede al sitio web, y se destruye en el momento en el que el usuario lo abandona.

En el caso del **GBP**, en la variable de sesión solamente se almacenan 3 datos: el nombre del usuario, la línea a la que pertenece, y el status que tiene este usuario dentro de la aplicación. Estos 3 datos son los más importantes para un usuario que esté dentro de la aplicación, y se muestran siempre encabezando la página. De esta forma, con el status en la variable sesión se puede hacer una gestión de los permisos que tiene el usuario para acceder a una página, comprobando simplemente que el

status de la variable de sesión tenga permisos para ver esa página. Las variables de sesión que contienen el nombre del usuario y la línea, se utilizan sobre todo a la hora de construir consultas SQL, ya que el dato de la línea del usuario se usa constantemente en las consultas y de esta forma lo tienes accesible continuamente, con lo que las consultas son más simples y es más cómodo construirlas.

Al acceder a la página principal de la aplicación, la sesión del usuario se crea mediante la instrucción

```
session_start();
```

Dicha instrucción debe ser siempre la primera en el script PHP, e inicia una nueva sesión o reanuda la sesión existente, en caso de que la haya. Existen 2 formas de propagar un id de sesión. La primera forma consiste en guardar el id de sesión en una cookie, y la segunda forma consiste en propagar el id de sesión en la url del navegador.

Almacenar el id en una cookie es lo más cómodo, además de ser lo que está configurado por defecto. En este caso, el desarrollador no tiene que hacer nada, PHP es el que se encarga de almacenar la información de la sesión en una cookie en el navegador del usuario. El problema que tiene este método, es que puede haber usuarios que tengan las cookies bloquedas en la configuración de su navegador.

El segundo método consiste en pasar el id de la sesión en todas las url de la aplicación como parámetro, para posteriormente acceder a él mediante ‘GET’. La ventaja de este método es que siempre será posible utilizarlo, independientemente de la configuración del navegador del cliente. Sin embargo, tiene importantes inconvenientes, ya que enviar el id de sesión de usuario en la URL puede hacer público dicho id de sesión.

Para este proyecto, se ha decidido utilizar cookies de sesión, ya que es lo más cómodo y común en la actualidad.

7.4. JavaScript

La mayor parte del código JavaScript del proyecto está realizado en JavaScript puro, es decir, sin utilizar ninguna librería de las muchas que hay, por ejemplo jQuery. Más adelante se explicará que en ocasiones puntuales sí se recurre a jQuery, ya que permite hacer operaciones complejas con mucho menos código que realizándolo en JavaScript puro. Sin embargo, se ha optado por esta opción debido a que solamente lo había utilizado en ocasiones puntuales, por lo que se ha considerado oportuno implementar el proyecto en JavaScript para aprender en profundidad su uso. De esta forma, el proceso de aprendizaje resulta más natural, aprendiendo primero a utilizar JavaScript con soltura y posteriormente, en caso de ser necesario, aprender a utilizar alguna librería como pueden ser jQuery o Prototype.

En el proyecto se ha utilizado JavaScript por ejemplo para desactivar el desplegable 'Nº de serie' y los botones en el menú 'Abrir proyecto', cuando no haya ningún proyecto seleccionado, ya que si no hay ningún proyecto seleccionado, no tiene sentido que se permita la opción de seleccionar número de serie, al igual que tampoco tiene sentido realizar cualquiera de las operaciones implementadas:

Imagen 7.6. Desplegable y botones desactivados

Y aquí vemos cómo quedaría el menú después de seleccionar un número de proyecto y un número de serie de dicho proyecto:

Imagen 7.7. Desplegable y botones activados por JavaScript

También se utiliza de forma intensiva JavaScript en el caso de uso 'Definir pruebas'. Por ejemplo, todos los intercambios de pruebas entre la tabla de pruebas asignadas y la tabla de pruebas posibles se realizan mediante JavaScript, borrando e insertando la fila donde corresponda. Las operaciones en AJAX también se realizan con JavaScript.

7.5. AJAX

Éste es el código mediante el cual se crea el objeto XMLHttpRequest en el fichero JavaScript:

```
var xmlHttp = crearXmlHttpRequest();

//crear el objeto XMLHttpRequest
function crearXmlHttpRequest(){
	// esta variable local
	// guarda la referencia al objeto XMLHttpRequest
	var xmlHttp;

	//para ejecutar en Internet Explorer
	if(window.ActiveXObject) {
		try {
			xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

```
            }
            catch (e) {
                  xmlHttp = false;
            }
      //otros navegadores
      }else{
            try {
                  xmlHttp = new XMLHttpRequest(); }
            catch (e) {
                  xmlHttp = false;
            }
      }

      //devolver el objeto creado, si no, enseñar un mensaje de error
      if (!xmlHttp){
            alert("Ha ocurrido un error al crear XMLHttpRequest.");
      }else{
            return xmlHttp;
      }
}
```

En el código anterior vemos cómo se tiene en cuenta la distinción entre los navegadores de la familia Internet Explorer y el resto de navegadores. A pesar de que nuestra aplicación solamente tiene que funcionar con IE8 y posteriores versiones, siempre se ha tenido en mente la compatibilidad con todos los navegadores, ya que una de las posibles tareas a hacer en el futuro es implementar la compatibilidad con el mayor número de navegadores posible. De esta forma, se intenta que dicho trabajo futuro en caso de considerarse oportuno que se lleve a cabo, se pueda realizar con la menor cantidad de esfuerzo posible.

Ahora el objeto está creado, por lo que se le hace una petición asíncrona al servidor, en este caso utilizando GET. El código es el siguiente, y se encuentra también en el fichero Javascript:

```
//------ Ajax para obtener las revisiones de HD de un proyecto -------

function getNumSeriePedido(){
      numPedido = document.getElementById("cbProyectosAbiertos");
      numPedido = encodeURIComponent(numPedido.value);
      xmlHttp.open("GET", "getNumSerie_ajax.php?numPedido="+
                              numPedido, true);

      xmlHttp.onreadystatechange = procesarNumSerie;
      xmlHttp.send(null);
}
```

Del código anterior se pueden comentar varias cosas. En primer lugar, se observa que el objeto XMLHttpRequest tiene un método llamado ‘open()’. Este método prepara una conexión http a través del objeto que le llama, con un método y una URL especificada.

A continuación se explica la sintaxis y los parámetros del método open:

```
XMLHttpRequest.open(método, URL, asincronía, usuario, clave)
```

- método: Es el tipo de petición http. Puede tomar los valores ‘GET’, ‘POST’ O ‘PUT’. En la aplicación solamente usamos ‘GET’ y ‘POST’, y en Ajax generalmente se suele utilizar ‘GET’, salvo excepciones.
- URL: URL a la que se realiza la petición http.
- asincronía: Es un campo booleano que indica si la conexión se va a realizar de forma asíncrona (true) o síncrona (false). En AJAX siempre se utiliza conexión asíncrona, por lo que el valor de este parámetro siempre será ‘true’.
- usuario: Usuario para la identificación en la URL destino. Es un parámetro opcional que en este caso no se utiliza.
- clave: Clave para la autentificación en la URL destino. Es un parámetro opcional que en el ejemplo no se utiliza.

A continuación, se encuentra el atributo onreadystatechange del objeto XMLHttpRequest, el cual almacena el nombre de la función que se ejecuta cada vez que el estado de la conexión cambie. Es en esta función donde se trata la respuesta recibida.

Por último, se observa el método ‘send()’ del objeto, mediante el cual se envía el contenido al servidor. En el caso de utilizarse el método ‘POST’ para enviar los parámetros, la función send deberá llevar como parámetro un string que represente los parámetos que se van a enviar por ‘POST’, y en el caso de ‘GET’, el parámetro será null.

Con esto ya está enviada la petición al servidor. Ahora falta el procesamiento de dicha petición y la generación de una respuesta. Esto es lo que se hace en el fichero ‘getNumSerie_ajax.php’:

```
<?php
	session_start();
	header ("Content-Type: text/xml");
	header( "Cache-Control: no-cache, must-revalidate" );
	// fecha en el pasado
	header( "Expires: Mon, 26 Jul 1997 05:00:00 GMT" );

	define ("__PATH__", dirname(dirname(dirname(__FILE__))));
	require_once (__PATH__.'/BBDD/BBDD_consultas_ORACLE.php');

	$numPedido = $_GET["numPedido"];
	if($_SESSION['linea']=='hydro'){
		// Si está en nomenclatura antigua, quita la I
		$numPedido = str_replace("I", "", $numPedido);
		$resul = obt_numSerie($numPedido);
	}else{
		$nPedido = 'S'.$numPedido;
		$resul = obt_numSerie($nPedido);

		// Si no se obtienen números de serie con este
		// número de pedido, probar con la nomenclatura antigua
		if(count($resul) == 0){
			$nPedido = 'C'.$numPedido;
			$resul = obt_numSerie($nPedido);
		}
	}

	//  Se crea el XML con el contenido
	//  del resultado de la consulta
	echo '<?xml version="1.0" encoding="iso-8859-15" ?>';
```

```
	echo "<content>";
	foreach ($resul as $nSerie){
		echo "<numSerie>".$nSerie."</numSerie>";
	}
	echo "</content>";
?>
```

En el código anterior se pueden encontrar cosas interesantes que deben ser explicadas. Justo a continuación de la instrucción session_start(), que como se ha explicado anteriormente, debe ser la primera de la página, se encuentran 3 instrucciones que son:

```
header ("Content-Type: text/xml");
header( "Cache-Control: no-cache, must-revalidate" );
// fecha en el pasado
header( "Expires: Mon, 26 Jul 1997 05:00:00 GMT" );
```

Estas instrucciones se utilizan para enviar encabezados HTTP sin formato. En estos encabezados se indica que el contenido es de tipo XML, que el fichero no debe ser cacheado, y una fecha en el pasado. Con las 2 últimas instrucciones ‘header()’ lo que se consigue es que no se almacene la página en la caché del navegador, ya que en caso contrario, si se llama más de una vez a la función que obtiene los números de serie de un proyecto dado, solamente se ejecutará la primera vez, y en las llamadas posteriores el fichero se encontrará en la caché del navegador, y el resultado es que o no se ejecuta o se ejecuta incorrectamente.

Otra peculiaridad de esta instrucción es que debe situarse en el código antes de mostrar nada en pantalla. Un error muy común es hacer la llamada a funciones como include o require antes de la instrucción header. En este caso, la respuesta XML será incorrecta.

Se han explicado de forma extensa las particularidades de la función header a la hora de crear respuestas XML en Ajax porque fueron problemas que supusieron una pérdida de tiempo importante hasta que finalmente pudieron ser corregidos.

Seguidamente, se pueden ver las instrucciones

```
define ("__PATH__", dirname(dirname(dirname(__FILE__))));
require_once (__PATH__.'/BBDD/BBDD_consultas_ORACLE.php');
```

que se utilizan para incluir el fichero de capa de datos que se conecta a la base de datos Oracle y obtiene los números de serie asociados a un proyecto dado. El define se utiliza para definir la ruta raíz del proyecto a partir de la dirección del fichero actual, que está contenida en la variable __FILE__. Después, se importa el fichero deseado concatenando el directorio donde se encuentran los ficheros correspondientes a la capa de datos de la aplicación. De esta forma, se importa el fichero haciendo uso de la ruta absoluta.

Se hace hincapié en ello porque inicialmente se importaban los ficheros mediante rutas relativas, lo cual en ocasiones dio problemas ya que PHP considera como ruta base sobre la que se realiza el direccionamiento relativo, no la ruta del fichero donde se escribe la instrucción en cuestión, si no la ruta del fichero desde donde el fichero en cuestión es llamado. Esta particularidad era desconocida para mí, ya que en Java, C y C++ se toma la ruta del fichero como la ruta a partir de la cual se realiza el direccionamiento relativo, y también me llevó un buen rato percatarme del error y solucionarlo.

A continuación, se puede ver en el fichero getNumSerie_ajax.php la forma de obtener los números de serie dependiendo de la unidad de negocio. Las operaciones que se realizan para obtener los números de serie no se explican, ya que tienen que ver con cómo se almacenan en SAP los nombres de proyecto, y cómo se almacenan en las bases de datos propias de cada unidad de negocio así que explicarlo no aporta nada desde el punto de vista de comprender el funcionamiento de Ajax.

Al final del fichero, vemos como mediante instrucciones echo se forma el fichero XML en el que se almacena la respuesta. Dicho fichero tiene un aspecto parecido a este, suponiendo un proyecto que tiene 3 números de serie asociados:

```
<?xml version="1.0" encoding="iso-8859-15" ?>
<content>
      <numSerie>3010000300</numSerie>
      <numSerie>3010000301</numSerie>
      <numSerie>3010000302</numSerie>
</content>
```

Este fichero XML constituye la respuesta que envía el servidor al cliente una vez procesada su petición.

```
function procesarNumSerie() {

if (xmlHttp.readyState == 4){

if (xmlHttp.status == 200) {
      respuestaXML = xmlHttp.responseXML;
      documentoXML = respuestaXML.documentElement;

      //Rellenar desplegable NumSerie
      var lista= documentoXML.getElementsByTagName("numSerie");
      var select= document.getElementById("nSerie");
      select.disabled=false;
      var option;
      var textnode;
      select.innerHTML="";

      option=document.createElement("option");
      textnode=document.createTextNode("");
      option.appendChild(textnode);
      select.appendChild(option);
      select.options[0].value = textnode.nodeValue;

      // Si el xml tiene números de serie los metemos en las opciones
      // manipulando el DOM mediante JavaScript
      if (lista.length>0){
            for (var i=0; i<lista.length; i++){
                  option  =document.createElement("option");
                  texto = lista[i].firstChild.nodeValue;
                  textnode = document.createTextNode(texto);
                  option.appendChild(textnode);
                  select.appendChild(option);

                  select.options[select.options.length-1].value =
textnode.nodeValue;
            } // Fin for

      // Si no tiene números de serie desactivamos el
      // select de números de serie y
```

```
      // los botones de definir cabeceras y pruebas
      // Este caso es cuando elegimos el proyecto
      // vacío en el select de proyectos
      }else{
            select.disabled = true;
            botonEntradaDatos.disabled = true;
            botonDefinirPruebas.disabled = true;
      }

      numPedido = document.getElementById("cbProyectosAbiertos");
      numPedido = encodeURIComponent(numPedido).value);

      var botonCabecera = document.getElementById("btn_defCabecera");
      var botonResImp = document.getElementById("btn_defCabecera");

      // Si no hay número de pedido seleccionado
      if (numPedido!=''){
            botonCabecera.disabled = false;
            botonResImp.disabled = false;
      }else{
            botonCabecera.disabled = true;
            botonResImp.disabled = true;
      }
}
      // HTTP status-a ez bada 200, errorea gertatu da
      else {
           alert("Zerbitzaria atzitzean arazoren bat dago: " +
                  xmlHttp.statusText);
        }
    }
}
```

En primer lugar se puede observar que el objeto XMLHttpRequest dispone de un atributo llamado ‘readyState’ que indica el estado del objeto mediante un valor entero. Hay 5 posibles valores:

- 0: Petición sin inicializar.
- 1: Conexión establecida con el servidor.
- 2: Petición recibida.
- 3: Procesando petición.

- 4: Petición completada y la respuesta está preparada.

Generalmente con tener en cuenta el estado ‘4’ es suficiente.

Otro atributo que se puede observar es el atributo status, el cual contiene un código enviado por el servidor, del tipo ‘404’ (no encontrado) o ‘200’ (OK). Generalmente, con tener en cuenta el estado OK es suficiente. De este modo cuando la respuesta está completada (readyState=4) y de forma correcta (status=200) se puede proceder a tratar dicha respuesta.

De forma resumida, el procedimiento que sigue la función que trata la respuesta es el siguiente:

1. Guarda en una lista los elementos que están contenidos con el tag ‘numSerie’ en el XML, que en este caso son los números de serie del proyecto.
2. La primera opción del select ‘Nº serie’ es el string vacío.
3. Para cada elemento de la lista, se crea un nodo con el valor del número de serie y se une al nodo option, que a su vez se une al nodo select.
4. Si no hay ningún número de serie, el select ‘Nº serie’ se desactiva, y los botones de ‘Entrada de datos’ y ‘Definir pruebas’ también.
5. Si no hay pedidos seleccionados, se desactivan todos los botones que pueden estar activados en ese momento.

Y estos son los pasos que sigue una consulta al servidor mediante AJAX. Se ha intentado explicar con toda la claridad posible de forma genérica las fases de una petición mediante AJAX, unido a la explicación de partes del código que se consideran relevantes bien por su relación con AJAX o bien porque en su día se perdió bastante tiempo con algunas situaciones de las comentadas, por lo que se considera interesante mencionarlas, ya que aunque no aportan nada a la comprensión del funcionamiento de AJAX, quizás puedan ahorrar tiempo al lector si algún día decide realizar alguna aplicación de este tipo.

7.6. jQuery

Esta librería JavaScript la hemos utilizado para 2 cosas principalmente: realizar algún efecto elaborado como puede ser mostrar un calendario para introducir la fecha, o la validación de formularios.

Los ficheros jQuery se pueden añadir de 2 formas: por un lado, se puede importar directamente indicándolo de esta forma en el documento HTML:

```
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"
type="text/javascript"></script>
```

Ésta es la opción más cómoda, pero presenta dos inconvenientes muy importantes:

1. En el caso de que no haya conexión a Internet por el motivo que sea, no se podrá importar la librería.
2. Realmente no se tiene el control de la librería, ya que si Google estima oportuno suprimir dicha librería, se tendrá un enlace roto, y tampoco se podrá importar el fichero, con el consiguiente perjuicio para la aplicación.

Por eso, se consideró que la mejor opción era descargar la librería para tenerla accesible dentro del propio servidor de la aplicación, con lo que los problemas anteriores se solucionan. La versión que se ha descargado y se utiliza en la aplicación es la 1.7.2.

A continuación explicaremos brevemente la validación de formularios mediante jQuery.

Esta función se realiza con el plugin llamado ‘Validation’, el cual permite de una forma simple validar los formularios. Las condiciones que debe cumplir un campo de tipo ‘input’ para ser validado se especifican mediante el atributo class.

Es decir, si se desea que un campo sea obligatorio, el atributo class del campo ‘input’ debe ser class=’required’. Si se desea que además de obligatorio sea un número, el atributo class debe ser class=’required number’. En la documentación del plugin pueden conocerse todos los tipos de validaciones que vienen predefinidas, sin embargo, también pueden establecerse reglas que se adapten a las necesidades

específicas del desarrollador. En el caso del **GBP**, se tuvo que definir una regla para validar que una hora introducida tenga el formato correcto.

La sintaxis para definir un método de validación propio es:

```
addMethod(nombre_del_metodo,funcion_anonima,_mensaje_error)
```

Y el código que implementa la validación de una hora es el siguiente:

```
// Comprueba que la hora tenga el formato correcto
// El mensaje está en inglés para mantener la coherencia con el resto
$.validator.addMethod("time", function(value, element) {
      return this.optional(element)
            || /^\d{1,2}:\d{2}([ap]m)?$/i.test(value);
}, "Please enter a valid time.");
```

Al principio puede parecer complicado, pero con un poco de práctica resulta bastante natural la sintaxis de jQuery.

Una vez están todas las reglas definidas, simplemente se valida el formulario con la instrucción:

```
$("#formulario").validate()
```

A continuación se muestran unas imágenes con los resultados de las validaciones. En la imagen 7.8 se muestra el resultado de la validación cuando hay campos obligatorios vacíos, y en la imagen 7.9 se muestra el resultado de la validación cuando hay un campo vacío y un campo 'fecha' con el formato de fecha incorrecto.

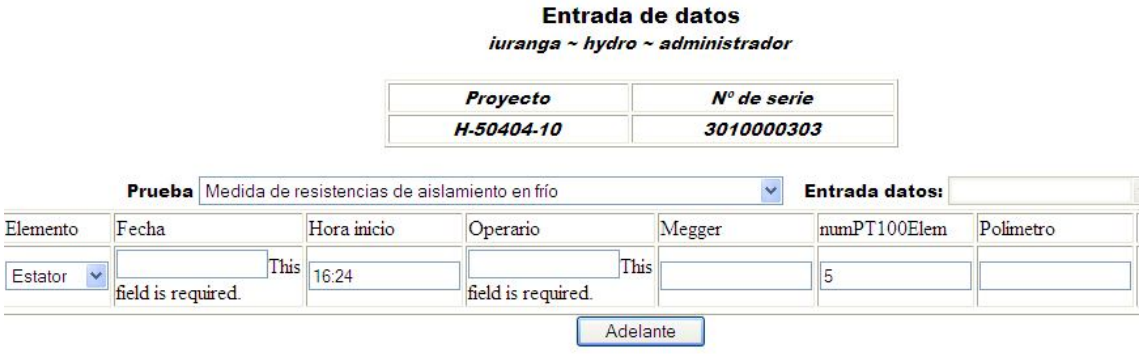


Imagen 7.8. Validación con campos obligatorios vacíos

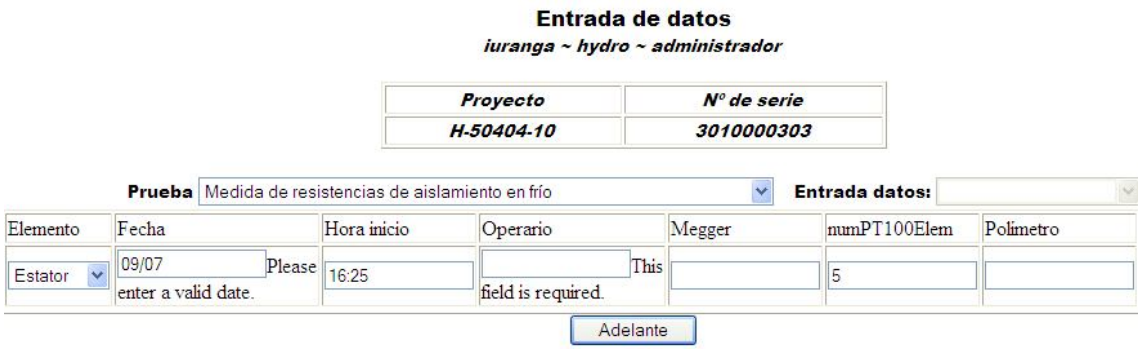


Imagen 7.9. Validación con formato de fecha incorrecto

También se ha comentado que otro uso de jQuery es realizar efectos visuales de forma sencilla, como es el caso de mostrar un calendario para solicitar la fecha a un usuario. Para ello se utiliza el plugin 'Datepicker', mediante el cual se muestra un calendario simplemente añadiendo el siguiente código JavaScript a la página:

```
// Obtiene todos los elementos que tengan
// definido el atributo class="datepicker"
$(".datepicker").datepicker({
      inline: true
});
```

Este código mostrará un calendario en todos los campos de la página que tengan el atributo class con valor 'datepicker'. El resultado es el siguiente:

Imagen 7.10 Calendario generado con jQuery para el campo 'Fecha'

8. PRUEBAS

8.1. Conceptos generales

Una de las últimas fases del ciclo de vida del software antes de entregar un programa al cliente para su explotación, es la fase de pruebas, la cual es además una de las más costosas del ciclo de vida. En la actualidad, se estima que esta fase representa la mitad del esfuerzo en el desarrollo de un programa. Esto se debe a que los fallos en el software pueden ocasionar grandes pérdidas económicas e incluso humanas. En el caso del **GBP**, al tratar la información obtenida en Banco de Pruebas para todos los generadores síncronos, es fundamental que funcione correctamente, ya que la inconsistencia o pérdida de estos datos sería un grave problema.

Estas son los términos que más se utilizan en esta fase:

- Prueba: Actividad en la cual se somete a un sistema o a uno de sus componentes a una evaluación de los resultados que arroja en base a la ejecución de éste en condiciones especificadas.

- Caso de prueba: Conjunto de entradas y condiciones que arrojan resultados esperados desarrollados con un objetivo en particular. Un buen caso de prueba es aquel que tiene una alta probabilidad de descubrir un error no encontrado hasta entonces.

- Error: Acción humana que produce o genera un resultado incorrecto.

- Defecto: Es la manifestación de un error en el software.

- Verificación: Determinar si los productos de una fase dada satisfacen las condiciones impuestas al inicio de la fase. ¿Estamos construyendo correctamente el producto?

- Validación: Evaluación de un sistema o uno de sus componentes durante o al final de su desarrollo para determinar si satisface los requisitos. ¿Estamos construyendo el producto correctamente?

La prueba ideal de un sistema sería exponerlo a todas las situaciones posibles, pues de esta forma se encontrarían todos los errores, garantizando su respuesta correcta ante cualquier caso que se presente en una ejecución real. En la práctica, esto es imposible ya que siempre se dispone de plazos de entrega que limitan el tiempo disponible, y tampoco se dispone de personal para llevar a cabo unas pruebas totales de cualquier software de una cierta complejidad. Por lo tanto, los factores económico y humano hacen que probar de forma total una aplicación sea una utopía.

Sin embargo, se someterá al proyecto a una serie de pruebas con el fin de encontrar y subsanar fallos existentes, consiguiendo prevenir al sistema del mayor número de fallos posible. Lograremos también confianza acerca del nivel de calidad del producto que se ha desarrollado, afirmando que funciona correctamente y según los requisitos de la empresa.

Para ser más efectivas, las pruebas deberían ser conducidas por un equipo independiente, ya que el criterio del propio desarrollador del código se encuentra sesgado en mayor o menor medida, al ser el programa a probar de su creación. Sin embargo, la mayor parte de las veces, por limitaciones de personal y/o tiempo principalmente, es el propio desarrollador el que se encarga de realizar las pruebas. En este caso, he sido yo el que se ha encargado de probar el software, aunque antes de utilizar la aplicación en un entorno productivo, se utilizará y probará en un entorno de preproducción por los usuarios finales de la aplicación, con el fin de detectar errores y sugerir posibles mejoras que consideren oportunas.

El desarrollo de este proyecto está basado en ciclo de vida en cascada, como muestra la siguiente imagen.

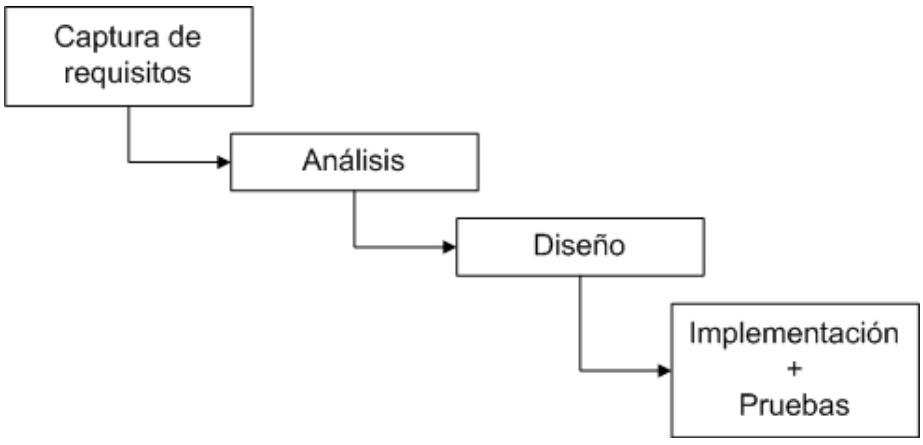


Imagen 8.1 - Desarrollo en cascada

Estrechamente ligado al modelo en cascada está el modelo en V, puesto que es una evolución del mismo. Este modelo involucra comprobaciones de cada una de las etapas del modelo de cascada una vez terminado el proceso de codificación.

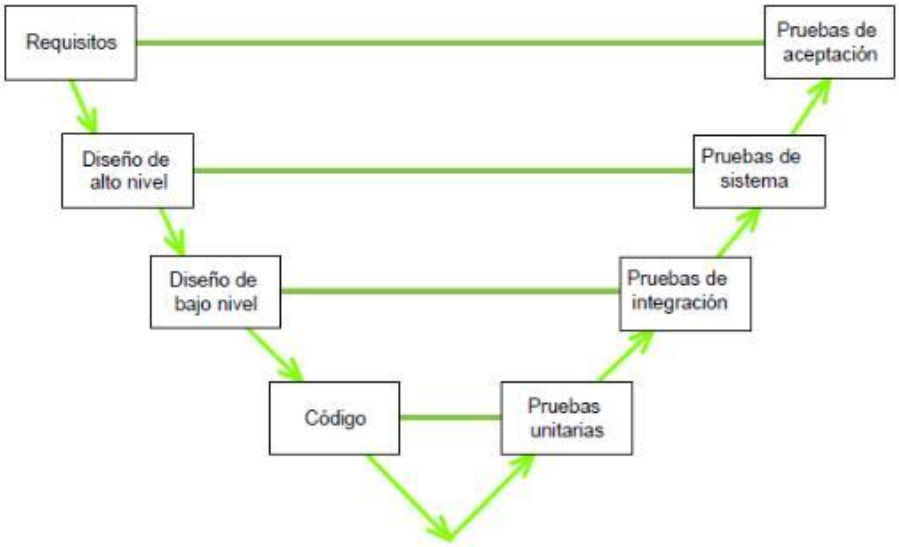


Imagen 8.2 - Modelo en V

En las siguientes secciones se realiza un análisis sobre las pruebas a las que se ha sometido al sistema:

8.2. Pruebas unitarias

Es la prueba de cada módulo, que normalmente realiza el propio personal de desarrollo en su entorno. Se prueba el correcto funcionamiento de los módulos del código, asegurando así el perfecto funcionamiento de cada módulo por separado. Se encargan de analizar que un único componente de la aplicación (función o método) devuelva resultados correctos de acuerdo a ciertas entradas.

Para cada caso de uso del **GBP** se han realizado las siguientes pruebas principalmente. Como puede observarse, hay una cierta metodología con respecto a las pruebas realizadas. Decir también que estas son las pruebas conceptualmente más importantes, pero durante el propio desarrollo del código se han realizado un gran

número de pequeñas pruebas para comprobar en la medida de lo posible que se estaba avanzando en la dirección correcta.

1. Identificar Usuario
    a. Entrar a la aplicación con un usuario correcto.
    b. Entrar a la aplicación con un usuario incorrecto.

2. Crear
    a. Comprobar que los números de proyecto obtenidos, son correctos.
    b. Comprobar que el valor de la última revisión es correcto. La consulta a la base de datos tiene que tener en cuenta que las revisiones preliminares se notan con una letra mayúscula, mientras que las revisiones de construcción de proyecto son números enteros. Se debe devolver el mayor número entero.
    c. Comprobar que el resumen obtenido de un proyecto dado, es correcto.
    d. Dar de alta un proyecto.
    e. Dar de alta un proyecto con responsabes e implicados.
    f. Dar de alta un proyecto de tal forma que falle la transacción, para comprobar que realmente el estado de la base de datos mantiene la consistencia.
    g. Dar de alta un proyecto que no tenga la nomenclatura estándar de INDAR.
    h. Comprobar las distintas excepciones que pueden darse en este caso de uso.

3. Abrir
    a. Comprobar que la obtención de proyectos dados de alta y sin cerrar es correcta.
    b. Comprobar que la obtención de números de serie de la base de datos Oracle de SAP es correcta.
    c. Comprobar las distintas excepciones que pueden darse en este caso de uso.

4. Definición de pruebas
    a. Comprobar que las pruebas asignadas a un número de serie son correctas.
    b. Comprobar que las pruebas posibles de un generador son correctas.
    c. Comprobar que el número de revisión es correcto.

d. Comprobar que los botones que cambian las pruebas de asignadas a posibles o viceversa funcionan correctamente.
e. Definir pruebas para un número de serie
f. Comprobar que no es posible definir pruebas si hay alguna prueba a asignar que no tenga un tipo asociado.
g. Definir pruebas de tal forma que la transacción no se ejecute correctamente.
h. Comprobar las distintas excepciones que pueden darse en este caso de uso.

5. Copiar pruebas
    a. Comprobar que no se puede copiar pruebas de un número de serie al mismo número de serie.
    b. Copiar pruebas de un generador a otro.
    c. Copiar pruebas de un generador a otro de tal forma que falle la transacción.
    d. Comprobar las distintas excepciones que pueden darse en este caso de uso.

6. Intercambiar pruebas
    a. Comprobar que no se puede intercambiar pruebas de un generador con el mismo generador.
    b. Intercambiar pruebas entre dos generadores.
    c. Intercambiar pruebas entre dos generadores de tal forma que la transacción se ejecute incorrectamente.
    d. Comprobar las distintas excepciones que pueden darse en este caso de uso.

7. Definición de datos de cabecera
    a. Comprobar que los tipos de cabecera posibles son los correctos.
    b. Comprobar que el formulario generado es el correcto.
    c. Comprobar si el número de revisión es correcto.
    d. Comprobar si los datos obtenidos de las Hojas de Diseño o de las revisiones anteriores son correctos.
    e. Definir datos de cabecera.
    f. Definir datos de cabecera de tal forma que falle la transacción.
    g. Comprobar que el tipo de ventilación del generador es correcto.
    h. Comprobar que la forma constructiva del generador es correcta.

i. En cabecera general, comprobar que la intensidad de corriente es correcta.
j. Validación de formularios.
k. Comprobar las distintas excepciones que pueden darse en este caso de uso.

8. Entrada de datos
    a. Comprobar que el formulario generado es el correcto.
    b. Entrada de datos.
    c. Entrada de datos de tal forma que falle la transacción.
    d. Comprobar las distintas excepciones que pueden darse en este caso de uso.

9. Modificar responsables e implicados
    a. Comprobar que los responsables del proyecto son correctos.
    b. Comprobar que los implicados del proyecto son correctos.
    c. Comprobar que los responsables posibles son correctos.
    d. Comprobar que los implicados posibles son correctos.
    e. Modificar responsables e implicados
    f. Modificar responsables e implicados de tal forma que la transacción se ejecute incorrectamente.
    g. Comprobar las distintas excepciones que pueden darse en este caso de uso.

10. Cerrar/Reabrir proyecto
    a. Comprobar que la obtención de los proyectos abiertos es correcta.
    b. Comprobar que la obtención de los proyectos cerrados es correcta.
    c. Comprobar que no se pueda pulsar el botón ‘Cerrar’ si no hay proyectos abiertos.
    d. Comprobar que no se pueda pulsar el botón ‘Abrir’ si no hay proyectos cerrados.
    e. Comprobar las distintas excepciones que pueden darse en este caso de uso.

11. Modificar lista master de pruebas
    a. Comprobar que la lista de pruebas obtenida es correcta.
    b. Comprobar que el número de revisión es correcto.

c. Comprobar que el desplegable de revisiones anteriores se genera correctamente.
d. Modificar lista master de pruebas.
e. Modificar lista master de pruebas de tal forma que la transacción no se ejecute correctamente.
f. Comprobar las distintas excepciones que pueden darse en este caso de uso.

Aunque, como se ha comentado anteriormente, probar un software totalmente es una utopía, se han realizado todas las pruebas arriba indicadas, y otras pequeñas pruebas que no se han comentado. Algunos de los errores que se han encontrado y solucionado son los siguientes:

2. Crear
    a. En el caso de la línea de negocio 'Cim', los números de proyecto obtenidos no eran del todo correctos, ya que había algunos nombres de proyecto que deberían salir y no lo hacían.
    c. En el caso de la línea de negocio 'Cim', al obtener mediante AJAX el resumen del proyecto, la configuración de la máquina (la cual puede ser vertical u horizontal), era incorrecta, pues no se obtenía de los campos de la base de datos que utilizaba en la aplicación, si no de un campo llamado IM[11] al cual había que aplicarle cierta lógica según la norma existente. Además, y para ambas unidades de negocio, al obtener el dato 'Nombre de obra' del resumen, mediante AJAX, había algunos casos muy aislados en los que el nombre de obra contenía algún carácter reservado de XML, por lo que la respuesta XML generada por el servidor no era correcta, y el resumen no se mostraba correctamente.
    g. En pedidos antiguos, es posible que la nomenclatura de los pedidos cambie, por lo que en algunos casos al dar de alta alguno de estos pedidos surgía un error.

7. Definir datos de cabecera
    a. Para un proyecto dado, sólo había que mostrar la opción de rellenar la información de electros si la refrigeración secundaria de la máquina se realiza con agua.

[11] International Mounting, norma internacional en la cual se especifican todas las formas posibles de montar una máquina, asociándole un código a cada forma de montaje.

Errores detectados que no se corresponden con ningún caso de uso en especial, si no que afectaban a la aplicación de forma general:

1. Almacenar algún campo de tipo string que supera el tamaño introducido en la declaración de la columna de base de datos y se produce un error al intentar guardar el contenido. Se soluciona añadiendo un limitador '*maxlength'* en la etiqueta de la caja de texto dentro del código HTML.

2. Algunos ficheros AJAX daban problemas debido a que se quedaban almacenados en la caché del navegador, y por eso solamente se ejecutaban la primera vez. Se soluciona especificando en las cabeceras de dichos ficheros, que no serán cacheados (no-cache).

8.3. Pruebas de integración

Aunque previamente se haya demostrado que los módulos funcionan correctamente mediante pruebas unitarias, puede darse el caso de que varios módulos al trabajar de forma conjunta ocasionen algún tipo de error. Por esa razón se realizan este tipo de pruebas, para asegurar que los módulos que están relacionados funcionen de forma conjunta correctamente.

En realidad, en el **GBP** los módulos son independientes en gran medida, por lo que la mayor parte de las comprobaciones han estado orientadas hacia la conservación de la consistencia de los datos en la base de datos, cuando hay muchos proyectos de varias unidades de negocio dados de alta, cada uno de los cuales teniendo también un buen número de datos asociados.

8.4. Pruebas del sistema

La fase de pruebas del sistema tiene como objetivo verificar el sistema software para comprobar si este cumple sus requisitos.

Para probar el sistema se han realizado las siguientes pruebas:

- Pruebas de contenido, verificando que las palabras usadas para transmitir una idea al usuario sean las adecuadas. Por ejemplo, se decidió cambiar el nombre

inicial de ‘Permutar pruebas’ por ‘Intercambiar pruebas’, ya que se consideró que la segunda denominación era más clara y concisa que la primera.

- Pruebas de funcionalidad. Este tipo de pruebas examina si el sistema cubre sus necesidades de funcionamiento, acorde a las especificaciones de diseño. Para estas pruebas se han utilizado los esquemas de pruebas de caja negra ya que nos interesa saber si funciona o no, independientemente de la forma en que lo haga. Las pruebas de caja negra son un tipo de prueba básico y fácil de comprender, que consiste básicamente en desplegar la aplicación e ir navegando por las páginas comprobando que se obtiene el resultado deseado tras completar cada formulario que encontremos y pulsar cada botón que veamos. Mediante estas pruebas se descubren la mayor parte de los errores, puesto que suelen ser errores evidentes, fácilmente detectables y cuya solución es necesaria. El único requisito es ser metódico para probar todos los casos que pueden darse en la aplicación. Al ser errores evidentes, deben ser corregidos de forma prioritaria antes de realizar pruebas más exhaustivas.

  Uno de los problemas que lleva implícitos el desarrollo de esta aplicación es la dificultad de determinar si la salida del programa es correcta o incorrecta, debido a los datos de carácter eléctrico y mecánico que gestiona la aplicación y los cuales desconozco. Para ello he necesitado que las salidas de estas pruebas sean revisadas por personal de la empresa.

- Pruebas de usabilidad. Tienen la finalidad de verificar como de fácil de usar el un sistema. Para realizar estas pruebas, así como las de contenido y funcionalidad, tenían lugar reuniones en las cuales se hacía una pequeña demostración sobre el funcionamiento de la aplicación, con el fin de detectar errores o sugerir posibles mejoras.

- Pruebas de aceptación: Estas pruebas las realiza el cliente. Son pruebas sobre todo el sistema completo y después de haber realizado todas las pruebas de integración. El cliente comprueba en su propio entorno de explotación si lo acepta tal y como está o no.

9. GESTIÓN DEL PROYECTO

Al final de todo proyecto se ha de realizar una contabilidad de los recursos consumidos y su desviación sobre lo inicialmente planificado.

9.1. Parte de horas mensuales

A continuación, se muestra un desglose mensual de las horas invertidas a lo largo de los meses del proyecto.

FEBRERO

TAREA	TOTAL HORAS
DOP	10
ESTUDIO DEL ENTORNO + HERRAMIENTAS	48
CAPTURA REQUISITOS	32
ANÁLISIS ITER1	0
DISEÑO ITER1	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER1	0
ANALISIS ITER2	0
DISEÑO ITER2	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER2	0
ANÁLISIS ITER3	0
DISEÑO ITER3	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER3	0
MEMORIA	0
PRESENTACIÓN	0
TOTAL HORAS	**90**

Imagen 9.1 - Horas mensuales Febrero

MARZO

TAREA	TOTAL HORAS
DOP	0
ESTUDIO DEL ENTORNO + HERRAMIENTAS	24
CAPTURA REQUISITOS	18
ANÁLISIS ITER1	16
DISEÑO ITER1	24
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER1	64
ANÁLISIS ITER2	8
DISEÑO ITER2	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER2	0
ANÁLISIS ITER3	0
DISEÑO ITER3	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER3	0
MEMORIA	0
PRESENTACIÓN	0
TOTAL HORAS	**154**

Imagen 9.2 - Horas mensuales Marzo

ABRIL

TAREA	TOTAL HORAS
DOP	0
ESTUDIO DEL ENTORNO + HERRAMIENTAS	12
CAPTURA REQUISITOS	0
ANÁLISIS ITER1	0
DISEÑO ITER1	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER1	24
ANALISIS ITER2	8
DISEÑO ITER2	26
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER2	20
ANÁLISIS ITER3	0
DISEÑO ITER3	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER3	0
MEMORIA	0
PRESENTACIÓN	0
TOTAL HORAS	**90**

Imagen 9.3 - Horas mensuales Abril

En abril se puede observar que se continuó con la implementación de la iteración 1, ya que se decidió hacer unos cambios en la base de datos y consecuentemente hubo que modificar tanto la base de datos como el código para adaptarlo.

MAYO

TAREA	TOTAL HORAS
DOP	0
ESTUDIO DEL ENTORNO + HERRAMIENTAS	16
CAPTURA REQUISITOS	0
ANÁLISIS ITER1	0
DISEÑO ITER1	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER1	0
ANALISIS ITER2	0
DISEÑO ITER2	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER2	34
ANÁLISIS ITER3	16
DISEÑO ITER3	48
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER3	0
MEMORIA	0
PRESENTACIÓN	0
TOTAL HORAS	**114**

Imagen 9.4 - Horas mensuales Mayo

JUNIO

TAREA	TOTAL HORAS
DOP	0
ESTUDIO DEL ENTORNO + HERRAMIENTAS	0
CAPTURA REQUISITOS	0
ANÁLISIS ITER1	0
DISEÑO ITER1	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER1	0
ANALISIS ITER2	0
DISEÑO ITER2	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER2	16
ANÁLISIS ITER3	0
DISEÑO ITER3	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER3	98
MEMORIA	30
PRESENTACIÓN	0
TOTAL HORAS	**144**

Imagen 9.5 - Horas mensuales Junio

Junio fue un mes de bastante trabajo, en el que se tuvo que terminar la iteración 2, con todas las modificaciones que se indicaron por los usuarios, y además hacer la implementación de la iteración 3, que es la más larga y complicada de todas, puesto que el caso de uso 'Entrada de datos', es el de mayor tamaño y dificultad, debido a que para cada tipo de prueba debe generarse un formulario con distintos datos. Definir qué datos se deben pedir y en qué orden no fue nada fácil. Además, la entrada de datos es el paso más importante de toda la aplicación, ya que es donde se determina qué datos se van a guardar para cada prueba y cómo van a guardarse.

JULIO

TAREA	TOTAL HORAS
DOP	0
ESTUDIO DEL ENTORNO + HERRAMIENTAS	0
CAPTURA REQUISITOS	0
ANÁLISIS ITER1	0
DISEÑO ITER1	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER1	0
ANALISIS ITER2	0
DISEÑO ITER2	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER2	0
ANÁLISIS ITER3	0
DISEÑO ITER3	0
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER3	0
MEMORIA	40
PRESENTACIÓN	24
TOTAL HORAS	**64**

Imagen 9.6 - Horas mensuales Julio

El mes de Julio, es un mes claramente marcado por la entrega del proyecto. Durante este mes la tarea principal fue la elaboración de la memoria, ya que el plazo límite para la entrega era el 12 de julio. Además de la memoria, también se aprovechó para hacer algunos retoques a la implementación del proyecto.

En la siguiente tabla se puede ver el número total de horas empleadas para cada una de las tareas. Como podemos observar, el cómputo total de horas invertidos en el Proyecto de Fin de Carrera, asciende a 656 horas de trabajo, un número bastante próximo al inicialmente estimado, ya que el desfase ha sido de 19 horas.

TOTAL

TAREA	TOTAL HORAS
DOP	10
ESTUDIO DEL ENTORNO + HERRAMIENTAS	100
CAPTURA REQUISITOS	50
ANÁLISIS ITER1	16
DISEÑO ITER1	24
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER1	88
ANALISIS ITER2	16
DISEÑO ITER2	26
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER2	90
ANÁLISIS ITER3	16
DISEÑO ITER3	48
IMPLEMENTACIÓN + PRUEBAS + CIERRE ITER3	78
MEMORIA	70
PRESENTACIÓN	24
TOTAL HORAS	**656**

Imagen 9.7 - Horas Totales

A continuación, se muestran algunas gráficas que ilustran el tiempo empleado por cada tarea en forma global, el tiempo en porcentaje de cada tarea específica y el tiempo real dedicado a cada tarea específica comparándolo con el tiempo planificado inicialmente para cada tarea.

En la imagen 9.8 se muestra el tiempo empleado por cada tarea de forma global. En ella se puede ver que la mayor parte del tiempo ha estado dedicado a la implementación y las pruebas, con mucha diferencia sobre el resto. Otras tareas que han ocupado mucho tiempo han sido el diseño y el estudio del entorno y de las herramientas, con aproximadamente 100 horas cada una.

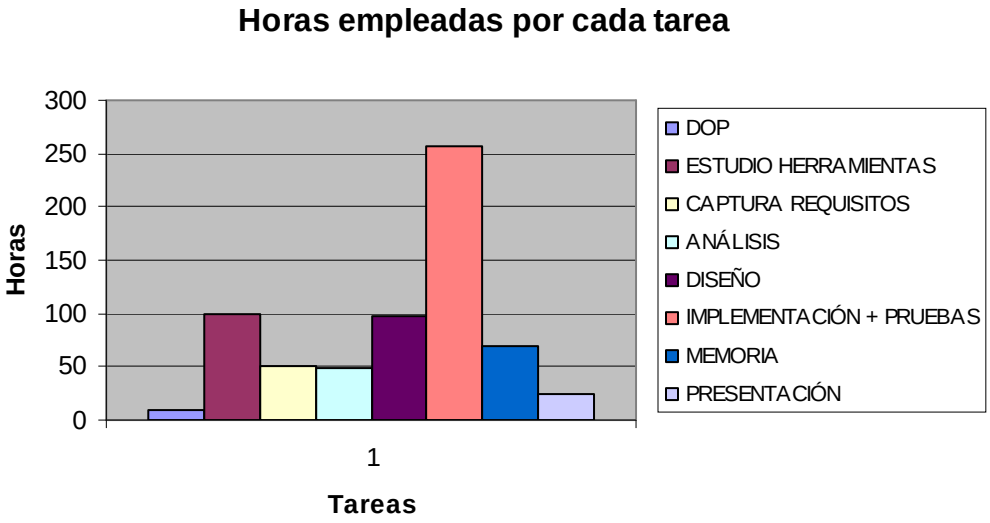


Imagen 9.8. Horas empleadas por cada tarea de forma global

En la imagen 9.9 se muestra el tiempo dedicado a cada tarea específica en forma de porcentaje sobre el total del tiempo dedicado al proyecto. Destacan sobre todo las 5 tareas que han supuesto el 65% del tiempo total, las cuales son:

- Estudio de entorno + herramientas: 15%
- Implementación y pruebas de la 1ª iteración: 13%
- Implementación y pruebas de la 2ª iteración: 14%
- Implementación y pruebas de la 3ª iteración: 12%
- Memoria: 11%

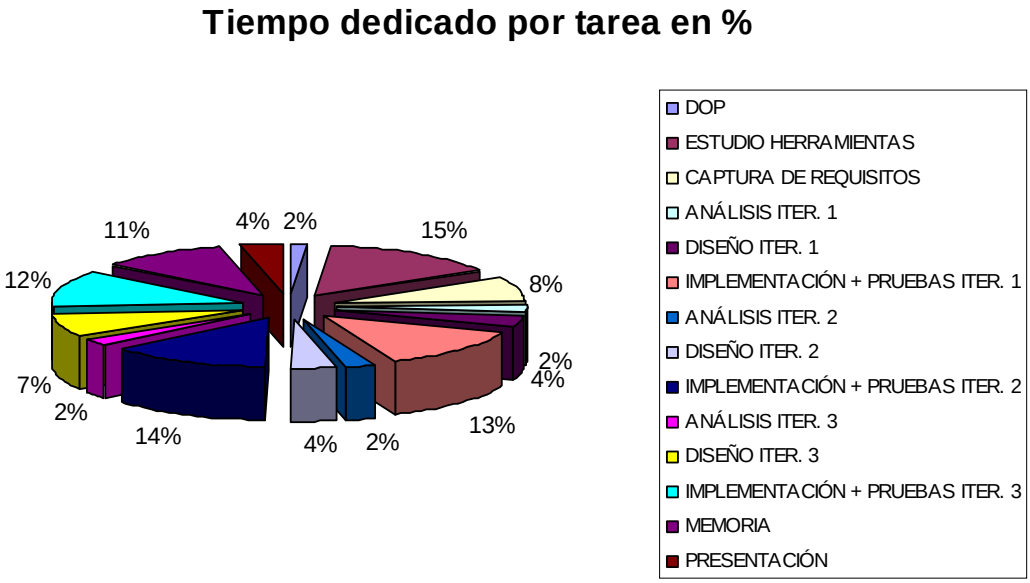


Imagen 9.9. Tiempo dedicado a cada tarea en %

La imagen 9.10, muestra la comparativa entre las horas planificadas y las reales. Se puede observar que el mayor desfase se ha producido en la estimación del tiempo de estudio del entorno y de las herramientas, el cual ha sido mucho mayor de lo previsto.

Por el contrario, el tiempo dedicado al análisis ha sido menor del previsto, aunque la estimación inicial fue bastante conservadora, por lo que tardar menos de lo planificado entra dentro de lo normal.

En cuanto a la implementación y las pruebas de las distintas iteraciones, se ha tardado más de lo previsto en la primera y la segunda iteración, mientras que en la tercera, se ha tardado menos. De todas formas, en la tercera iteración todavía habrá que hacer cambios, por lo que las horas planificadas y las reales es probable que se igualen.

Como hecho a destacar, comentar que a pesar de que no ha habido desfase con el tiempo previsto para realizar la memoria y el tiempo finalmente dedicado, sí que pensaba que la estimación de tiempo para la memoria era muy conservador, al igual que en el caso del análisis, pero finalmente se ha tardado lo previsto. Así, aunque no haya desfase entre ambos tiempos, sí que puede decirse que me ha sorprendido la cantidad de tiempo que supone redactar la memoria.

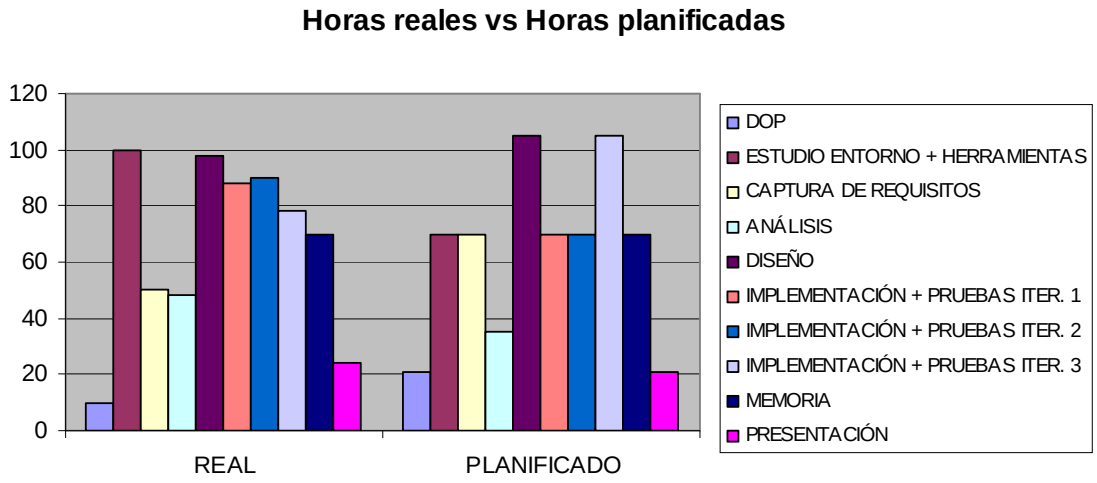
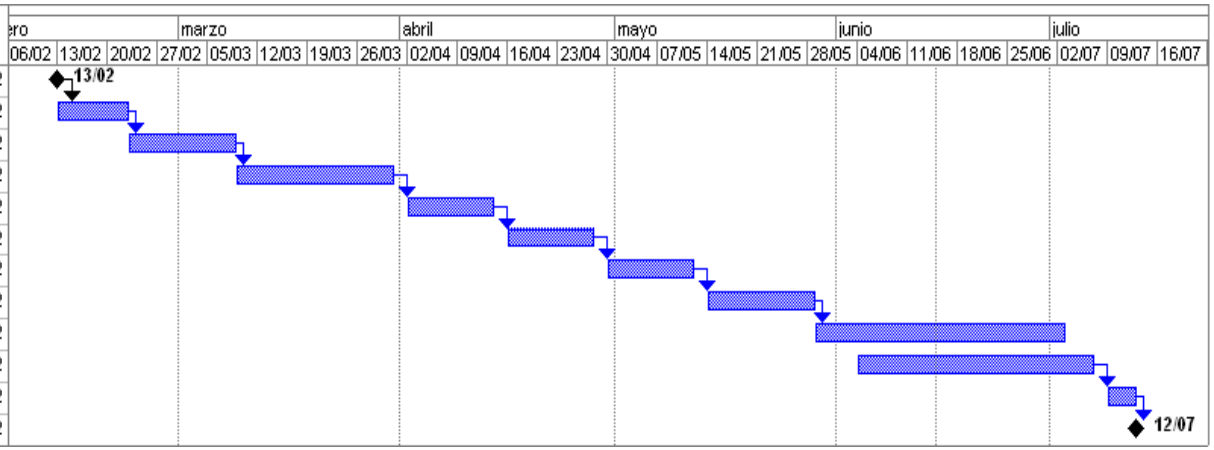


Imagen 9.10. Horas reales vs Horas planificadas

9.2. Gantt planificado vs Gantt real

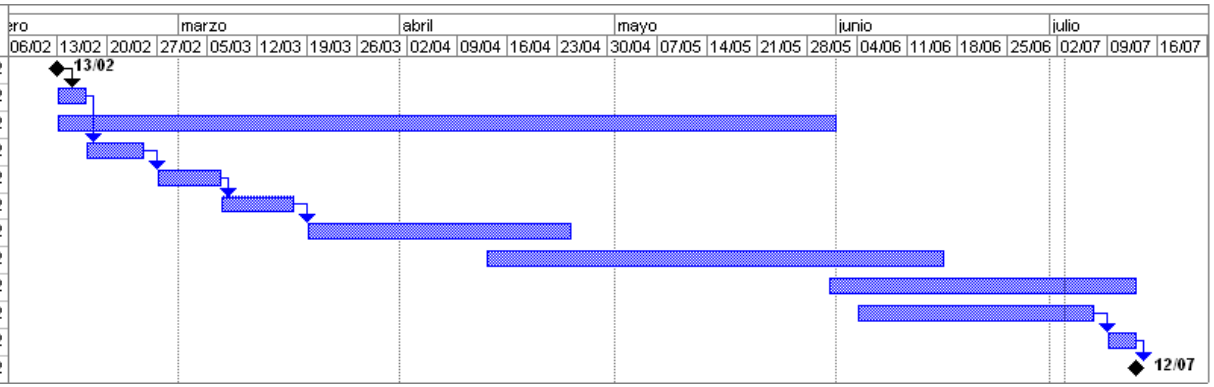
Planificación inicial:

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Comienzo del proyecto	0 días	lun 13/02/12	lun 13/02/12
2	Fase 1	8 días?	lun 13/02/12	mié 22/02/12
3	Fase 2	11 días?	jue 23/02/12	jue 08/03/12
4	Fase 3	16 días?	vie 09/03/12	vie 30/03/12
5	Fase 4	10 días?	lun 02/04/12	vie 13/04/12
6	Fase 5	10 días?	lun 16/04/12	vie 27/04/12
7	Fase 6	10 días?	lun 30/04/12	vie 11/05/12
8	Fase 7	11 días?	lun 14/05/12	lun 28/05/12
9	Fase 8	25 días?	mar 29/05/12	lun 02/07/12
10	Fase 9	25 días?	lun 04/06/12	vie 06/07/12
11	Fase 10	4 días?	lun 09/07/12	jue 12/07/12
12	Fin de proyecto	0 días	jue 12/07/12	jue 12/07/12



Planificación real:

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Comienzo del proyecto	0 días	lun 13/02/12	lun 13/02/12
2	Fase 1	4 días?	lun 13/02/12	jue 16/02/12
3	Fase 2	79 días?	lun 13/02/12	jue 31/05/12
4	Fase 3	6 días?	vie 17/02/12	vie 24/02/12
5	Fase 4	7 días?	lun 27/02/12	mar 06/03/12
6	Fase 5	8 días?	mié 07/03/12	vie 16/03/12
7	Fase 6	27 días?	lun 19/03/12	mar 24/04/12
8	Fase 7	46 días?	vie 13/04/12	vie 15/06/12
9	Fase 8	31 días?	jue 31/05/12	jue 12/07/12
10	Fase 9	25 días?	lun 04/06/12	vie 06/07/12
11	Fase 10	4 días?	lun 09/07/12	jue 12/07/12
12	Fin de proyecto	0 días	jue 12/07/12	jue 12/07/12



Una de las cosas del Gantt planificado que no han sido correctas es la duración y dependencia de las implementaciones, que se corresponden con las tareas 6, 7 y 8. En él se puede ver que había considerado que las implementaciones se cerraban totalmente, y una vez se empezaba con la siguiente iteración ya se daba por finalizada dicha tarea. La realidad es bien distinta, en ella lo que ha sucedido es que una vez está implementada y probada una iteración, surgen nuevos cambios que hacen que en realidad nunca se cierre del todo su desarrollo. En su lugar, y como se puede ver en el Gantt real, lo que sucede es que las tareas se van superponiendo. Aunque también es cierto que aunque la barra en el Gantt sea continua, no quiere decir que hayas estado dedicado a esa tarea tanto tiempo, pues por ejemplo puede suceder que entre el cierre de la iteración y los cambios que puedan surgir, pase un tiempo.

9.3. Conclusiones de la gestión

A lo largo del desarrollo de un proyecto, siempre surgen imprevistos que te obligan a modificar la dedicación planificada inicialmente. Los imprevistos son inherentes al desarrollo de software, por lo tanto, no es real pensar que no vamos a tener ninguna dificultad o contratiempo. Por eso, es muy importante hacer una planificación inicial lo más real y objetiva posible.

En el caso de los Proyectos Fin de Carrera, las estimaciones de tiempos las realiza gente inexperta, como es un universitario. Por eso, es normal que al final del proyecto se demuestre que la estimación de esfuerzo inicial sea muy corta o muy larga. En nuestro caso, nuestras estimaciones en horas, no difieren tanto la planificada de la real. Esto se debe a que a partir de las horas dedicadas en la empresa cada día y la fecha de entrega del proyecto, es fácil estimar el número total de horas que se van a dedicar. Donde sí que se han visto las divergencias es en algunas tareas concretas.

A continuación expongo los temas a los que hemos dedicado más tiempo, imprevistos que nos han surgido o en el peor de los casos, problemas que nos han obligado a retrasar o modificar la planificación establecida a medida que ha ido avanzando el proyecto.
Una lección importante, cuando te enfrentas a un trabajo de este calibre, no dependes únicamente de ti mismo. Tienes que trabajar con mucha gente (en este caso, otras 7 personas), preguntar muchas cosas, pedir información, pedir que te realicen determinadas tareas que quedan fuera de tu campo y que necesitas para proseguir con tus objetivos, que te proporcionen recursos, consejos e incluso, que te brinden

ayuda en muchas ocasiones. Todo esto y mucho más exige una planificación previa y conjunta para poder llevar adelante el proyecto y a tiempo. En mi opinión, este es el factor más imprevisible y muy a tener en cuenta a la hora de realizar una buena gestión de proyecto.

Del punto anterior se deduce la importancia de la gestión. Afortunadamente, en la empresa esto ya lo saben y llevan un control exhaustivo de la gestión de todos los proyectos. La ventaja es que queda todo perfectamente documentado, pero tiene la 'desventaja' de que he tenido que emplear mucho tiempo para redactar distintos tipos de documentos relacionados con el proyecto, como por ejemplo documentación sobre la arquitectura del programa para uso interno de la empresa. O el manual de usuario que se adjunta en el Anexo I, también se redactó para que los usuarios de la aplicación pudiesen hacer distintas pruebas.

Todo este tiempo no lo contemplé al comienzo del proyecto, y la verdad es que supone un buen número de horas.

Con respecto al desarrollo del proyecto, uno de las cosas que más tiempo y esfuerzo me ha costado, ha sido la definición de la base de datos. Hay que tener en cuenta que no conocía nada sobre generadores síncronos, y mucho menos sobre las pruebas que se les hacen para probarlos en Banco de Pruebas. Por ello, ha sido bastante complicado definir las distintas tablas y las relaciones entre ellas.

Otro aspecto que ha consumido gran cantidad de tiempo es el requerimiento de que todos los formularios deben generarse de forma dinámica con las variables almacenadas en la base de datos, de forma que para añadir una nueva variable, se añada en los lugares correspondientes de la base de datos y la aplicación siga funcionando sin modificar el código.

Haber realizado el proyecto en empresa lleva implícitas una serie de ventajas y desventajas. Hablando desde el punto de vista de la gestión, ha sido un trabajo organizado y con un seguimiento continuo. Es decir, mi objetivo era llegar a la convocatoria de Julio, pero la empresa por su parte, también ha establecido objetivos y fechas que había que cumplir.

- Ventajas: Te obliga a coger una dinámica de trabajo diario y a cumplir tu planificación.

- Desventajas: Trabajar cuando tienes una fecha encima, no encuentras las herramientas necesarias para satisfacer los objetivos de la empresa provoca momentos de agobio y estrés, sensaciones que se convierten en plena satisfacción cuando logras aquello que tanto te ha costado y has conseguido completar exactamente como querías o te pedían.

10. CONCLUSIONES

Llegados a este punto, es conocido que se han cumplido los objetivos del proyecto, que se ha finalizado con éxito y en los tiempos esperados.

El Gestor de Datos de Banco de Pruebas para Generadores Síncronos es una aplicación web capaz de gestionar la información de proyectos, generadores, pruebas de generadores y datos de las pruebas obtenidos, con fiabilidad.

La aplicación se encuentra todavía en la fase de pruebas por lo que todavía no puede decirse que sea una realidad, aunque cada vez está más cerca de ello. Cuando se haya completado esta fase, la aplicación se ubicará en la Intranet de la empresa INDAR Electric S.L. En la Extranet no es necesario que esté, ya que el Banco de Pruebas se encuentra en las propias instalaciones de la empresa.

Otro aspecto muy importante es que la aplicación es ejecutable de forma concurrente, ya que garantiza en cualquier caso la coherencia de los datos que se encuentran en la base de datos.

El preservar la confidencialidad de la empresa, me ha obligado a omitir mucho volumen de información y ha dificultado poder explicar con claridad: la estructura de las clases, los archivos que manejo, la implementación, los datos de entrada de los formularios, distintos tipos de formularios dependiendo de la cabecera o la prueba que esté seleccionada, datos que hay que almacenar, cómo se almacenan, etc…

Desde el punto de vista personal, he encontrado útil haber cursado las asignaturas de ‘Ingeniería de Software’ y ‘Herramientas Avanzadas de Desarrollo de Software’, pero al fin y al cabo no dejan de ser ejercicios académicos que pretenden simular lo que se va a encontrar el alumno al terminar sus estudios. También tenía cierta experiencia previa, pero la diferencia entre este proyecto y los demás es que hay bastantes personas implicadas, cosa que en los trabajos anteriores no sucedía.

De esta forma, en el transcurso de estos 5 meses, he tenido que organizar y asistir a multitud de reuniones redactando posteriormente sus actas correspondientes, en las que se planteaban acciones a desarrollar y los integrantes del grupo que tenían que llevarlas a cabo. Planificar y gestionar las tareas para una óptima organización, es la

acción más difícil que se presenta en este tipo de trabajos. Sin una buena planificación, es probable que alguno de los trabajadores se quede pendiente del trabajo de los demás sin poder avanzar en sus propias tareas, no se cumplan los plazos establecidos, además de por los imprevistos surgidos, por la mala planificación temporal… Nunca es plato de buen gusto asumir errores propios, aunque sean en áreas en los que todavía se es inexperto. Pero esta experiencia que he adquirido en estos 5 meses, forma parte del aprendizaje que aplicaré en futuros proyectos.

Otra de las cosas que me ha llamado la atención, es la cantidad de cambios que se dan en el software. Durante el desarrollo he visto cómo hay cambios en la base de datos, en las especificaciones, en la capa de presentación… A veces es un poco frustrante ver cómo algo que creías tener casi acabado de repente necesita una buena cantidad de trabajo de nuevo. En cualquier caso, ya me han dicho que este es el pan de cada día de los desarrolladores, así que tendré que acostumbrarme a ello.

Sin embargo, puedo considerar que la experiencia ha sido muy positiva, ya que he aprendido multitud de cosas que me servirán en el futuro, y como se ha podido ver en el tiempo dedicado al estudio del entorno y de herramientas, he adquirido un montón de conocimientos sobre un gran número de temas tales como PHP, JavaScript, AJAX, jQuery, bases de datos en general, MySQL y MS SQL Server en particular, transacciones, concurrencia, XML, DOM…

Y fuera del ámbito de la informática, la verdad es que también he aprendido muchas cosas sobre los distintos aspectos que rodean la construcción de generadores eléctricos, especialmente síncronos. Al principio todo me sonaba a chino pero a día de hoy, puedo decir que más o menos conozco un gran número de características mecánicas y eléctricas de los generadores, cosa que me alegra, ya que siempre he sido curioso por naturaleza.

11. TRABAJO FUTURO

11.1. CSS

Una de las tareas futuras más interesantes consiste en integrar CSS3 con el proyecto **GBP**.

CSS (Cascading Style Sheets) u hojas de estilo en cascada es un lenguaje utilizado para definir la presentación de un documento estructurado en formato XML o HTML. El W3C es el encargado de definir la especificación de las hojas de estilo que servirán de estándar para los navegadores.

Es un mecanismo simple que indica cómo se va a mostrar un documento en pantalla, cómo se va a imprimir, o incluso cómo se va a pronunciar la información presente en ese documento a través de un dispositivo de lectura. Las hojas de estilo proporcionan al desarrollador el control total sobre el estilo y el formato de sus documentos.

La idea subyacente en el CSS es separar el contenido de la presentación de una página. Los estilos definen la forma de mostrar los elementos HTML y XML. CSS permite al desarrollador web controlar el estilo y el formato de múltiples páginas web al mismo tiempo. De esta forma, cualquier cambio en el estilo marcado para un elemento en la hoja de estilo, afectará a todas las páginas vinculadas a esa CSS en las que aparezca ese elemento.

El uso de hojas de estilo está ampliamente extendido en la actualidad en el desarrollo web, y se pueden obtener grandes resultados haciendo que la interfaz de la aplicación tenga un aspecto profesional y personalizado.

11.2. Generación de documentación

Como se comentó con anterioridad, el proyecto puede dividirse en 3 grandes bloques, de los cuales, por limitaciones de tiempo, solamente se ha llevado a cabo el primero. Estos bloques consisten el almacenamiento y gestión de los datos, la generación de documentación en base a los datos introducidos, y la realización de consultas sobre dichos datos.

De esta manera, la generación de documentación es una tarea a realizar en el futuro, ya que hasta que no se pueda generar la documentación en base a los datos introducidos, la aplicación no será completamente funcional para su utilización.

Esta tarea consiste fundamentalmente en generar documentación tanto para uso interno como para entregar al cliente, en la cual se encuentran las características más importantes de la máquina, junto con las pruebas que se le han realizado y los resultados obtenidos para cada una de ellas.

Aunque no se ha implementado por falta de tiempo debido a que la fecha límite para la entrega del proyecto es el 12 de julio de 2012, la idea es continuar con su desarrollo a lo largo del verano.

11.3. Consultas

Las consultas conforman el último bloque de la aplicación. En él se realizarán consultas sobre todo tipo de datos, tanto de pruebas de un proyecto, como de consultas cruzadas entre datos de proyectos distintos, etc. Las posibilidades son muy numerosas, ya que tener almacenados los datos en una base de datos relacional como es MySQL permite un gran número de posibles consultas.

No obstante, a día de hoy, este bloque está sin definir, ya que no se conocen aún las consultas que se van a realizar. Este bloque se irá desarrollando cuando la aplicación sea completamente funcional en Banco de Pruebas, y se irán añadiendo distintas consultas paulatinamente.

El tamaño de este apartado será relativamente grande, ya que a medida que se utilice la aplicación irán surgiendo distintas consultas que los usuarios de la misma consideren útiles.

11.4. Soporte multilenguaje

El soporte multilenguaje puede ser otra tarea a realizar en el futuro. En la actualidad, la aplicación ofrece una interfaz en castellano. Sin embargo, muchos usuarios pueden encontrar más cómodo que dicha interfaz se encuentre en euskera o incluso en inglés.

11.5. Soporte múltiples navegadores

Otra posible tarea futura es el soporte de múltiples navegadores. Aunque en los requisitos de la aplicación sólo se solicitaba la compatibilidad con IE8, ya que es el navegador utilizado habitualmente por los usuarios de la Intranet de la empresa, parece una buena idea proporcionar compatibilidad con algunos de los navegadores más populares, como Firefox, Chrome, Opera o Safari.

La implementación de todo el proyecto se ha realizado teniendo esta posible tarea futura en mente. Por ejemplo, a la hora de crear el objeto XMLHttpRequest que se utiliza para hacer peticiones Ajax al servidor, en el código se distinguen los casos de que se tenga que crear el objeto para un navegador de la familia IE o para algún otro. El código en general y el JavaScript en particular, se ha implementado de la manera más estándar posible. En cualquier caso, lo más probable es que aún así haya que hacer algún retoque para conseguir el soporte para distintos navegadores.

12. BIBLIOGRAFÍA

- URL: http://es.wikipedia.org/
  Descripción: Wikipedia
- URL: http://stackoverflow.com/
  Descripción: Foro sobre desarrollo web
- URL: http://www.w3c.es/
  Descripción: Página oficial del World Wide Web Consortium
- URL: http://www.php.net/
  Descripción: Página oficial de PHP
- URL: http://www.w3schools.com/
  Descripción: Web sobre desarrollo web
- URL: http://www.forosdelweb.com/
  Descripción: Foro de desarrollo web
- URL: http://www.mysql.com/
  Descripción: Página oficial del SGBD MySQL
- URL: http://social.msdn.microsoft.com/Forums/eses/sqlserver/es/threads
  Descripción: Foro oficial de SQL Server
- URL: http://jquery.com/
  Descripción: Página oficial de jQuery
- URL: http://docs.jquery.com/Plugins/Validation
  Descripción: Página del plugin Validation de jQuery
- URL: http://jqueryui.com/demos/datepicker/
  Descripción: Página del plugin Validation de jQuery

13. ANEXO I. MANUAL DE USUARIO

13.1. Crear revisión de master de pruebas

La finalidad de esta acción es crear una configuración de la master de pruebas por defecto, que se cargará a la hora de definir las pruebas para cada máquina.

Para acceder a esta función, habrá que seleccionar en el apartado ‘Administración’ del menú principal de la aplicación el botón ‘Configuración’.

Imagen 13.1. Menú principal de la aplicación. Botón ‘Configuración’

De este modo, se accederá a otra pantalla en la cual se selecciona el botón ‘’Modificar lista master de pruebas’.

Imagen 13.2. Modificar lista master de pruebas

De esta forma, se accede al menú ‘Modificar lista master pruebas’ de la aplicación. En él se mostrará una lista con todas las pruebas que se pueden realizar a una máquina. Se seleccionará el tipo de las pruebas que se desee. Podrá asignarse como mínimo un tipo de prueba (Interna o Cliente) y un máximo de 2 (un tipo de prueba Interna y otro tipo de prueba Cliente). En caso de equivocación, puede deseleccionarse un tipo de prueba haciendo doble click sobre el botón incorrecto. En el caso de no seleccionar ningún tipo de prueba para una prueba determinada, no formará parte de la revisión.

Imagen 13.3. Menú ‘Modificar lista master de pruebas’

Una vez se hayan seleccionado las pruebas, se guardará la configuración mediante el botón ‘Guardar’, ubicado debajo de la tabla. La aplicación mostrará un mensaje indicando que la operación se completó con éxito, o un mensaje de error en caso contrario.

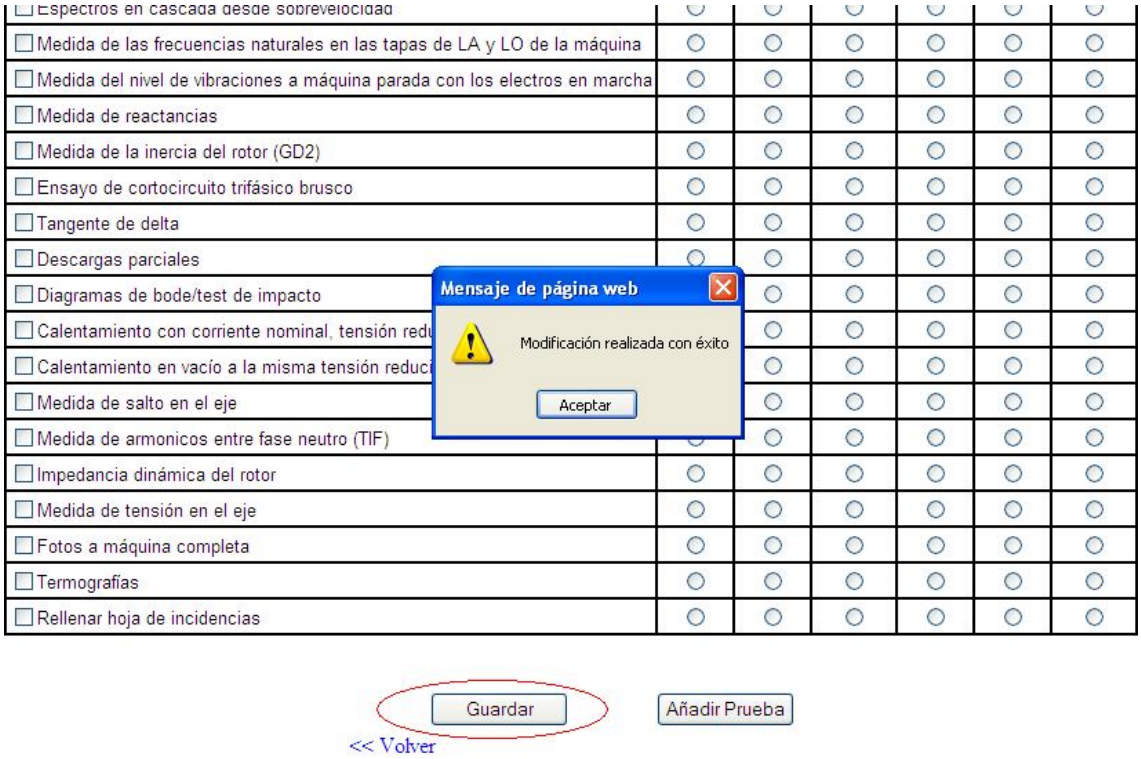


Imagen 13.4. Resultado 'Modificar lista master de pruebas'

Cuando haya 2 o más revisiones, se activará el menú desplegable 'Anteriores', que encabeza la página, el cual permitirá al usuario visualizar la configuración de las revisiones anteriores.

Imagen 13.5. Última revisión y revisiones anteriores

13.2. Crear proyecto

La finalidad de esta acción es dar de alta un proyecto en el **GBP**.

Para acceder a esta función, habrá que seleccionar en el apartado 'Gestión' del menú principal de la aplicación el botón 'Crear'.

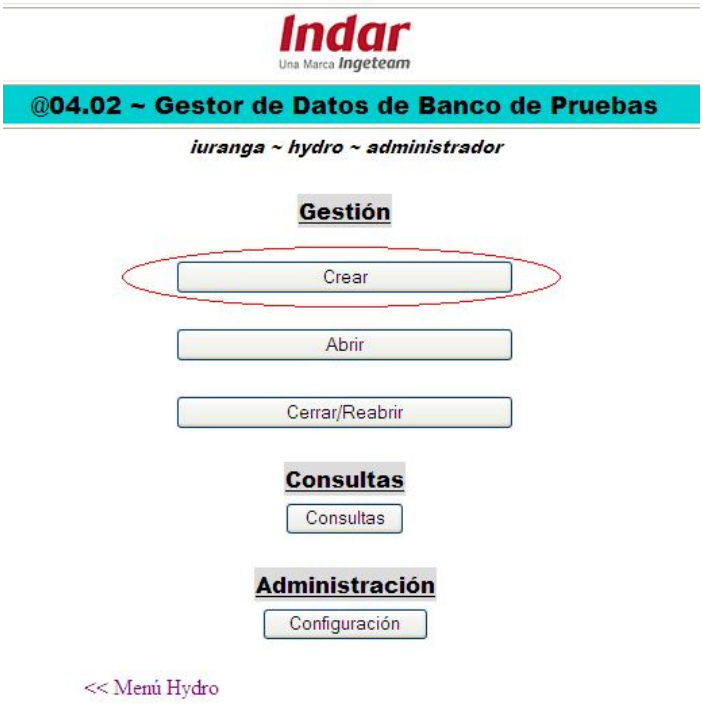


Imagen 13.6. Menú principal. Botón ‘Crear’

En la pantalla de la función se encuentra un menú desplegable llamado ‘Nº pedido’ en el cual se selecciona el número de pedido que se desea dar de alta. Para el número de pedido seleccionado, se mostrará un resumen del proyecto con la siguiente información: ‘Tipo Máquina’, ‘Nombre proyecto’, ‘Unidades’ y ‘Configuración’, además del número de la última revisión del proyecto seleccionado.

Imagen 13.7. Menú 'Crear proyecto'

Mediante los enlaces 'Definir responsables' y 'Definir implicados' se podrán asignar responsables e implicados para ese proyecto. Este paso no es obligatorio, ya que se podrán modificar a gusto del usuario posteriormente.

Para guardar los cambios y dar de alta el proyecto, se pulsa el botón 'Crear'. La aplicación mostrará un mensaje con el resultado de la operación.

Imagen 13.8. Resultado 'Crear proyecto'

13.3. Definir pruebas

La finalidad de esta acción es asignar una serie de pruebas a un número de serie de un proyecto.

Para acceder a este caso de uso, pulsar el botón ‘Abrir’ del apartado ‘Gestión’ del menú principal de la aplicación:

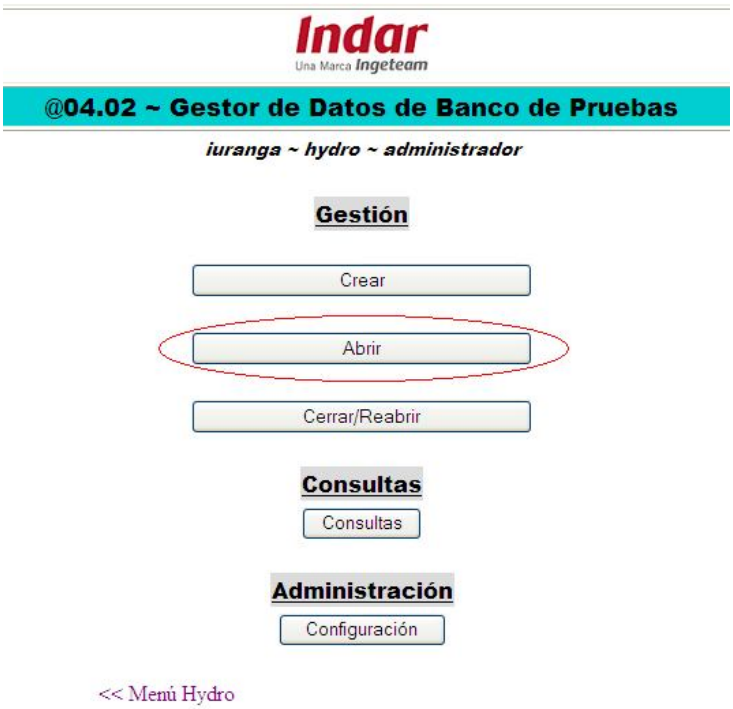


Imagen 13.9. Menú principal. Botón ‘Abrir proyecto’

De esta forma accedemos al menú correspondiente a ‘Abrir’, el cual tendrá varias acciones que se podrán realizar con los proyectos. En el desplegable ‘Nº Proyecto’ y ‘Nº Serie’, se selecciona el proyecto y el número de serie al que se desea asignar las pruebas, y se pulsa el botón ‘Def. pruebas’.

Imagen 13.10. Botón definir pruebas

De esta forma se accede al menú de la función 'Definir pruebas', en el cual se encuentran 2 tablas claramente diferenciadas.

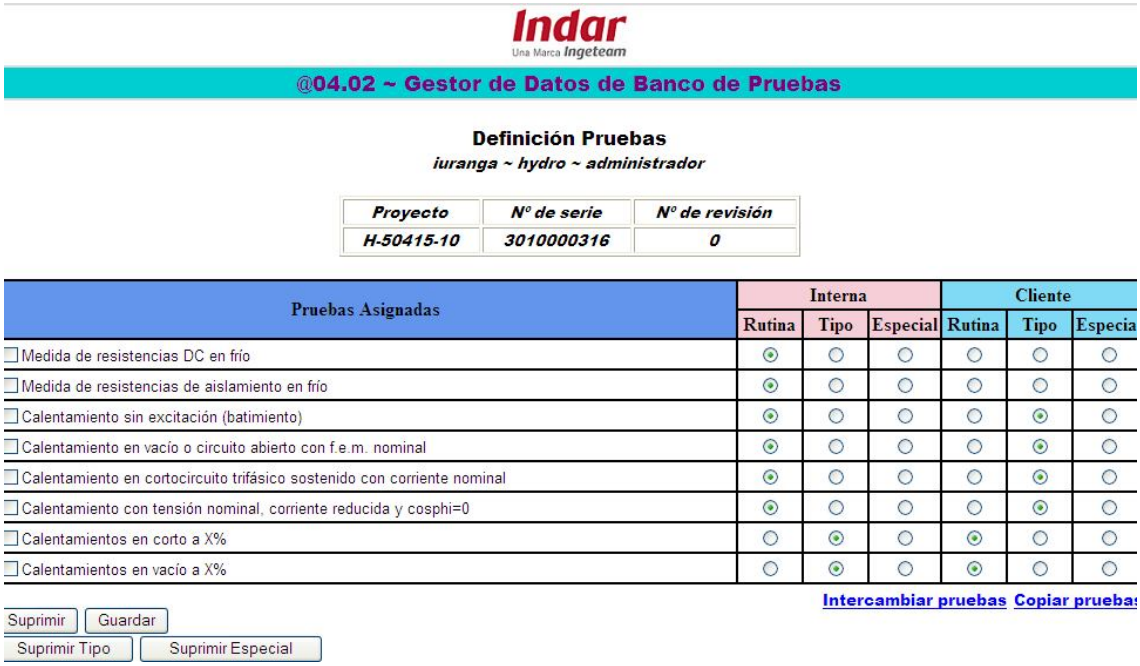


Imagen 13.11. Menú 'Definir pruebas'

En la tabla superior, llamada 'Pruebas asignadas', se muestran las pruebas que tiene asignadas un número de serie. Si el número de serie tiene asignadas pruebas, en esta tabla se mostrarán dichas pruebas. Si no tiene ninguna prueba asignada, lo indicará mediante el valor 'Nº de revisión' que encabeza la página junto al proyecto y el número de serie, y su valor será igual a 0. En este caso, para comodidad del usuario, en la

tabla ‘Pruebas asignadas’ se encontrarán seleccionadas las pruebas ‘Rutina’ de la última revisión de la master de pruebas, en el caso de tratarse de un proyecto con más de un número de serie, y en caso de ser un proyecto con un solo número de serie, se cargarán por defecto en esta tabla todas las pruebas de la última revisión de la master de pruebas.

En la tabla inferior, se muestran las pruebas posibles, es decir, el resto de pruebas que se pueden asignar a un número de serie y que no están asignadas.

Se pueden pasar pruebas de la tabla ‘Pruebas asignadas’ a ‘Pruebas posibles’ de las siguientes maneras:

1. En la tabla ‘Pruebas asignadas’, se seleccionan una o varias pruebas con el checkbox que se encuentra a la izquierda del nombre de la prueba, y cuando se han seleccionado todas las pruebas deseadas, se pasan a la tabla ‘Pruebas posibles’ pulsando el botón ‘Suprimir’.

2. Haciendo doble click sobre la prueba que se desea cambiar de la tabla ‘Pruebas asignadas’ a ‘Pruebas posibles’.

3. Si se desea cambiar de tabla todas las pruebas con el atributo ‘Tipo’, basta con pulsar el botón ‘Suprimir Tipo’.

4. Si se desea cambiar de tabla todas las pruebas con el atributo ‘Especial’, basta con pulsar el botón ‘Suprimir Especial’.

De forma análoga se pueden pasar pruebas de la tabla ‘Pruebas posibles’ a ‘Pruebas asignadas’:

1. En la tabla ‘Pruebas posibles’, se seleccionan una o varias pruebas con el checkbox que se encuentra a la izquierda del nombre de la prueba, y cuando se han seleccionado todas las pruebas deseadas, se pasan a la tabla ‘Pruebas asignadas’ pulsando el botón ‘Asignar’.

2. Haciendo doble click sobre la prueba que se desea cambiar de la tabla ‘Pruebas posibles’ a ‘Pruebas asignadas’.

3. Si se desea cambiar de tabla todas las pruebas con el atributo ‘Tipo’, basta con pulsar el botón ‘Asignar Tipo’.

4. Si se desea cambiar de tabla todas las pruebas con el atributo ‘Especial’, basta con pulsar el botón ‘Asignar Especial’.

5. Si se desea cambiar de tabla todas las pruebas con el atributo ‘Rutina’, basta con pulsar el botón ‘Asignar Rutina’.

a. Copiar pruebas

Para acceder a esta funcionalidad, se debe pinchar en el enlace que se encuentra en la parte inferior derecha de la tabla ‘Pruebas asignadas’, llamado ‘Copiar pruebas’, tal y como se indica en la siguiente imagen:

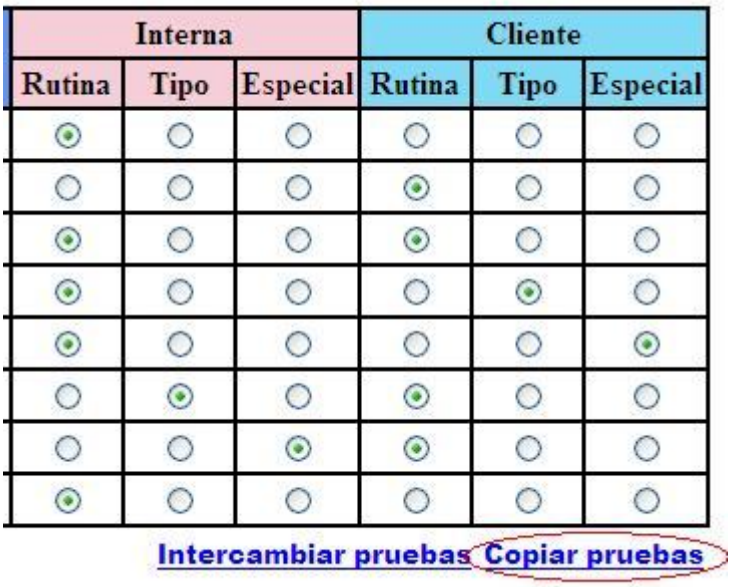


Imagen 13.12. Ubicación de ‘Copiar pruebas’

Imagen 13.13. ‘Copiar pruebas’

b. Intercambiar pruebas

Para acceder a esta funcionalidad, se debe pinchar en el enlace que se encuentra en la parte inferior derecha de la tabla 'Pruebas asignadas', llamado 'Intercambiar pruebas', tal y como se indica en la siguiente imagen:

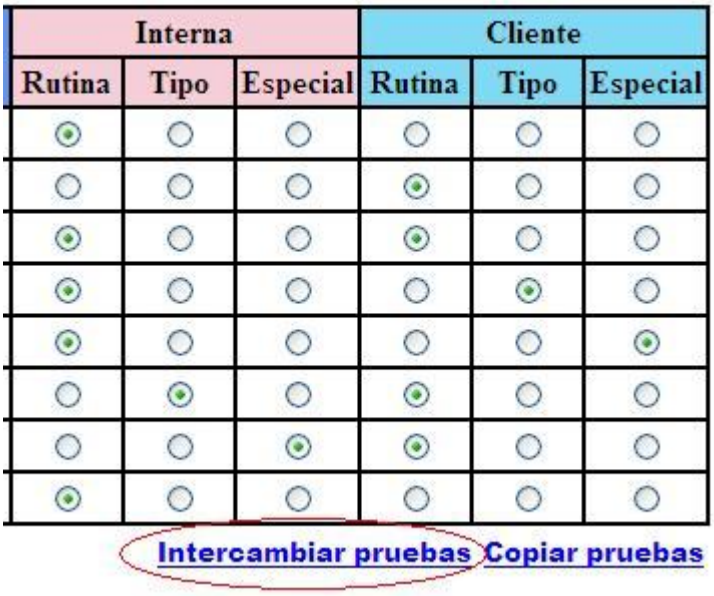


Imagen 13.14. Ubicación de 'Intercambiar pruebas'

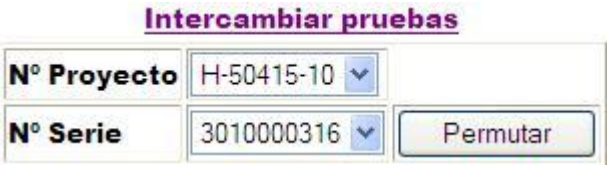


Imagen 13.15. 'Intercambiar pruebas'

Imagen 13.16. Resultado de 'Intercambiar pruebas'

13.4. Cerrar/Reabrir

Mediante esta funcionalidad de la aplicación podremos cerrar o reabrir un proyecto. Los proyectos se cierran cuando se finalizan, de esta forma la aplicación no tendrá en cuenta los pedidos 'antiguos' al mostrar los proyectos sobre los que se pueden realizar operaciones, manteniendo de esta forma una lista no demasiado extensa de opciones.

Para acceder a esta función, se debe pulsar el botón 'Cerrar/Reabrir' que se encuentra en el apartado 'Gestión' del menú principal.

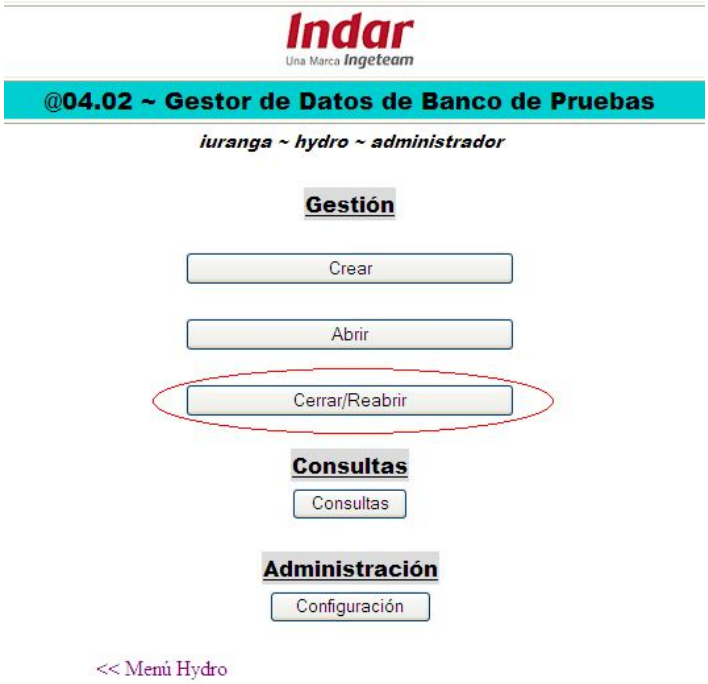


Imagen 13.17. Menú principal. Botón 'Cerrar/Reabrir proyecto'

Una vez realizado este paso, se muestran 2 secciones en pantalla: 'Cerrar proyecto' y 'Reabrir proyecto'.

Si se desea cerrar un proyecto, en la sección 'Cerrar proyecto' se selecciona una de las opciones que ofrece el menú desplegable 'Proyectos activos' y se pulsa el botón 'Cerrar'.

Imagen 13.18. Proyectos abiertos

Se puede comprobar fácilmente que el proyecto ha sido cerrado ya que ahora aparecerá en el desplegable 'Proyectos cerrados'. Por ejemplo, si cerramos el proyecto H-50415-10:

Imagen 13.19. Proyectos cerrados

De forma análoga, si lo que se desea es abrir un proyecto, en la sección 'Reabrir proyecto' se selecciona una de las opciones que ofrece el menú desplegable 'Proyectos cerrados' y se pulsa el botón 'Reabrir'.

13.5. Def. datos de cabecera

A través de esta funcionalidad se definen los datos relacionados con la cabecera de un proyecto.

Para acceder a este caso de uso, pulsar el botón ‘Abrir’ del apartado ‘Gestión’ del menú principal de la aplicación:

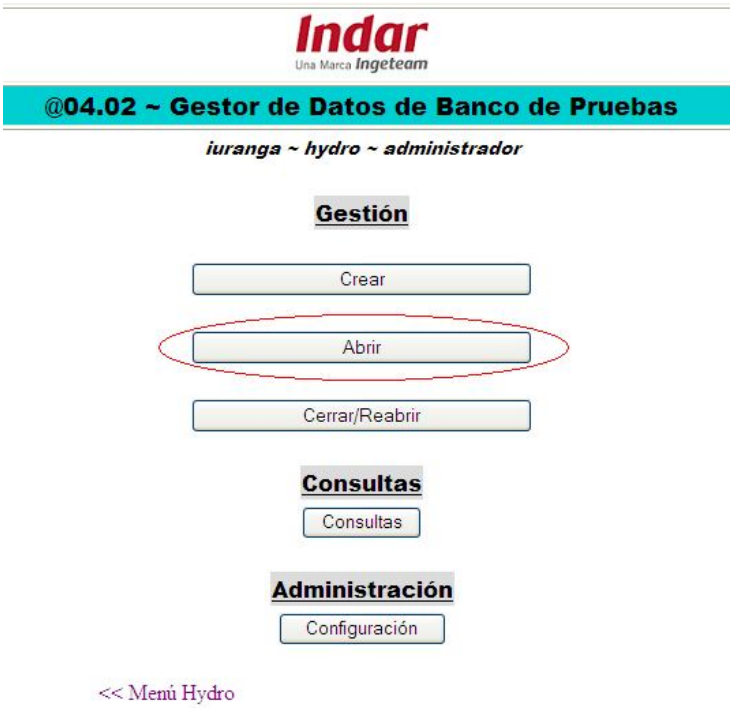


Imagen 13.20. Menú principal. Botón ‘Abrir proyecto’

De esta forma accedemos al menú correspondiente a ‘Abrir’, el cual tendrá varias acciones que se podrán realizar con los proyectos. En el desplegable ‘Nº Proyecto’ se selecciona el proyecto, y se pulsa el botón ‘Def. datos cabecera’.

Imagen 13.21. Botón 'Definir datos de cabecera'

En este caso no es necesario seleccionar un número de serie para poder acceder a la función 'Definir datos de cabecera', ya que esta información es relativa a todo el proyecto.

Una vez se pulsa el botón correspondiente, se accede al menú principal del caso de uso, que consta de un desplegable que nos permitirá seleccionar un tipo de cabecera, de entre los 6 disponibles, a saber: 'General', 'Apoyos', 'Electros', 'Refrigeración', 'Excitación' y 'Varios'. El apartado 'Electros' sólo será posible seleccionarlo cuando el proyecto tenga electroventiladores.

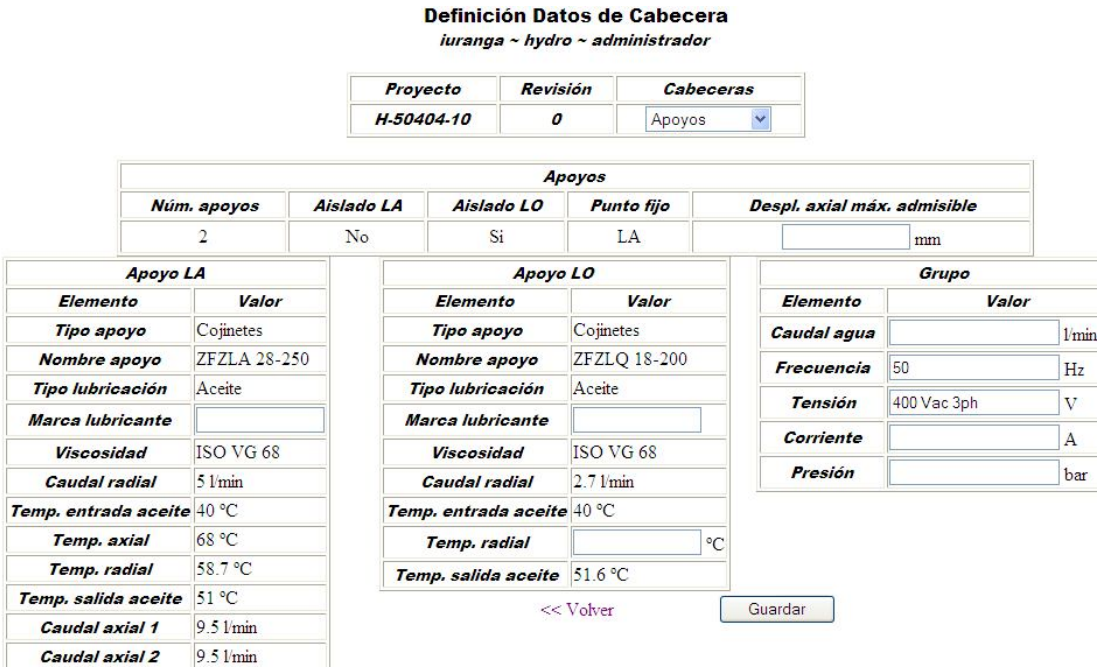


Imagen 13.22. Menú 'Apoyos'

Se podrán introducir o editar los datos que se encuentren en campos de texto. El resto de campos no será posible modificarlos. Se introducirán los datos que se deseen, en el ejemplo se introduce la marca lubricante del apoyo LA[12] y del apoyo LO[13], y la corriente y tensión del grupo de lubricación. Se pulsa 'Guardar' y el sistema nos devuelve un mensaje con el resultado de la operación.

Imagen 13.23. Resultado 'Guardar cabeceras'

Se puede comprobar el resultado accediendo nuevamente al menú correspondiente de cabeceras, en este caso el de apoyos. De esta forma se ven los datos introducidos anteriormente.

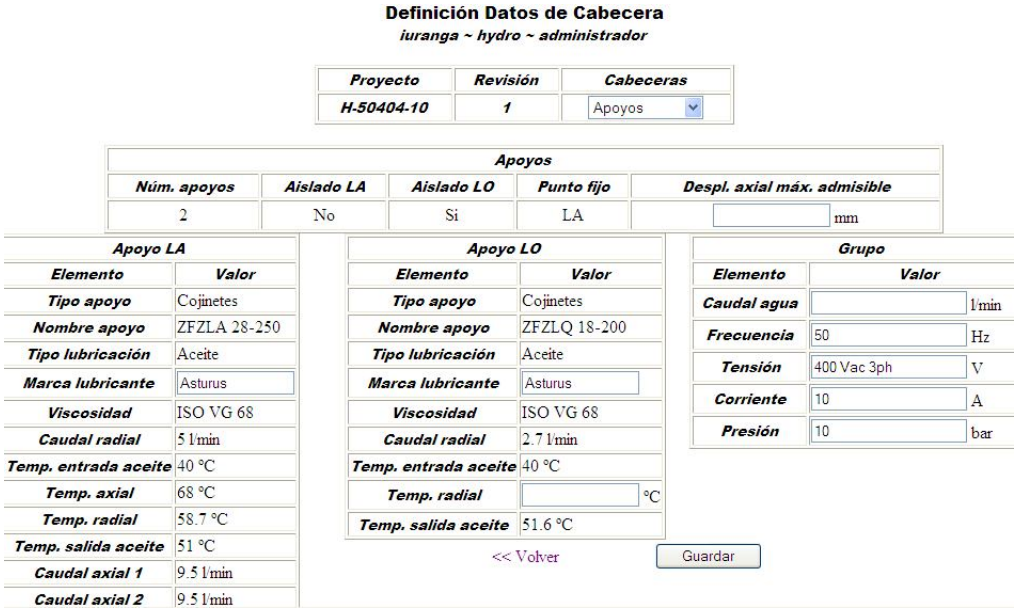


Imagen 13.24. Menú 'Apoyos' mostrando los datos guardados

[12] Lado de activación
[13] Lado opuesto

Se comprueba fácilmente viendo que el apartado ‘Revisión’ que encabeza la página ha pasado de ‘0’ a ‘1’, y también se ven los 4 valores que se han introducido, ‘Asturus’ como marca lubricante, y ‘10’ como valores de corriente y presión del grupo de lubricación.

13.6. Entrada de datos

A través de esta funcionalidad se definen los datos relacionados con la cabecera de un proyecto.

Para acceder a este caso de uso, pulsar el botón ‘Abrir’ del apartado ‘Gestión’ del menú principal de la aplicación:

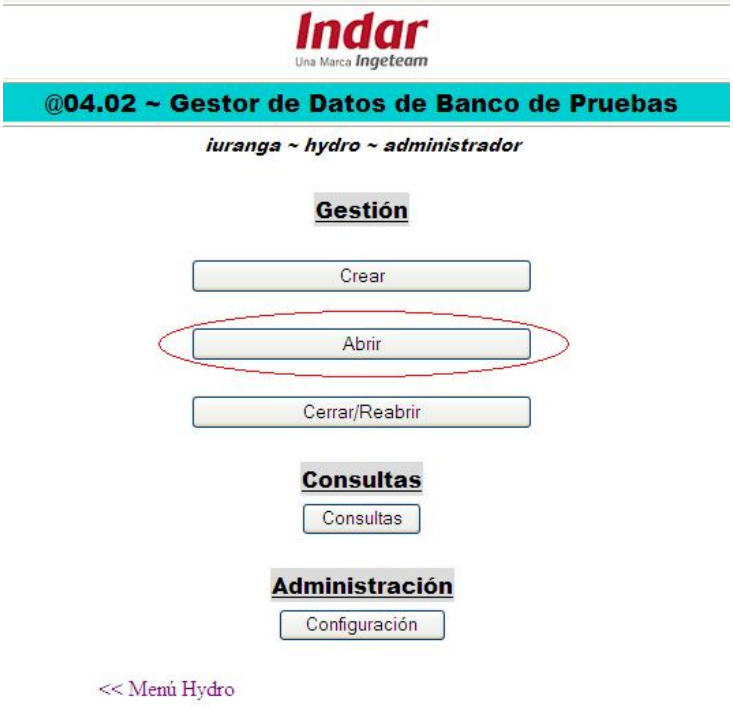


Imagen 13.25. Menú principal. Botón ‘Abrir proyecto’

De esta forma accedemos al menú correspondiente a ‘Abrir’, el cual tendrá varias acciones que se podrán realizar con los proyectos. En el desplegable ‘Nº Proyecto’ y ‘Nº Serie’, se selecciona el proyecto y el número de serie del que se desean introducir datos, y se pulsa el botón ‘Entrada datos’.

Imagen 13.26. Botón 'Entrada datos'

A continuación se seleccionará la prueba de la que se quiere hacer la entrada de datos. Además, en el desplegable 'Entrada datos' se selecciona si se quiere hacer una nueva entrada o se desea editar una de las anteriores. Al ser la primera vez, solamente tendremos la opción de seleccionar 'Nueva'. Como son varios formularios, solamente se mostrará el primer paso.

Imagen 13.27. Primer formulario de 'Entrada datos'

Cuando se completen todos los pasos, se guardan los datos en la base de datos mediante el botón 'Guardar'. El sistema mostrará el resultado de la operación.

Imagen 13.28. Resultado 'Entrada de datos'

Se puede comprobar que efectivamente se han introducido los datos seleccionando la misma prueba de nuevo. Ahora el desplegable 'Entrada datos' también dará la opción de editar la entrada anterior. Seleccionamos esa opción y se comprueban los datos.

13.7. Modificar responsables e implicados

Mediante esta funcionalidad de la aplicación podremos modificar los responsables e implicados asignados a un proyecto. Para acceder a ella seleccionamos el botón 'Abrir' en el menú principal de la aplicación:

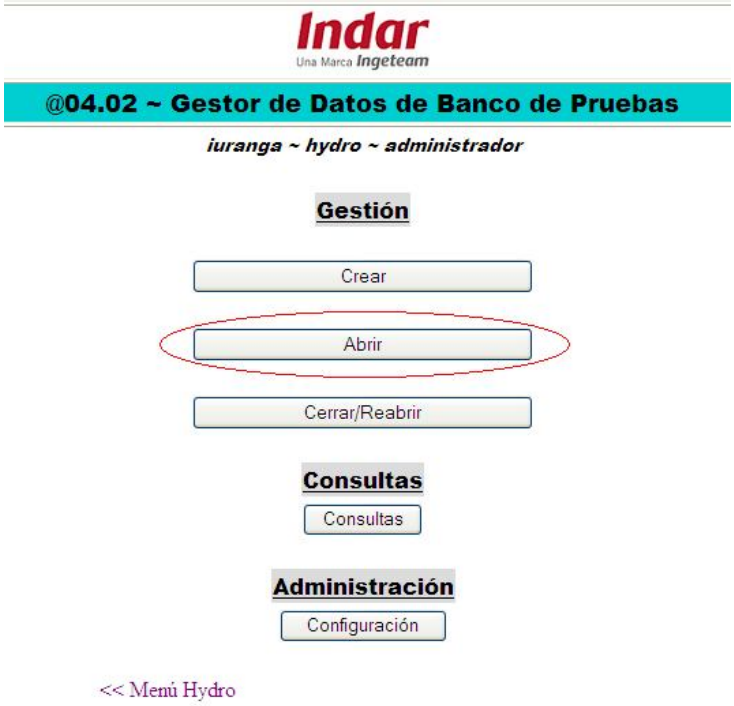


Imagen 13.29. Menú principal. Botón 'Abrir proyecto'

A continuación, seleccionamos en el desplegable 'Nº proyecto' el número de proyecto del cual queremos modificar los responsables e implicados. Al seleccionarlo se activará el botón correspondiente. Para acceder al menú del caso de uso, se pulsa el botón indicado.

Imagen 13.30. Botón 'Modificar responsables e implicados'

Dentro del menú, se seleccionan los responsables e implicados que se desea que tenga el proyecto. Esto se muestra en la siguiente imagen:

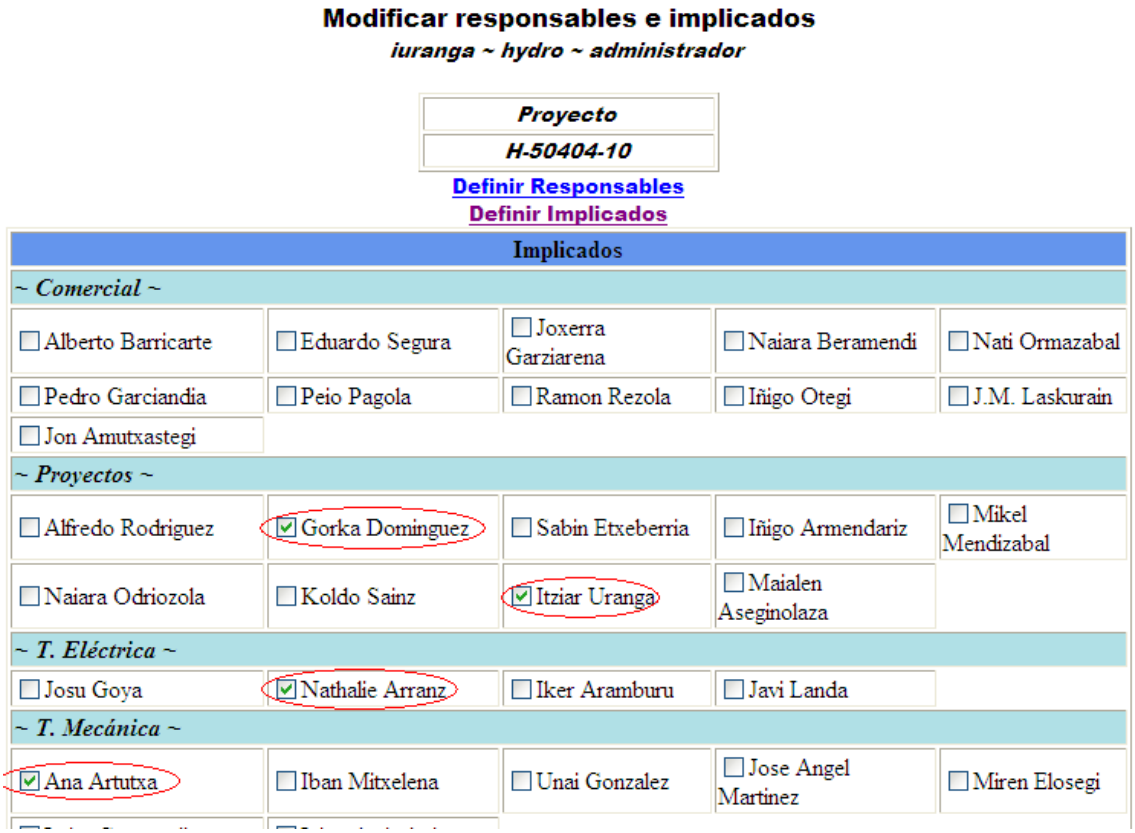


Imagen 13.31. Menú 'Modificar responsables e implicados'

Cuando se hayan seleccionado todos los responsables e implicados deseados, se guardan los datos mediante el botón 'Modificar'. El sistema devolverá un mensaje con el resultado de la aplicación.

Imagen 13.32. Resultado 'Modificar responsables e implicados'