UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Logbook: Herramienta de Supervisión de las Actividades de Pilotos de Aeronaves.

Trabajo de Fin de Carrera
Ingeniería Técnica en Informática de Gestión

Autora: Luisa Rodríguez Algar
Tutor: José Simó Ten
Septiembre 2015

Agradecimientos:

Me gustaría aprovechar este espacio para expresar mi sincero agradecimiento a todos los que han ayudado a que este trabajo llegue a su fin.

A José Simó, que me brindó su ayuda durante un vuelo y, a pesar de los años que he tardado en decidirme a aceptarla, ha confiado en mi y en este proyecto.

A mis amigos y familiares, que siempre me han apoyado: a mi madre porque, sin su insistencia, hubiera incumplido mi promesa y este proyecto no sería una realidad. A Carlos, que siempre está ahí para mi y a Dani, compañero piloto-informático, por sus buenos consejos y sus ánimos.

Muchas gracias a todos.

Resumen.

Este proyecto tiene como objetivo la creación de una aplicación web que permita la supervisión de las actividades de los pilotos de aeronaves, concretamente propone una solución informática multiplataforma que suponga una ayuda para la anotación de los vuelos y las sesiones de simulador en el cuarderno de bitácora, recordatorios para mantener en vigor las licencias y certificados médicos y una herramienta de búsqueda para generar informes con diferentes tipos de parámaetros. Se utiliza una métodología en tres capas con una base de datos MySQL, una aplicación en Java y Java Servlet y una interfaz con HTML5, CSS3 y JavaScript.

Palabras clave: cuaderno, bitácora, logbook, piloto, aeronave, MySQL, HTML, CSS, JavaScript, boceto, Servlet, Java.

Summary.

The objective of this project is to develop a web application which permits to survey the duties of aircraft pilots, more precisely what has been proposed is a multi-platform software solution which eases the task of filling up the logbook with the flights and simulator sessions, reminders to help to maintain medical certificates, licenses and type ratings up to date and a search tool able to generate reports with different types of parameters. A three layer methodology is used, using MySQL for the database, Java and Java Servlet for application logic and HTML5, CSS3 and JavaScript for user interface.

Keywords: logbook, pilot, aircraft, MySQL, HTML, CSS, JavaScript, Mockup, Servlet, Java.

1. Introducción.

1.1. Descripción general.

El logbook o cuaderno de bitácora es un documento que deben de tener en posesión y actualizado todos los pilotos de aeronaves. En él se reflejan los datos significativos de los vuelos que este piloto ha realizado: la fecha, los aeropuertos a los que voló, la matrícula del avión, tipo de vuelo, etc.

El logbook es mucho más que una simple anotación de actividades, puesto que se trata de una declaración jurada realizada por el piloto y prueba legal de su experiencia. Es por esto que se debe de tener siempre actualizado y sin errores.

Así pues, es una importante tarea para un piloto tener el logbook correcto y actualizado en todo momento. Pero es una tarea tediosa: hay que tomar nota de todos los parámetros a completar, ordenar todos los vuelos cronológicamente, comprobar todas las sumas de los tiempos para evitar cometer errores, etc.

1.2. Objetivo del proyecto.

Para dar un solución al problema propuesto en el apartado anterior, se propone una aplicación informática que facilite tan importante tarea y permita tener este documento actualizado en tiempo real.

Se desea una aplicación multiplataforma, capaz de almacenar los datos de los vuelos de diferentes usuarios y presentarla de la misma forma en la que esta aparecería en una página del cuaderno de bitácora y formularios que permitan todo tipo de búsquedas: vuelos en una fecha, en un aeropuerto, con un tipo concreto de avión, etc. Se puede añadir, además, información sobre las licencias y reconocimientos médicos del piloto, de forma que la aplicación envíe recordatorios cuando estén próximos a la fecha de cumplimiento.

1.3. Instrucciones de uso del Cuaderno de Bitácora.

Se incluyen a continuación las instrucciones de uso, tal y como aparecen en el documento original, traducidas del inglés. En este caso utilizaremos un logbook de la marca Jeppesen, ampliamente conocido por la mayoría de pilotos, además de aceptado por la Agencia Estatal de Seguridad Aerea, autoridad española, dependiente del Ministerio de Fomento, que regula las actividades aéreas.

1. EU-FCL[1] 1.080 y EU-FCL 2.080 requiere a los poseedores de una licencia de tripulación de vuelo anotar los detalles de todos sus vuelos operados, en un formato aceptable por la autoridad de aviación nacional responsable de la expedición de la licencia o de la habilitación. Este logbook permite a los pilotos poseedores de una licencia anotar su experiencia de vuelo de forma que facilite este proceso a la vez que proporciona un archivo permanente de sus vuelos. A los pilotos que vuelen regularmente aeroplanos y helicópteros u otros tipos de aeronaves les recomendamos que mantengan logbooks separados para cada tipo. Jeppesen proporciona columnas adicionales para los pilotos que deseen anotar los vuelos con otros tipos de aeronave en el mismo logbook.

2. Las entradas en el logbook deben de ser hechas lo antes posible después de cada vuelo realizado. Todas las entradas en el libro tendrán que ser hechas con tinta o lápiz permanente.

3. Las particularidades de cada vuelo, durante el curso del cual, el poseedor de una licencia de tripulante de vuelo actúe como un miembro de la tripulación que opera el avión, se tienen que anotar en las columnas apropiadas usando una línea para cada vuelo. Siempre que la aeronave lleve a cabo un número de vuelos en el mismo día, volviendo en cada ocasión al mismo aeropuerto de salida y en un intervalo entre vuelos consecutivos que no exceda los treinta minutos, esta serie de vuelos se podrá anotar en un sola línea.

4. El tiempo de vuelo se anota desde el momento en el que el avión empieza a moverse por sus propios medios con el propósito de despegar hasta el momento en el que el avión se para finalmente después del aterrizaje (ver EU-FCL 1.001)

[1] European Union-Flight Crew Licensing

5. Cuando un avión lleva dos o más pilotos como miento de una tripulación, uno de ellos debe, antes de que el vuelo comience, ser designado por el operador como el "comandante" de la aeronave, de acuerdo a EU-OPS[2], el cual puede delegar la realización del vuelo a otro piloto competente. Todo el tiempo de vuelo llevado a cabo como "comandante" debe de ser anotado en el logbook como "piloto-al-mando". Un piloto volando como "piloto-al-mando bajo supervisión" o "alumno piloto-al-mando" debe de registrar los tiempos de vuelo como "piloto-al-mando" pero todas estas entradas tendrán que ser certificadas por el comandante o el instructor de vuelo en la columna "Comentarios" del logbook.

6. **Instrucciones para el registro de los tiempos de vuelo:**

- **Columna 1:** Anotar la fecha (dd/mm/yy) en la que el vuelo comienza.
- **Columna 2/3:** Anotar el aeropuerto de salida y de llegada, con su código completo de tres o cuatro letras, internacionalmente reconocido. Todos las horas tienen que ser en UTC.
- **Columna 5:** Indicar si la operación fue mono o multipiloto, y para monotripulada, si se llevo a cabo en un monomotor o multimotor.
- **Columna 6:** El tiempo total de vuelo debe de registrarse en horas y minutos o en notación decimal, según se desee.
- **Columna 7:** Anotar el nombre del "piloto-al-mando" o SELF según corresponda.
- Columna 8: Indicar el número de despegues y aterrizajes como piloto en vuelo de día y/o de noche.
- **Columna 9:** Registrar tiempo de vuelo nocturno o bajo reglas de vuelo instrumental, si se aplica.
- **Columna 10:** Tiempo de función del piloto:
    - Registrar el tiempo de vuelo como "piloto-al-mando" (PIC[3]), "alumno piloto al mando" (SPIC[4]) y "piloto-al-mando bajo supervisión" (PICUS[5]) como PIC.
    - Todo tiempo registrado como SPIC o PICUS tiene que ser firmado por el comandante de la aeronave o el instructor de vuelo en la columna "Comentarios" (Columna 12).

[2] European Union-Operations

[3] Pilot-in-command

[4] Student Pilot-in-command

[5] Pilot-in-command under supervision

    - El tiempo como instructor de vuelo debe de registrarse apropiadamente y además, introducirse como PIC.
- **Columna 11:** Simulador de vuelo (FS o FFS[6]) o entrenador de procedimientos de navegación de vuelo (FNPT[7]) :
    - Para FS registra el tipo de aeronave y el número de calificación del dispositivo. Para otro tipo de entrenadores de vuelo inserta FNTP I o FNPT II según corresponda.
    - El tiempo total de la sesión incluye todos los ejercicios llevados a cabo en el dispositivo, incluyendo los chequeos pre y postvuelo.
    - Registra el tipo de ejercicio realizado en Comentarios (Columna 12). Ejemplo: Chequeo de pericia del operador (OPC[8]), revalidación, etc.
- **Columna 12:** La columna de "Comentarios" puede ser utilizada para anotar detalles del vuelo a discreción del propietario. Las siguientes entradas, sin embargo, deben de realizarse obligatoriamente:
    - Tiempo de vuelo en reglas instrumentales como parte de un entrenamiento para la obtención de una licencia o habilitación.
    - Detalles de un chequeo de habilidad o pericia.
    - Firma del PIC si el piloto está anotando el vuelo como SPIC o PICUS.
    - Firma del instructor si el vuelo es parte de una revalidación de clase de monomotor o de motovelero.

7. Cuando cada página sea completada, los tiempos de vuelo acumulados deben ser cumplimentados en la columna apropiada y firmada por el piloto en la columna de comentarios.

Dada la gran cantidad de información almacenada en cada página del libro, formada por doce columnas, algunas de ellas divididas en varios apartados, y dieciocho líneas, cada registro o anotación se representa a doble página.

[6] Flight Simulator o Full Flight Simulator

[7] Flight Navigation Procedures Trainer

[8] Operator Proficiency Check

Adjuntamos a continuación dos capturas que representan un ejemplo de cómo rellenar el cuaderno en los diferentes casos explicados anteriormente, extraída del documento originar de Jeppesen:

1	2		3		4		5						6		7	8			
DATE (dd/mm/yy)	DEPARTURE		ARRIVAL		AIRCRAFT		SINGLE PILOT TIME				MULTI-PILOT TIME		TOTAL TIME OF FLIGHT		NAME PIC	TAKEOFFS		LANDINGS	
	PLACE	TIME	PLACE	TIME	MAKE, MODEL, VARIANT	REGISTRATION	SE	ME								DAY	NIGHT	DAY	NIGHT
14/11/98	LFAC	1025	EGBJ	1240	PA34-250	G-SENE		✓					2	15	SELF	1			
15/11/98	EGBJ	1810	EGBJ	1930	C152	G-NONE	✓						1	20	SELF		2		
22/11/98	LGW	1645	LAX	0225	B747-400	G-ABCD					9	40	9	40	SPEAKIN		1		

Figura 1: Ejemplo de anotaciones en el logbook. Página 1.

9				10								11				12
OPERATIONAL CONDITION TIME				PILOT FUNCTION TIME								SYNTHETIC TRAINING DEVICES SESSION				REMARKS AND ENDORSEMENTS
NIGHT		IFR		PILOT-IN-COMMAND		CO-PILOT		DUAL		INSTRUCTOR		DATE (dd/mm/yy)	TYPE	TOTAL TIME OF SESSION		
		2	15	2	15											
1	20			1	20					1	20					Night rating training (A L Pilot)
												20/11/98	B747-400 (Q1234)	4	10	Revalidation Prof Check
8	10	9	40	9	40											PIC(US) C Speaker

Figura 2: Ejemplo de anotaciones en el logbook. Página 2.

2. Dependencias.

2.1. Descripción general de la aplicación.

Se plantea una aplicación web que pueda ser utilizada en cualquier navegador, independientemente del tipo de dispositivo: móviles, tabletas y ordenadores. Es especialmente interesante la opción del móvil, puesto que ayudaría al piloto a poder actualizar toda la información del día al terminar su jornada laboral.

Para estructurar la aplicación utilizaremos la programación por capas, una arquitectura cliente-servidor (las capas inferiores proporcionan servicios, mientras que las capas superiores son los usuarios de éstos) en el que el objetivo primordial es la separación de la las tres capas que la conforman.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido.

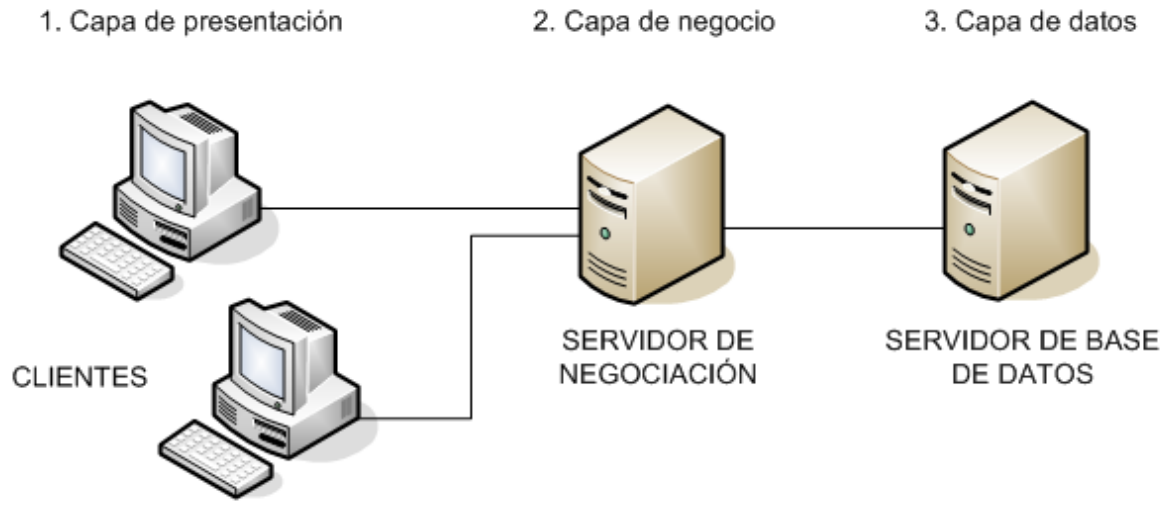


Figura 3: Arquitectura "tres capas".

En el caso concreto de las aplicaciones web, en la **capa de presentación**, que implementa la interacción con los usuarios a través de una representación visual, encontraremos el navegador web del cliente que interpretará el código HTML y JavaScript, además de mostrar una interfaz agradable al usuario gracias a CSS.

La **capa de negocio, de dominio o de aplicación**, que implementa completamente el comportamiento de las clases del dominio, especificadas en la fase de modelado conceptual, estará soportada por el contenedor de servlets Tomcat. Estos servlets, programados en Java, proporcionarán servicios web e interactuarán con los objetos, implementados como clases Java.

Para la base de datos, es decir, la **capa de datos o persistencia**, que proporciona una serie de servicios que permiten a los objetos del dominio interactuar con su repositorio permanente asociado, usaremos un servidor MySQL, que nos permitirá crear la base de datos y acceder a ella.

2.2. Dependencias Software.

Tal y como se ha comentado en el párrafo anterior, necesitaremos hacer uso de una serie de programas, tecnologías, lenguajes y software en general, tanto para empezar a trabajar como para poner en marcha nuestra aplicación. Distinguiremos estas dos fases como desarrollo y despliegue y comentamos a continuación, de forma más detallada, donde encontrar estas herramientas, cómo instalarlas y configurarlas. El proyecto se desarrollará y desplegará íntegramente usando un MacBook Air con el sistema operativo OS X Yosemite, versión 10.10.5.

2.2.1. Desarrollo de la aplicación.

2.2.1.1. MySQL.

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario, bajo GNU GPL[9] para cualquier uso compatible con esta licencia, desarrollado por Sun Microsystems, perteneciente a Oracle Corporation.

Necesitaremos descargarnos el fichero `mysql-5.6.26-osx10.9-x86_64.dmg` en el enlace `Mac OS X 10.9 (x86, 64-bit) DMG Archive`, que corresponde a la aplicación MySQL Community Server 5.6.2 de la página oficial de MySQL[10]. Procederemos a la instalación de paquete y podremos arrancar o parar nuestro servidor desde el panel de Preferencias del Sistema > MySQL. Podremos conectarnos a nuestra base de datos con `127.0.0.1` como Host y `root` como Usuario, sin contraseña.

[9] GNU General Public License

[10] http://dev.mysql.com/downloads/mysql/

MySQL no sólo será usado durante la fase de desarrollo, si no que permanecerá como parte fundamental de nuestra aplicación, formando la capa de persistencia, durante el despliegue de la misma.

2.2.1.2. Servlet Java.

Un servlet es una clase del lenguaje de programación Java que se ejecuta en el servidor. Son utilizados para ampliar las capacidades del mismo y presentan como ventajas sobre los CGI[11] la eficiencia, la potencia expresiva y la portabilidad. Aunque los servlets pueden responder a cualquier tipo de solicitudes, éstos son utilizados comúnmente para extender las aplicaciones alojadas por servidores web, generando páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

Para el desarrollo de la aplicación descargaremos Eclipse IDE for Java EE Developers Versión 4.4, proyecto Luna. Eclipse está compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar entornos de desarrollo integrados. Este paquete de Eclipse posee herramientas específicas para desarrolladores que deseen crear Java EE y aplicaciones Web, incluyendo Java IDE, herramientas para Java EE, JPA[12], JSF[13], Mylyn[14], EGit[15] y otros. De la página de Eclipse[16] seleccionaremos la versión Mac OS X (Cocoa) 64-bit, descargándonos el fichero `eclipse-jee-luna-R-macosx-cocoa-x86_64.tar.gz`.

La instalación es sencilla, siguiendo una serie de pasos con un asistente.

2.2.1.3. JavaScript.

JavaScript es un lenguaje de programación interpretado, que se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

---

[11] Common Gateway Interface

[12] Java Persistence API

[13] Java Server Faces

[14] Extensión de Eclipse para la gestión de proyectos y tareas

[15] Control de sistemas distribuidos para proyectos

[16] http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/luna/R/eclipse-jee-luna-R-macosx-cocoa-x86_64.tar.gz

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

Durante el desarrollo de este proyecto usaremos los navegadores Safari Versión 8.0.8, Firefox Versión 40.0.3 y Google Chrome Versión 44.0.2403.157 (64-bit). Todos ellos soportan JavaScript de forma nativa y podemos activarlo o desactivarlo de la siguiente forma en Safari > Preferencias… > Seguridad > Permitir JavaScript. En Firefox y Chrome no parece que haya opción de desactivar JavaScript, por lo que lo permitirán siempre.

2.2.1.4. HTML 5

HTML5[17] es la quinta revisión importante del lenguaje básico de la web. En esta nueva versión se especifican dos variantes de sintaxis: una "clásica", HTML5 (text/html) y una variante, XHTML5, que deberá servirse con sintaxis XML (application/xhtml+xml).

Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014. e incluye nuevos elementos semánticos, un mayor controle los atributos en los formularios, así como nuevos elementos gráficos y multimedia. El desarrollo de este lenguaje de marcado es regulado por el Consorcio W3C.

2.2.1.5. CSS 3

Hojas de estilo en cascada o CSS[18], es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML. El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.
La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

A diferencia de CSS2, que fue una única especificación que definía varias funcionalidades, CSS3 está dividida en varios documentos separados, llamados "módulos". Cada módulo añade nuevas funcionalidades a las definidas en CSS2, de manera que se preservan las anteriores para mantener la compatibilidad.

---

[17] HyperText Markup Language, versión 5

[18] Cascading Style Sheets

2.2.2. Despliegue de la aplicación.

2.2.2.1. Tomcat.

Apache Tomcat, o simplemente Tomcat, funciona como un contenedor web con soporte de servlets y JSPs[19] desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JSP de Oracle Corporation.

Tomcat no es un servidor de aplicaciones, pero sí puede funcionar como servidor web por sí mismo y hoy en día es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que fue desarrollado en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Para instalar Java en nuestro sistema abrimos la aplicación Terminal y escribimos `java`, obteniendo el siguiente mensaje:

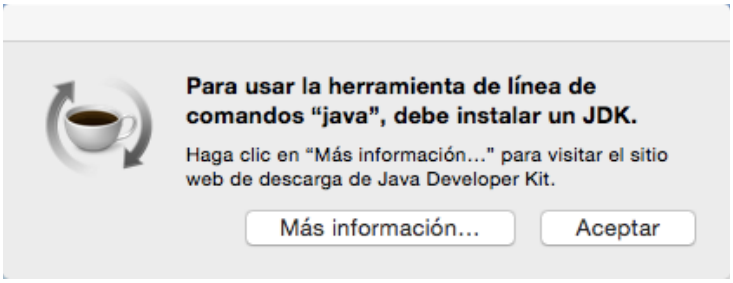


Figura 4: Instalación de Java en OS X Yosemite.

Al aceptar seremos redirigidos a la página de Oracle, donde podremos descargar Java SE JDK[20] 8u40, mediante la descarga del fichero `jre-8u40-macosx-x64.dmg`. Podemos comprobar que la instalación se ha realizado correctamente volviendo al Terminal e introduciendo:

```
java -version
```

y obtendremos algo similar a esto:

```
java version "1.8.0_40"
Java(TM) SE Runtime Environment (build 1.8.0_40-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.40-b25, mixed mode)
```

[19] Java Server Pages

[20] Standard Edition Java Development Kit

Ahora ya podemos instalar Tomcat siguiendo una serie de pasos:

1. Descargamos la distribución binaria del núcleo de la página de Apache Tomcat[21], `apache-tomcat-7.0.59.tar.gz`.
2. Al descomprimir el archivo obtenemos una estructura de carpetas en nuestro directorio de Descargas.
3. Abrimos una Terminal y movemos la estructura a /usr/local. Para utilizar el comando sudo se solicitará la contraseña de administrador.
   ```
   sudo mkdir -p /usr/local
   sudo mv ~/Downloads/apache-tomcat-7.0.59 /usr/local
   ```
4. Para que sea sencillo remplazar esta versión por futuras actualizaciones, crearemos un enlace simbólico que usaremos cuando nos refiramos a Tomcat.
   ```
   sudo rm -f /Library/Tomcat
   sudo ln -s /usr/local/apache-tomcat-7.0.59 /Library/Tomcat
   ```
5. Cambiamos el propietario del directorio /Library/Tomcat.
   ```
   sudo chown -R Luisa /Library/Tomcat
   ```
6. Hacemos todos los scripts ejecutables.
   ```
   sudo chmod +x /Library/Tomcat/bin/*.sh
   ```

Podemos arrancar y parar Tomcat de la siguiente manera en nuestra Terminal:

```
ponki:~ Luisa$ /Library/Tomcat/bin/startup.sh
Using CATALINA_BASE:   /Library/Tomcat
Using CATALINA_HOME:   /Library/Tomcat
Using CATALINA_TMPDIR: /Library/Tomcat/temp
Using JRE_HOME:        /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/
Contents/Home
Using CLASSPATH:       /Library/Tomcat/bin/bootstrap.jar:/Library/Tomcat/
bin/tomcat-juli.jar
Tomcat started.
ponki:~ Luisa$ /Library/Tomcat/bin/shutdown.sh
Using CATALINA_BASE:   /Library/Tomcat
Using CATALINA_HOME:   /Library/Tomcat
Using CATALINA_TMPDIR: /Library/Tomcat/temp
Using JRE_HOME:        /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/
Contents/Home
Using CLASSPATH:       /Library/Tomcat/bin/bootstrap.jar:/Library/Tomcat/
bin/tomcat-juli.jar
```

[21] http://tomcat.apache.org/download-70.cgi

También podemos usar una pequeña aplicación gratuita de Activata[22], Tomcat Controller Versión 1.2.0, que nos proporciona una interfaz de usuario para iniciar y parar nuestro servidor.

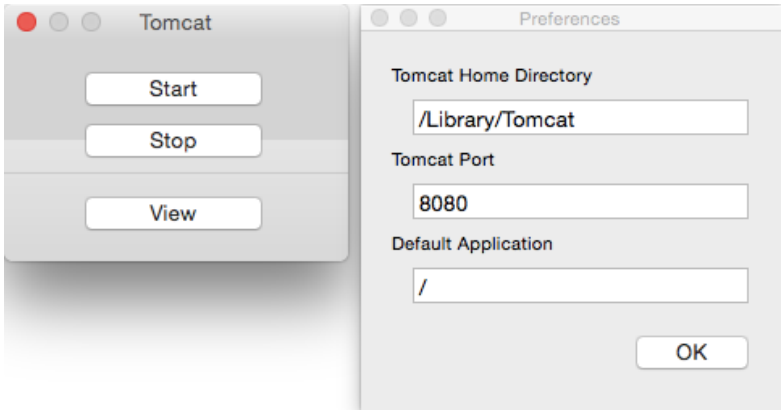


Figura 5: Tomcat Controller, de Activata.

Una vez arrancado Tomcat, podemos comprobar su correcto funcionamiento si entrando en http://localhost:8080 obtenemos la página por defecto de Apache.

[22] http://www.activata.co.uk/downloads/

3. Metodologías.

3.1. El Proceso del Software.

Con el Proceso del Software se establece un marco para el desarrollo del mismo que facilita los procedimientos de desarrollo y la gestión de éstos.

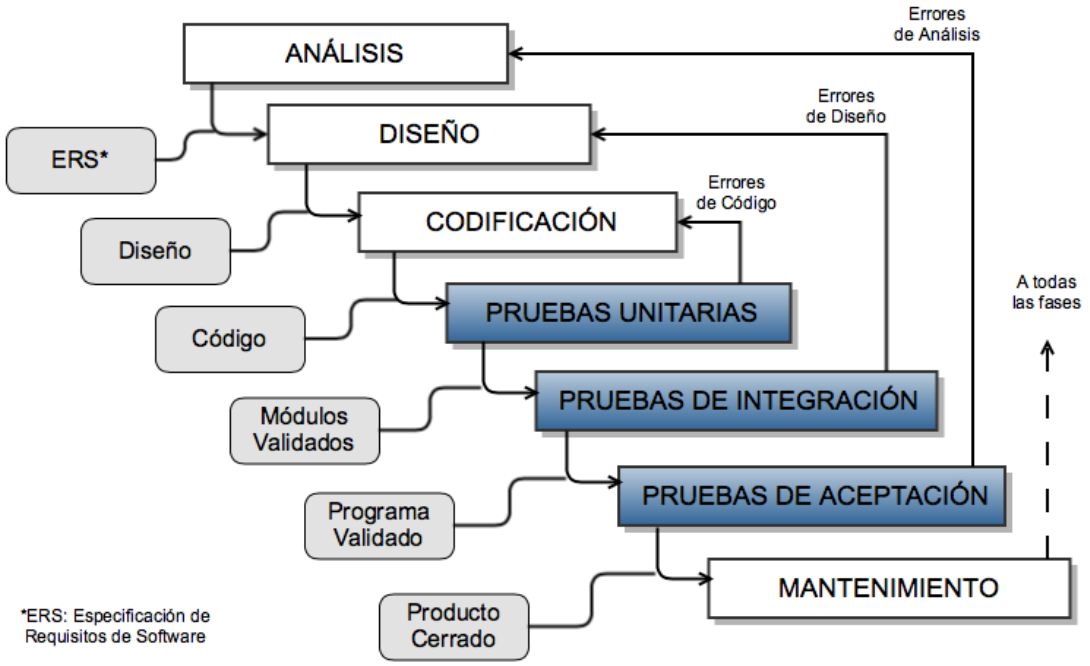


Fig. 6: Modelo Clásico o en Cascada (Real)

Seguiremos un modelo en cascada real, derivado del ideal, en el que desde cada fase sólo se puede volver a la anteriormente superior. En el modelo usado, sin embargo, se vuelve a cualquier fase anterior a medida que se encuentran errores según avanza el desarrollo, por lo que es mucho más flexible y realista.

Siguiendo este modelo tenemos que tener en cuenta la dificultad de establecer todos los requisitos al comienzo del proceso, para evitar detectar errores muy tardíamente y que se han propagado a lo largo de varias etapas, por lo que son difíciles de eliminar. Por esto hacemos una reflexión en el siguiente apartado sobre la importancia de las primeras fases de nuestra aplicación: el análisis y el diseño.

3.1.1. Importancia de las fases de análisis y diseño.

En el desarrollo de un Sistema Software, podemos encontrar las siguientes fases, descritas en el apartado anterior: análisis, diseño, codificación y pruebas, pertenecientes al desarrollo inicial, así como el mantenimiento. A continuación mostramos estas fases desglosadas en el tiempo invertido para su desarrollo.

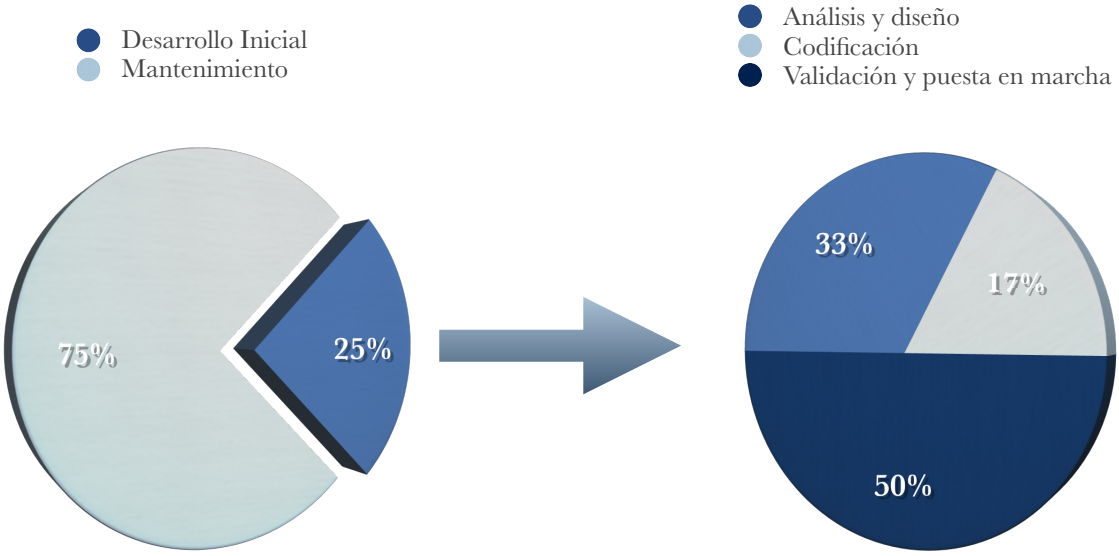


Figura 7: Inversión en el desarrollo de Sistemas Software.

El 50% de los errores cometidos en el desarrollo de Sistemas Software se produce durante la fase de análisis y, a su vez, estos errores son los más costosos de eliminar ya que si nos damos cuenta durante la fase de codificación de un mal análisis, el error ya se ha propagado y habrá que subsanar no sólo la codificación, sino también el diseño.

Por esto es importante hacer dedicar tiempo a hacer un buen análisis para obtener un software de calidad, que concuerde con:

- Los requisitos funcionales y de rendimiento establecidos explícitamente.
- Los estándares de desarrollo explícitamente documentados.
- Las características implícitas que se espera de todo software desarrollado profesionalmente.

Podemos clasificar los factores de calidad centrándonos en tres aspectos importantes de un producto software: sus características operativas, su capacidad para soportar los cambios y su adaptabilidad a nuevos entornos.

3.2. Metodología de Diseño de Bases de Datos.

Existen distintas metodologías para el diseño de bases de datos, pero para este proyecto, se considerarán tres fases en el desarrollo del Sistema de Información, tal y como propone [1], y que se muestran en la siguiente figura:

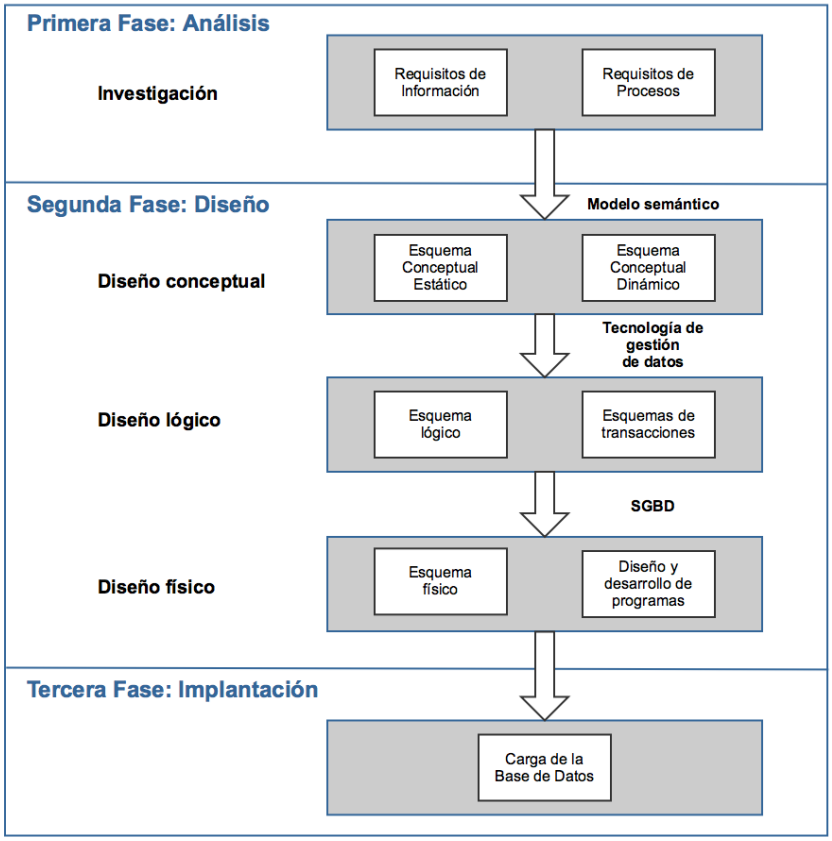


Fig 8: Metodología para el diseño de BD en tres fases.

En la fase de diseño, durante el diseño físico, se tienen en cuenta detalles de la representación física de los datos , y atendiendo a criterios de eficiencia se eligen estructuras de almacenamiento y caminos de acceso específicos para que las aplicaciones que acceden a la información contenida en los ficheros de la base de datos tengan un buen rendimiento.

Dado el pequeño tamaño del sistema de información requerido y la baja previsión del número de usuarios, obviaremos esta fase durante esta memoria y el desarrollo de la aplicación.

4. Análisis.

4.1. Modelado conceptual.

El modelado conceptual es el proceso de construcción de un modelo o especificación detallada del problema del mundo real al que nos enfrentamos. Está desprovisto de consideraciones de diseño e implementación.

Es necesaria la construcción de un modelo del sistema. Haciendo un símil, el desarrollo de un modelo de un sistema software de calidad industrial antes de su construcción o renovación es tan necesario como obtener un plano para construir un edificio. Construimos modelos de sistemas complejos porque inicialmente no somos capaces de comprender el sistema en su totalidad, y además disponer de buenos modelos es esencial para facilitar la comunicación entre los diversos miembros de los equipos de desarrollo.

4.1.1. Modelado conceptual Orientado a Objetos.

Según **Booch** [2], el modelado conceptual OO[23] es *"un proceso que examina los requisitos desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema"*.

El modelado conceptual OO es próximo a los mecanismos cognitivos humanos. Los métodos de este tipo de modelado indican cómo capturar y representar el conocimiento del problema desde una perspectiva orientada a objetos, proporcionando:

- Un modelo de objetos bien definido: Conceptos fundamentales del modelado OO y su semántica asociada: clases, relaciones entre clases, interacción entre objetos, métodos, atributos, comportamiento, etc.
- Una notación: Una representación visual o textual de los conceptos incluidos en el Modelo de Objetos.
- Guías: Para identificar, especificar y verificar conceptos del modelado OO. Orientadas a la estructura y/o al comportamiento.

[23] Orientado a Objetos

4.1.1.1. Actividades del Modelado OO.

- Definición del problema y recogida de datos.
- Identificación de las clases y la estructura del dominio del problema.
- Identificación del comportamiento de los objetos.
- Identificación de los patrones de interacción entre objetos.
- Integrar y revisar modelos, redefiniendo si es necesario.

4.2. UML: Lenguaje Unificado de Modelado.

ULM[24], es un lenguaje de propósito general para el modelado OO. Según el OMG[25], desarrollador de este lenguaje gráfico de modelado [3], *"UML ayuda a especificar, visualizar y documentar modelos de sistemas software, incluyendo su estructura y diseño, de una forma en la que que se cumple con todos los requisitos".*

ULM combina notaciones del Modelado:

- Orientado a Objetos.
- De datos.
- De Componentes.
- De flujos de trabajo.

4.2.1. Diagrama de Clases.

El Diagrama de Clases es el diagrama principal para el modelado y diseño OO. Presenta las clases del sistema con sus relaciones estructurales, asociación y agregación, y de herencia. La definición de una clase incluye definiciones para atributos y operaciones.

Los enlaces a nivel de objetos pueden representarse en el mundo de las clases. Las formas de relación entre clases son:

- Asociación y Agregación, vista esta última como un caso particular de asociación. Para definir estas, necesitamos saber que un enlace es una conexión conceptual semántica entre dos o más objetos. Es una instancia de una asociación.
    - Asociación: describe un grupo de enlaces con estructura y semántica comunes. Se representa con una línea.

[24] Unified Modeling Language

[25] Object Management Group

- Inherentemente bidireccionales.
- Pueden ser binarias, ternarias o de orden superior.
- Pueden etiquetarse con un nombre que identifica de forma única un extremo de la la asociación (semánticamente).
- Debe definirse la multiplicidad: especifica el número de instancias de una clase que pueden estar relacionadas con una única instancia de una clase asociada.

- Agregación: Es una relación parte-de en la cual los objetos que representan los componentes de algo, se asocian a un objeto que representa el ensamblaje completo. Es una forma fuertemente acoplada de asociación con más semántica. Algunas propiedades son: transitividad, antisimetría y propagación. Se representa con un rombo en el lado del objeto completo.

- Generalización y Especialización: Permiten gestionar la complejidad mediante un ordenamiento taxonómico. Se obtienen usando los mecanismos de abstracción de generalización y/o especialización y consiste en factorizar los elementos comunes de un conjunto de clases en una clase más general. Se representa con una flecha desde las clases derivadas o subclases hacia la clase base o superclase. Al igual que en el diseño de la base de datos encontramos restricciones semánticas entre las clases derivadas: solapada o disjunta y completa o incompleta.

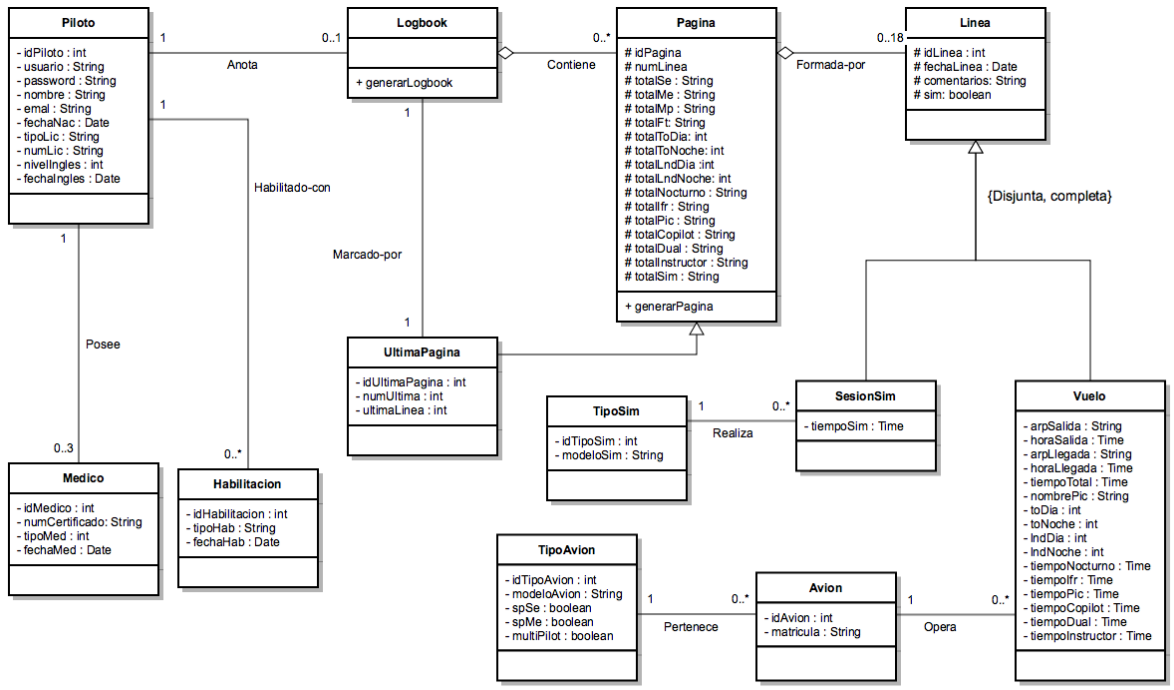


Fig. 9: Diagrama de Clases.

4.2.2. Modelo de Casos de Uso.

Los modelos de casos de uso fueron introducidos por Ivar Jacobson durante su etapa en la compañía Objectory AD, alrededor de 1992.
Los casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista de los usuarios.
Son descripciones de la funcionalidad del futuro sistema independientes de la implementación, que sirven para captar los requisitos funcionales de un sistema software.

Ivar Jacobson propone dos definiciones:

- Documento que describe una secuencia de eventos que realiza un actor o agente externo, que usa el sistema para llevar a cabo un proceso que tiene algún valor para él.
- Cada caso de uso está formado por una secuencia de eventos, iniciada por un actor, que describe la interacción que tienen lugar entre el actor y el sistema.

Están estructurados en tres capas:

1. Diagrama de contexto y modelo inicial.
2. Plantillas de descripción.
3. Modelo estructurado.

Para la construcción del modelo de casos de uso usaremos una técnica descendente.

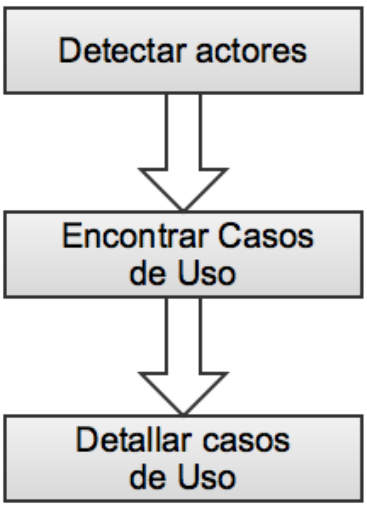


Figura 10: Técnica descendente.

4.2.2.1. Diagrama de Contexto.

El diagrama de contexto muestra los límites del sistema y los actores (agentes externos) que interactuarán con el mismo.

En nuestra aplicación encontraremos sólo un tipo de usuario, un Piloto. Aunque la base de datos es compartida cada piloto tendrá acceso únicamente a sus registros, aunque compartirá con el resto de usuarios información sobre los aviones y

los simuladores ya que estas tablas no contienen una clave ajena que referencie a cada piloto por separado.

A pesar de esto se muestra en el diagrama un segundo tipo de usuario ficticio llamado Supervisor. Este supervisor podría tener diferentes roles. Entre ellos, podría ser el administrador de la base de datos y tener acceso por lo tanto a todos los registros idependientemente del usuario. En otro caso hipotético y futuro en el que los registros electrónicos fueran legales, el usuario supervisor recibiría permisos de cierto número de usuarios, que a través de la aplicación podrían generar unas claves para dar acceso a sus registros a terceros. Este usuario podría pertenecer a una compañía aérea y controlar a sus empleados o pertencer a una autoridad aeronaútica competente.

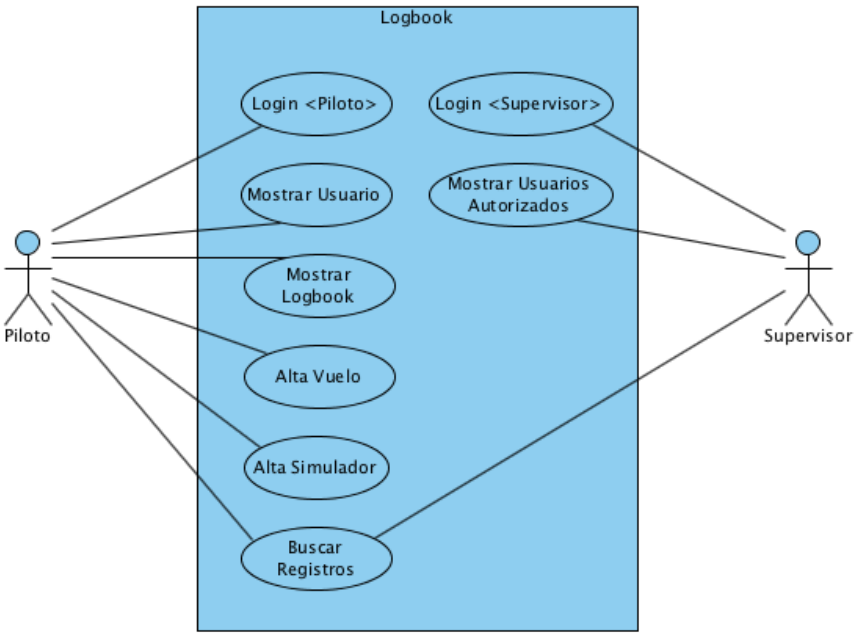


Figura 11: Diagrama de contexto.

4.2.2.2. Plantillas de Descripción.

Los casos de uso se describen utilizando plantillas en lenguaje natural. Habitualmente muestran: nombre el Caso de Uso, descripción, actores (primario y secundarios), precondiciones, postcondiciones y el flujo de eventos.

Se describen a continuación los casos de uso Login, Mostrar Usuario, Mostrar Logbook, Alta Vuelo, Alta Simulador y Buscar Registros.

Caso de uso	Login.
Actores	Piloto.
Propósito	Entrar en la aplicación.
Resumen	Un cliente entra en la aplicación mediante su usuario y contraseña. Si es la primera vez que accede tendrá que registrarse. Si el usuario no existe será redirigido automáticamente a la página de registro, mientras que si comete un error en la contraseña se le llevará de nuevo a la página de login.
Precondiciones	Poseer un usuario y contraseña válido o registrarse.
Postcondiciones	Si el usuario se registra, se almacena la información en la base de datos.
Incluye	-
Extiende	Mostrar Usuario, Mostrar Logbook, Alta Vuelo, Alta Simulador y Buscar Registros.
Hereda de	-

Figura 12: Plantilla Login.

Caso de uso	Mostrar Usuario.
Actores	Piloto.
Propósito	Mostrar la página de bienvenida a la aplicación con los datos del usuario.
Resumen	
Precondiciones	Usuario autenticado.
Postcondiciones	Guardar todos los cambios de la aplicación en la base de datos.
Incluye	Modificar Datos Personales, Modificar Certificado de Inglés, Modificar Certificado Médico, Modificar Habilitación y Eliminar Usuario
Extiende	-
Hereda de	-

Figura 13: Plantilla Mostrar Usuario.

Caso de uso	Mostrar Logbook
Actores	Piloto
Propósito	Mostrar libro del usuario a partir de la última página que rellenó en su versión física.
Resumen	Muestra todas las líneas después de la última línea de la última página, es decir, todas las líneas que el usuario aún no ha anotado en su libro físico. - Si hay menos de 18 nuevas líneas: Muestra las líneas y los totales de las páginas anteriores. - Si hay 18 líneas: La página está completa. Muestra las líneas, los totales de la página actual y los totales de las páginas anteriores. Actualiza última página con los nuevos valores. - Si hay más de 18 líneas: Muestra un botón de página siguiente y se va de vuelta al primer punto.
Precondiciones	Usuario autenticado, que exista alguna línea nueva que anotar.
Postcondiciones	Anotar todos los cambios en la BBDD, especialmente cambios en última página.
Incluye	-
Extiende	-
Hereda de	-

Figura 14: Plantilla Mostrar Logbook.

Caso de uso	Alta Vuelo.
Actores	Piloto.
Propósito	Mostrar un formulario para introducir un nuevo vuelo y controlar la información sobre los tipos de avión y las matrículas.
Resumen	Desde la página Vuelo se pueden realizar las siguientes operaciones: - Introducir un vuelo: El sistema calcula el tiempo de vuelo con la hora de salida y llegada y obtiene el tipo de operación del tipo de avión seleccionado por el usuario. Así mismo, muestra de forma asíncrona un menú desplegable con las matrículas que pertenecen al tipo de avión seleccionado por el usuario. - Dar de alta un modelo de avión: Se abre un formulario en una ventana nueva y, una vez guardado el tipo de avión, se actualiza el menú desplegable en Vuelo conteniendo el nuevo tipo. - Dar de alta una matrícula: Igual que dar de alta un tipo de avión. Debe de existir previamente el tipo de avión al que pertenece la matrícula que queremos introducir.
Precondiciones	Usuario autenticado, existe el modelo de avión y la matrícula en la BBDD.
Postcondiciones	Guardar todos los cambios de la aplicación en la base de datos.
Incluye	Alta Modelo y Alta Matrícula
Extiende	-
Hereda de	-

Figura 15: Plantilla Alta Vuelo.

Caso de uso	Alta Simulador.
Actores	Piloto.
Propósito	Mostrar un formulario para introducir una nueva sesión de simulador y controlar la información sobre los tipos de simuladores
Resumen	Desde la página Simulador se pueden realizar las siguientes operaciones: - Dar de alta una sesión de simulador. - Dar de alta un nuevo tipo de simulador: Se abre un formulario en una ventana nueva y, una vez guardado el nuevo tipo, se actualiza el menú desplegable en la página de Simulador, conteniendo el nuevo tipo
Precondiciones	Usuario autenticado, existe el tipo de simulador con el que se realiza la sesión.
Postcondiciones	Guardar todos los cambios en la aplicación en la base de datos.
Incluye	Alta Tipo Simulador.
Extiende	-
Hereda de	-

Figura 16: Plantilla Alta Simulador.

Caso de uso	Buscar Registros.
Actores	Piloto.
Propósito	Mostrar un formulario para introducir los criterios de búsqueda y generar una tabla similar a la vista de logbook con los registros encontrados.
Resumen	Desde la página Buscar se pueden realizar las siguientes operaciones: - Buscar registros que cumplan con los requisitos de búsqueda especificados por el usuario. - Borrar los registros encontrados desde la página de resultados.
Precondiciones	Usuario autenticado.
Postcondiciones	Guardar todos los cambios en la aplicación en la base de datos.
Incluye	Borrar Registros.
Extiende	-
Hereda de	-

Figura 17: Plantilla Buscar Registros.

4.2.2.3. Modelo estructurado.

Contiene los actores y las relaciones entre los casos de uso. Diagramas realizados con Visual Paradigm v.12.2[26]

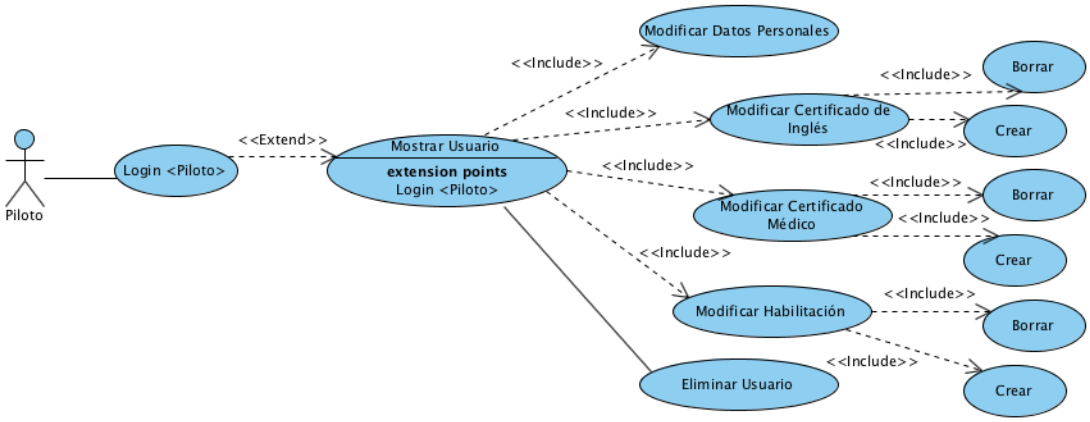


Figura 18: Modelo Mostrar Usuario.

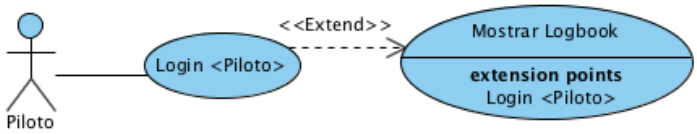


Figura 19: Modelo Mostrar Logbook

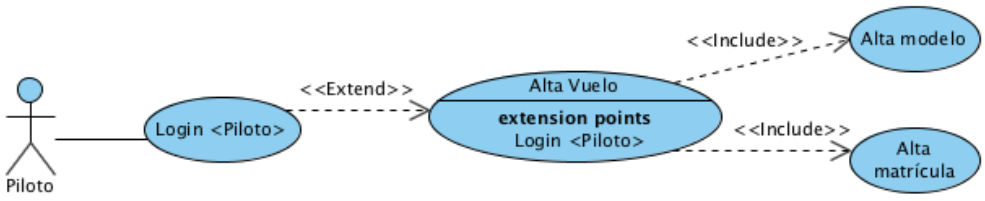


Figura 20: Modelo Alta Vuelo.

[26] www.visual-paradigm.com

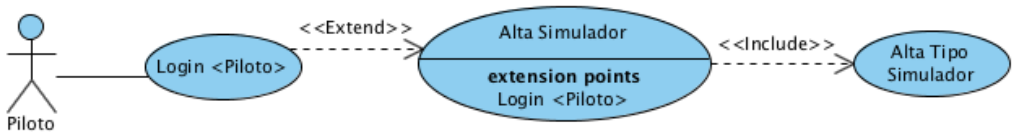


Figura 21: Modelo Alta Simulador.

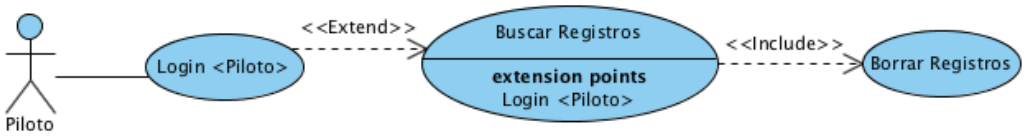


Figura 22. Modelo Buscar Registros.

4.2.3. Diagrama de secuencia.

Se caracteriza por mostrar la secuencia de mensajes entre objetos durante un escenario concreto:

- Un escenario es una secuencia de sucesos que se produce durante una ejecución concreta de un sistema, una transición de información en un instante determinado.
- Un suceso o evento es algo que transcurre durante un periodo de tiempo. Un suceso es una transmisión de información de dirección única entre un objeto y otro. No es como una llamada a subrutina, que proporciona un valor: un objeto envía un suceso a otro objeto y puede esperar una respuesta, pero la respuesta es otro suceso distinto.
- Muestra los actores y objetos que participan.
- Cada objeto viene dado por una barra vertical, que representa la línea de vida del objeto.
- El tiempo transcurre de arriba hacia abajo.
- El orden de aparición de los objetos no es importante.
- Un mensaje se representa como una línea con fecha desde el emisor hasta el receptor, uniendo sus líneas de vida.
- Cuando un objeto tiene una activación, un rectángulo estrecho cubre su línea de vida (caja de activación). Representa la activación del método en la pila de ejecución.

- Los enlaces muestra el intercambio de mensajes entre objetos:
  - El intercambio puede ser en los dos sentidos.
  - Se pueden enviar varios mensajes en la misma dirección.
- La flecha indica la dirección del mensaje.
- La secuencia de números describe el orden de envío.
- Un objeto también puede dirigirse mensajes a sí mimo.

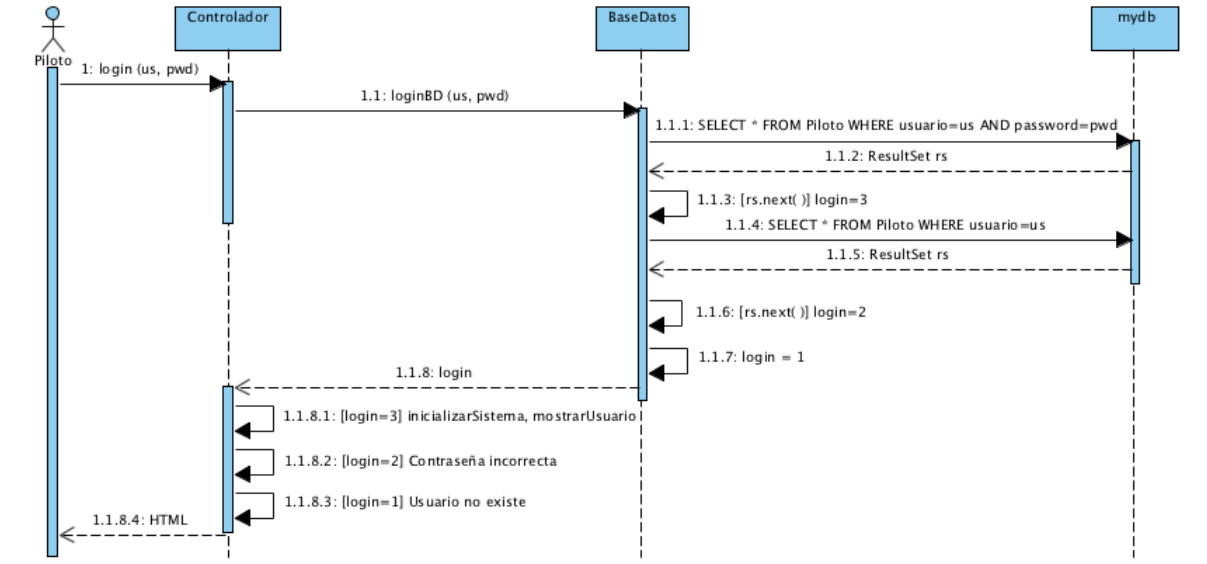


Figura 23: Diagrama de secuencia: Login.

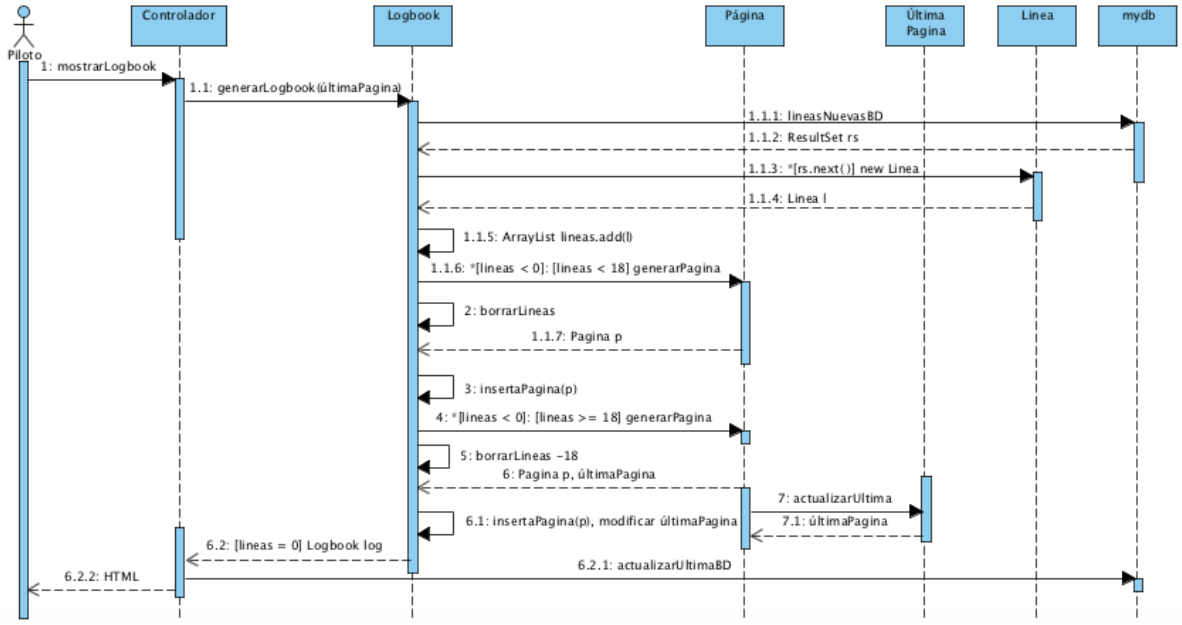


Figura 24: Diagrama de secuencia: Mostrar Logbook.

4.3. Base de Datos: Requisitos de Información.

Como se ha explicado anteriormente, y se muestra en la Figura x, durante la fase de análisis de nuestra aplicación y, concretamente, de la base de datos; debemos realizar una investigación en la que el objetivo será descubrir el conjunto de requisitos de información por una parte, y de proceso por otra, que nuestro logbook necesita para cumplir sus fines. Este proceso se lleva a cabo mediante técnicas heurísticas, es decir, reglas, recomendaciones o sugerencias a seguir que en ningún caso establecen el proceso exacto de realización de una tarea por su alto componente creativo.

Para la puesta en marcha de nuestro logbook electrónico, queremos almacenar la siguiente información:

- Pilotos: serán los usuarios de nuestra aplicación, por lo que guardaremos su nombre de usuario y contraseña. También datos personales como su nombre, su email y su fecha de nacimiento, así como información sobre su licencia y su nivel de Competencia Lingüística ICAO en el idioma inglés. De estos últimos guardaremos también su fecha de expiración para mostrar avisos al usuario.
- Reconocimiento Médico: Los pilotos con licencia deben de acompañar a esta con un reconocimiento médico válido. Existen tres tipos de reconocimientos:
  - Clase 1: Para pilotos comerciales y de transporte de líneas aéreas.
  - Clase 2: Pilotos privados, de ultraligero y tripulantes de cabina de pasajeros.
  - Clase 3: Controladores Aéreos y otras actividades aéreas.

  Para volar se requiere un reconocimiento de Clase 1 o de Clase 2, dependiendo de la actividad que se vaya a ejercer, y un mismo piloto puede estar en posesión de uno de los dos, o de ambos, dependiendo de las funciones que haya ido ejerciendo a lo largo de su vida profesional.
- Habilitaciones: Existen habilitaciones de Clase y de Tipo, pero esta información no es relevante para nuestra aplicación. Si es importante almacenar información de cada una de las habilitaciones que posea el piloto y su fecha de renovación para mandar avisos cuando esta esté próxima.
- Vuelos: Guardaremos toda la información que se requiere anotar en el logbook: fecha del vuelo, aeropuerto y hora de salida, de llegada, tiempo total del vuelo, nombre del piloto al mando, despegues y aterrizajes diurnos y nocturnos, tiempo de vuelo nocturno, bajo reglas instrumentales, de piloto al mando, de copiloto, en dual, o como instructor, así como los comentarios. Para identificar un vuelo tendríamos que hacer uso de la tupla fecha y hora de salida, por lo que necesariamente usaremos una clave primaria autoincrementada de forma similar a la numeración de vuelos de las compañías aéreas. También es necesario anotar el tiempo de

monomotor, de multimotor y de multipiloto; pero, ya que esta información es relativa al tipo de avión, se autocompletará en función del tipo al que pertenezca la matrícula del avión con el que se operó el vuelo, aliviando así el número de campos del formulario a rellenar por el usuario.

- Avión: Se identifican por su número de matrícula y pertenecen a un tipo de avión, que se codifica de forma estándar, como por ejemplo C152 para la Cessna 152, A320 para el Airbus 320, B737 para el Boeing 737, etc.
- Tipo de Avión: Pueden ser multipiloto o mono-piloto y, dentro de esta última, con un motor o con varios.
- Sesión de Simulador: Los pilotos se entrenan en simuladores. Se guardarán la fecha del simulador, el tiempo y los comentarios. Guardaremos en una tabla aparte el tipo de Simulador.
- Tipo de Simulador: FNPTI, FNPT II o FFS. Si es FFS tiene una matrícula que lo identifica, al igual que si se tratara de un avión real.

En el análisis de la realidad se han detectado las siguientes **restricciones de integridad**:

- No puede haber dos pilotos con el mismo nombre de usuario ni con el mismo correo electrónico.
- Un vuelo será realizado por un piloto, y sólo uno. Podría darse la coincidencia en el mundo real de que dos pilotos, usuarios de este logbook, realizaran un vuelo juntos en un avión multitripulado. Sin embargo, las posibilidades de que esto suceda son mínimas. Además, cada usuario debe tener su logbook independiente del resto de usuarios, por lo que este vuelo sería representado dos veces en la tabla con un número de vuelo diferente. Por esto, a efectos de implementación, podemos decir que un vuelo será realizado por un solo piloto.
- No puede haber dos vuelos con el mismo número de vuelo.
- Un vuelo se realiza con un avión, que pertenece a un tipo de avión. Este tipo de avión determina el número de tripulantes y, en caso de tener un solo piloto, se especificará si es de un solo motor o de varios. Diferentes aviones, con diferentes matrículas pueden pertenecer a un mismo tipo de avión.
- No puede haber dos aviones con la misma matrícula.
- No puede haber dos tipos de avión que pertenezcan al mismo modelo.
- Una sesión de simulador será realizada por un sólo piloto, al igual que se ha comentado con respecto a los vuelos, y se realizará en un tipo de simulador.
- No puede haber dos sesiones de simulador con el mismo índice de sesión de simulador.

- Un piloto puede tener varias habilitaciones, las cuales deben de incluir la fecha de renovación.
- Un piloto puede tener varios certificados médicos, los cuales deben de incluir la fecha de renovación.
- No puede haber dos certificados médicos con el mismo número de certificado.

Además de todas las características anteriores, se detectan los siguientes problemas a los que habrá que buscar solución:

- Aunque la información sobre Vuelos y Simuladores se guarda en tablas separadas, tiene que representarse de forma conjunta en la vista del logbook. Una posible solución al problema sería crear una entrada adicional en la tabla Vuelo cada vez que se introdujera un Simulador en la que únicamente se guardara la fecha y una referencia al índice de sesión de simulador en alguno de los campos. Posteriormente se halla una solución mucho más sencilla y elegante: definir una generalización. La generalización es el proceso inverso a la especialización por el que se generalizan varias clases para obtener una abstracta de más alto nivel que incluya los objetos de todas estas clases. Es una síntesis conceptual y podemos llegar a ella siguiendo una estrategia ascendente: existe en el esquema un conjunto de entidades con algunas propiedades similares (Vuelo y SesionSim) y que, en la realidad se podrían clasificar en un objeto común (podríamos referirnos a este objeto como un "Entrenamiento" o simplemente una "Línea" de nuestro logbook, que tendría como atributos el número de entrenamiento/línea y la fecha y que serían heredados por Vuelo y SesionSim. La generalización total, pues toda entrada debe de pertenecer a alguna de las dos subclases y disjunta, pues sólo perteneceran a una.
- En cada página del logbook se incluye un total de horas de las páginas previas. Cuando un usuario no tenga mucha experiencia, sumar todas estas horas no será un proceso muy costoso, pero cuando hablamos de miles de horas y de entradas en el logbook, hacer esta operación una y otra vez puede ralentizar el sistema. Podríamos guardar información sobre la última página actualizada: el número de página y las horas totales en ese momento. Si se opta por esta solución, cada piloto tendrá una única última página que hará referencia al último vuelo de su última página completada en el cuaderno.

5. Diseño.

Es un proceso que extiende, refina y reorganiza los aspectos detectados en el proceso de modelado conceptual, para generar una especificación rigurosa del sistema de información siempre orientada a la obtención de la solución del sistema software.

5.1. Diseño de Objetos.

Una vez se ha analizado el problema, es necesario decidir la estrategia de alto nivel que nos ayude a construir la solución software del modelo conceptual. En la fase de diseño de objetos se explica:

- Cómo implementar los modelos del dominio de la fase de modelado
- Cómo representar las clases, cómo implementar asociaciones y agregaciones.
- Qué estructuras de datos internas son necesarias para la representación y almacenamiento de los objetos y atributos y cómo ajustar las relaciones de herencia, eliminando o añadiendo subclases, reorganizando el estado y comportamiento de éstas, etc.
- Cómo obtener los métodos básicos de lectura y escritura, y los métodos de inserción y borrado de objetos en agregaciones o asociaciones.

Aplicando los patrones de diseño, obtenemos las clases que representarán los objetos de nuestro sistema de información, que se encuentran en el Anexo A. En este apartado representaremos únicamente un ejemplo de la estructura de las mismas:

```
public class Ejemplo {

      /* Atributos */

      /* Métodos de consulta y modificación de los atributos: para cada
      atributo implementaremos un método obtenerAtributo() y
      asignarAtributo(tipoAtributo a) */

/* Constructores */
            /* Un constructor con todos los parámetros para recuperar
            objetos de la base de datos. */
            /* Un constructor sin el atributo id para insertar en la base
            de datos. Esto se debe a que este campo es autoincrementado en
            nuestro diseño. */
```

```
            /* Constructor vacío */
      /* Un método que modifique un objeto */

      /* Relaciones y cardinalidades */

      /* Base de datos */
            /* Método insertarEjemploBD(Ejemplo e) */
            /* Método borrarEjemploBD(Ejemplo e) */
            /* Método modificarEjemploBD(Ejemplo e, String col, String
            nuevoValor) */
            /* Método consultarEjemploBD(id) que devuelve un objeto
            Ejemplo */
}
```

La solución propuesta a través de la aplicación de los patrones es general, dependiendo de la navegabilidad y del uso de las clases, se pueden eliminar:

- Atributos objeto-valuados (referencias a colecciones y objetos)
- Métodos de lectura y escritura de atributos.
- Se pueden personalizar y optimizar los métodos de inserción y borrado en colecciones o arrays.

A pesar de no ser un diseño definitivo, será una buena base y estructura para comenzar a desarrollar nuestra aplicación.

Además de los objetos que conforman el sistema de información surgen clases nuevas, que no dependen del dominio del problema, sino de la solución y que no aparecían en el diagrama de clases de la fase de modelado conceptual. Implementaremos pues, una clase `Controlador`, que será el servlet que interprete la comunicación entre el cliente y el servidor y realice las diferentes opciones demandadas; una clase `BaseDatos`, que realizará la conexión y contendrá métodos genéricos de lectura y escritura con la misma y una clase `Utilidades`, que contendrá métodos varios, como por ejemplo, los encargados de sumar horas, calcular tiempos de vuelo, expiración de licencias y certificados, etc.

5.1.1. Encapsulación.

Si se desea flexibilidad, buen mantenimiento y extensibilidad, nuestro diseño en el código debe de incluir encapsulamiento, para ello debemos de hacer lo siguiente:

1. Mantener las variables de instancia protegidas. En este caso incluimos en cada variable el modificador de acceso private.
2. Hacer métodos de acceso públicos para forzar al acceso a las variables por medio de dichos métodos en lugar de acceder directamente.
3. Utilizar convenciones de código para los nombres de los métodos. En nuestra aplicación usaremos `asignar<nombreAtributo>` y `obtener<nombreAtributo>`. Para los arrays usaremos `insertar<nombreObjeto>, borrar<nombreObjeto>, consultar<nombreObjeto>` y `tamanyo<nombreObjetos>`. Para las ArrayList los métodos serán los mismos que para los arrays, pero añadiremos además `consultarIndice<nombreObjeto>`.

Aunque, en un principio, en nuestros métodos `asignar<nombreAtributo>` no existan validaciones para prevenir que un valor no válido sea asignado a la variable, el proveer de un método de este tipo desde el diseño inicial de la aplicación nos permite posteriormente modificar el comportamiento de la misma sin afectar las variables utilizadas. Así, si en un futuro se desea, por ejemplo, que dicha variable solamente pueda tomar un rango de valores, se podrán aplicar posteriormente los cambios sin que haya repercusiones negativas.

5.1.2. Herencia.

El principio por el cual se dice que el conocimiento de una clase más general es aplicable también a la clase más específica se llama herencia. Los datos y el comportamiento asociados con un nivel de una jerarquía de clases es aplicable automáticamente a los niveles inferiores de la jerarquía.

En el sistema de información propuesto las clases Vuelo y SesionSim heredan de la clase Linea, pues independendientemente del tipo de entrenamiento que el piloto realice se anotará una linea en el cuaderno , pero las caracteresicas de esta anotación son diferentes. También podemos decir que la UltimaPagina es un tipo de Pagina, por lo que usaremos la herencia también en este caso.

Para reconocer a que subtipo pertenece un objeto instanciado como su clase superior podemos usar el operador **`instanceof`**, que devuelve true o false según se cumpla o no la condición. Encontramos un ejemplo en el código de la aplicación insertar una `Línea l` en la base de datos:

```
if (l instanceof Vuelo) {
	/* Es un vuelo y tenemos que insertarlo en la tabla Vuelo además de
	insertar en Linea */
	insertaVueloSim = Vuelo.insertarVueloBD(l);
} else if (l instanceof SesionSim) {
	/* Es una sesión de simulador y tenemos que insertarlo en la tabla
	SesionSim además de insertar en Linea
	insertaVueloSim = SesionSim.insertarSesionSimBD(l);
```

También nos será muy útil para trabajar con la herencia el operador de casting. Desde una de las clases herederas podremos utilizar todos los métodos de la clase base, pero si deseamos hacer lo contrario necesitamos hacer un casting al objeto como en el siguiente ejemplo, en el que después de forzar el tipo de una `Linea l` al de una de sus subclases, `SesionSim`, podemos utilizar el método `obtenerTiempoSim()`, propio de la misma.

```
SesionSim ss = (SesionSim)l;
Time tiempoSim = ss.obtenerTiempoSim();
```

5.1.3. Borrado de Objetos.

Podemos eliminar objetos en Java usando `<nombreObjeto>.dispose()`, pero se toma como decisión durante esta fase de diseño no implementaremos métodos de borrado. Dejaremos que el Garbage Colector se encargue de los objetos que ya no están instanciados. No se descarta, sin embargo, usar la primera solución en algún caso concreto.

5.2. Diseño de la Interfaz de Usuario.

La interfaz de usuario es el medio de comunicación que permite que un usuario interaccione con un sistema. Los principios generales de diseño de las GUI[27] son, según Sommerville [4]:

- **Familiaridad del usuario:** La interfaz debe utilizar términos y conceptos que se toman de la experiencia de las personas que más utilizan el sistema. Esta aplicación contiene una gran cantidad de términos propios del lenguaje aeronáutico, pero puesto que ese es su público exclusivo, el uso de un lenguaje técnico se convierte en una ventaja.

[27] Graphic User Interface

- **Consistencia:** La interfaz debe ser consistente en el sentido de que operaciones similares se realizan o activan de la misma forma. Aplicamos la consistencia a la modificación de objetos. Encontramos un menú desplegable en el que si seleccionamos "Nuevo Objeto" crearemos uno, si en cambio seleccionamos uno de la lista lo modificaremos y, por último, si ticamos la casilla "Eliminar" lo borraremos. Estas pantallas serán iguales para todos los objetos.
- **Mínima sorpresa:** El comportamiento del sistema no debe provocar sorpresa a los usuarios.
- **Recuperabilidad:** La interfaz debe incluir mecanismos que permitan a los usuarios recuperarse de los errores.
- **Guía al usuario:** La interfaz debe proveer retroalimentación significativa y características de ayuda sensible al contexto. Se plantea en el capítulo de ampliaciones desarrollar un sistema de ayuda en línea.
- **Diversidad de usuarios:** La interfaz debe proveer características de interacción apropiadas para los diferentes tipos de usuarios del sistema. El grupo de usuarios en el que se centra la aplicación a desarrollar está tan delimitado que podemos obviar este punto, así como las funciones de accesibilidad, dadas las condiciones de buena salud que deben de poseer los pilotos para pasar sus reconocimientos médicos.

Un **prototipo** es una representación concreta de un sistema interactivo o una parte del mismo. Su finalidad es explorar los diferentes aspectos del sistema, como usabilidad, accesibilidad y funcionalidad. Es una herramienta muy útil para hacer participar activamente al usuario en el desarrollo y poder realizar la evaluación del producto.

Existen diferentes técnicas de prototipado, pero ya que la definición de prototipos muy elaborados desde un inicio es una actividad muy costosa, en esta ocasión se ha elegido la técnica de bocetos. Éstos están realizados con la aplicación Balsamiq Mockups 3.1.9[28].

[28] https://balsamiq.com

Figura 25: Mockup login.html.

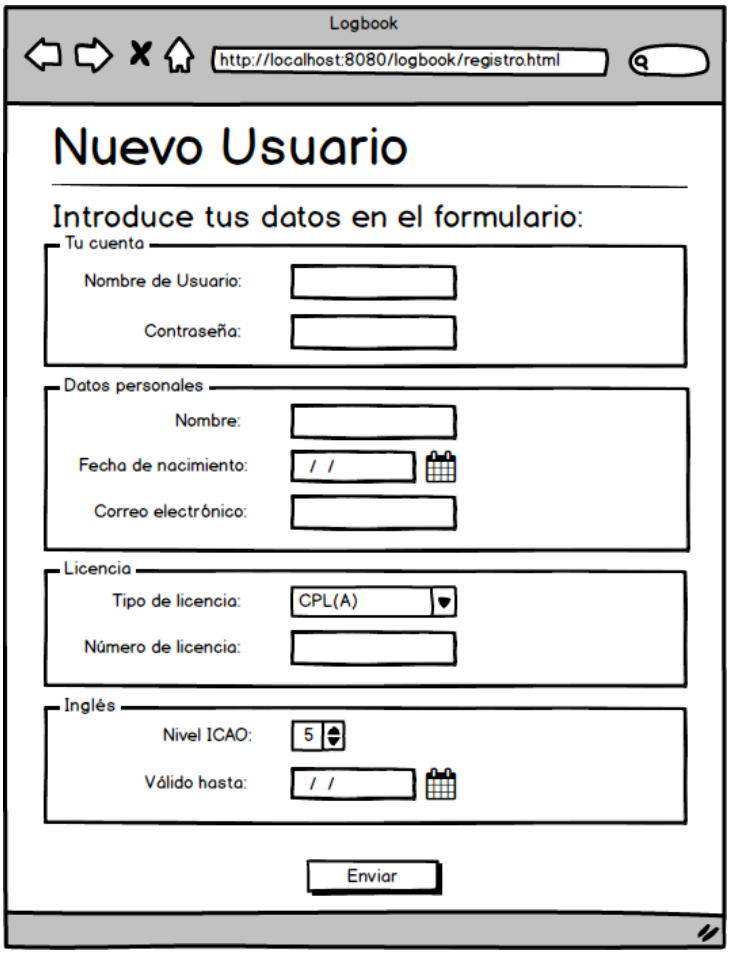


Figura 26: Mockup registro.html.

Figura 27: Mockup usuario.html.

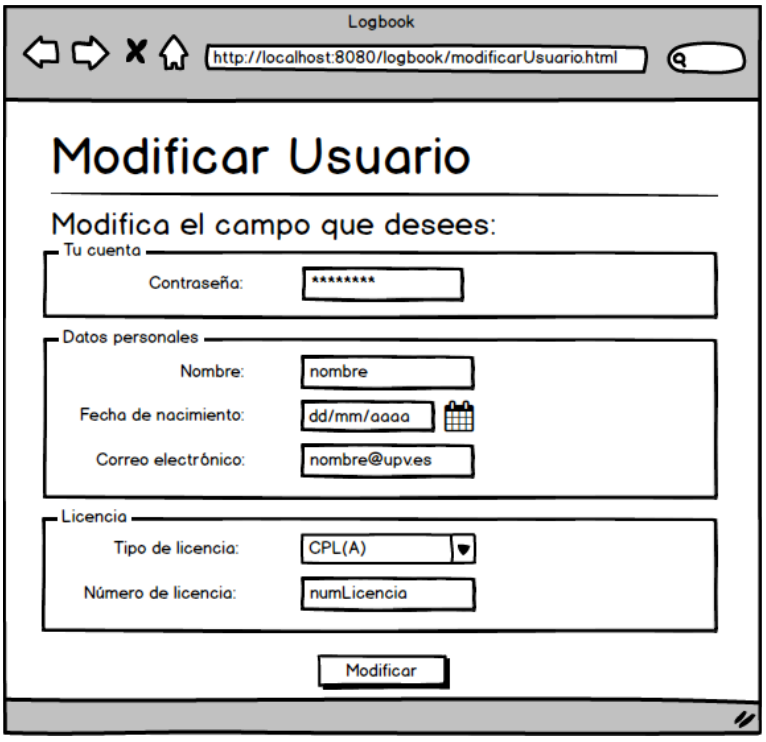


Figura 28: Mockup modificarUsuario.html.

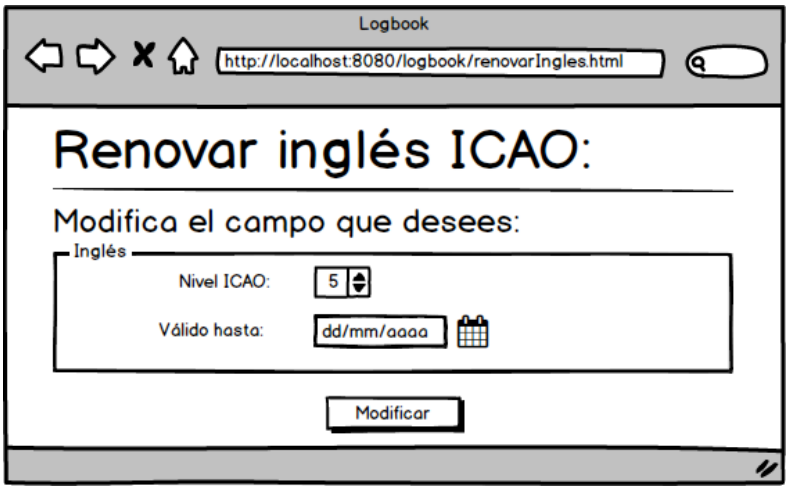


Figura 29: Mockup renovarIngles.html.

Figura 30: Mockup modificarCertificado.html.

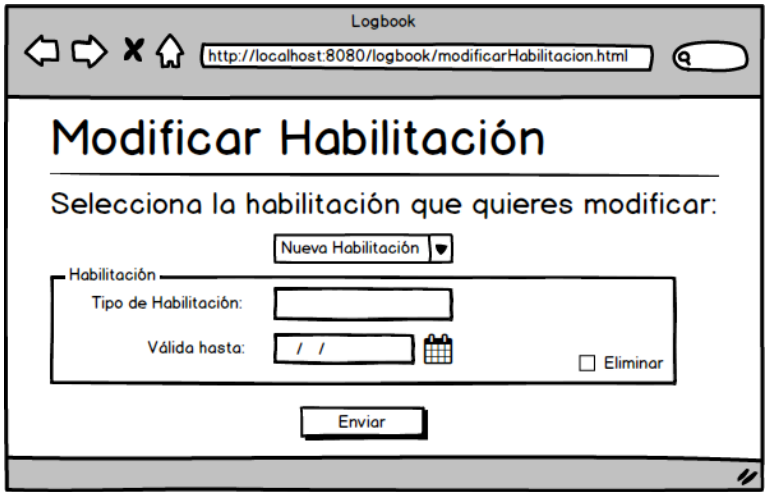


Figura 31: Mockup modificarHabilitacion.html.

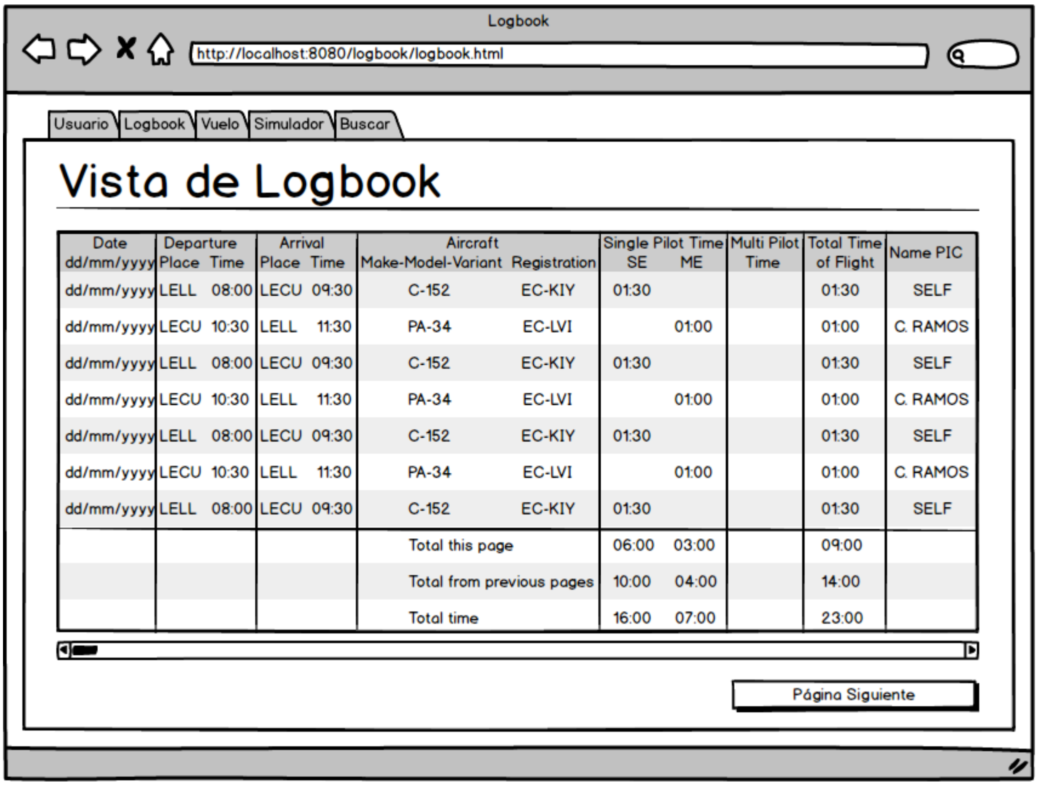


Figura 32: Mockup logbook.html.

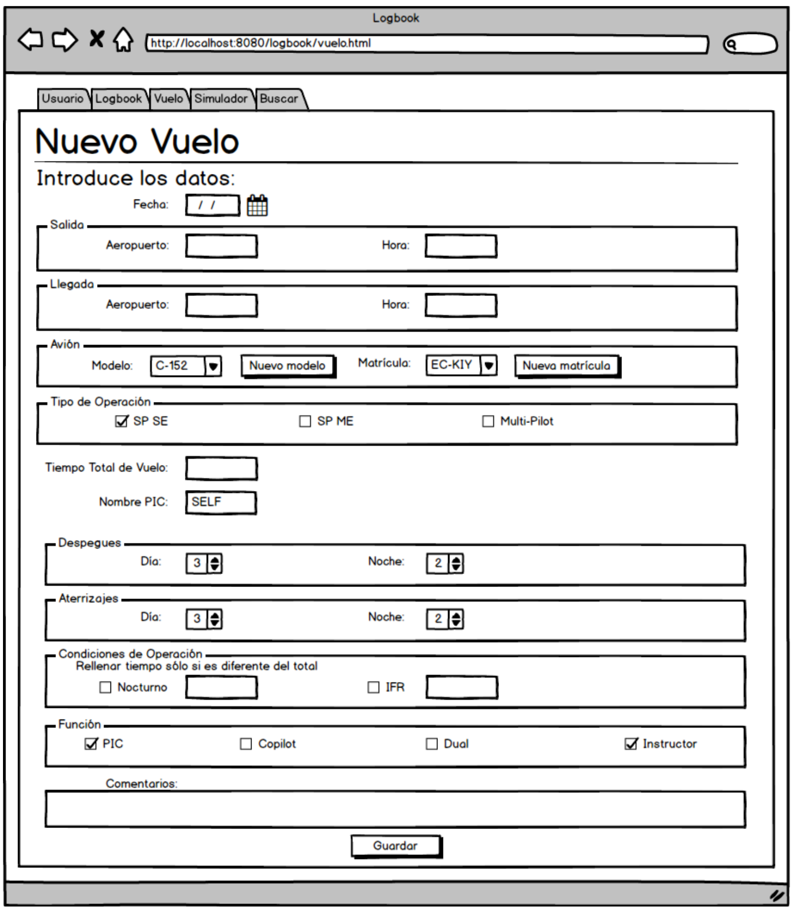


Figura 33: Mockup vuelo.html.

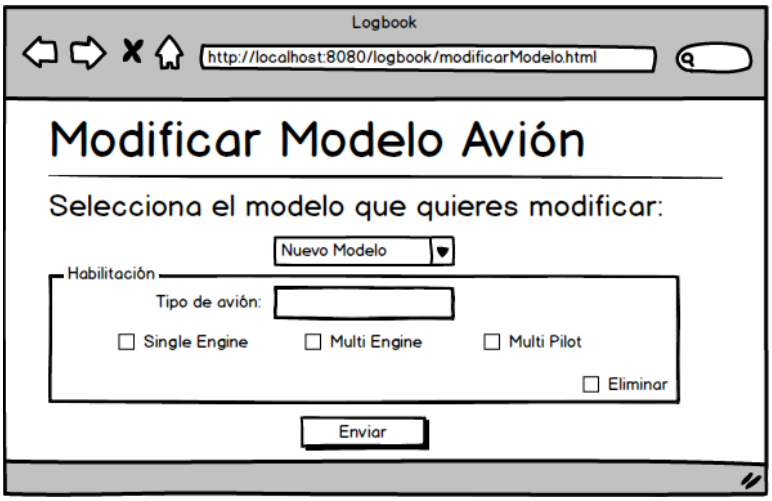


Figura 34: Mockup modificarModelo.html.

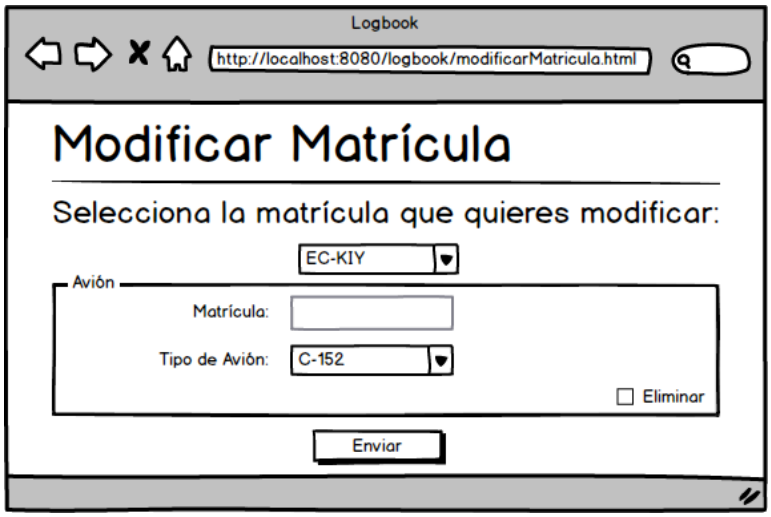


Figura 35: Mockup modificarMatricula.html.

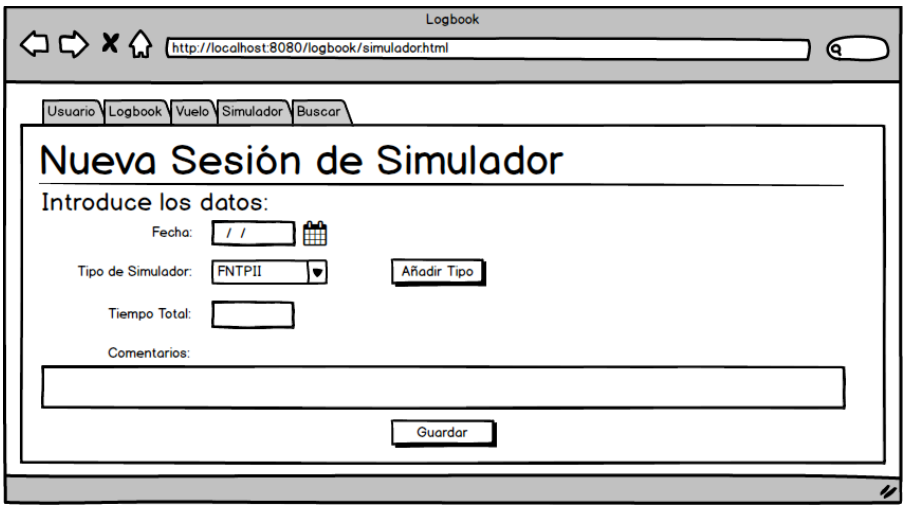


Figura 36: Mockup simulador.html.

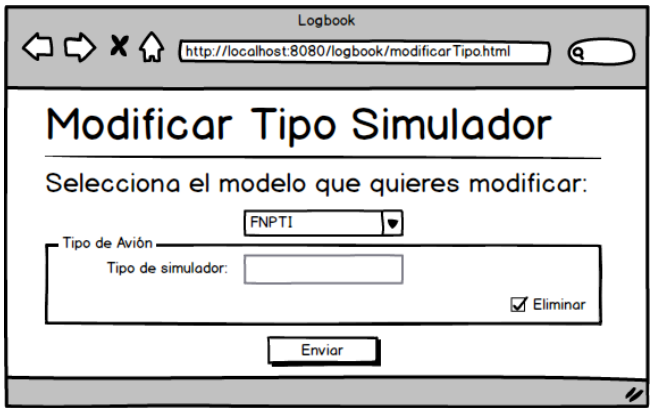


Figura 37: Mockup modificarTipo.html.

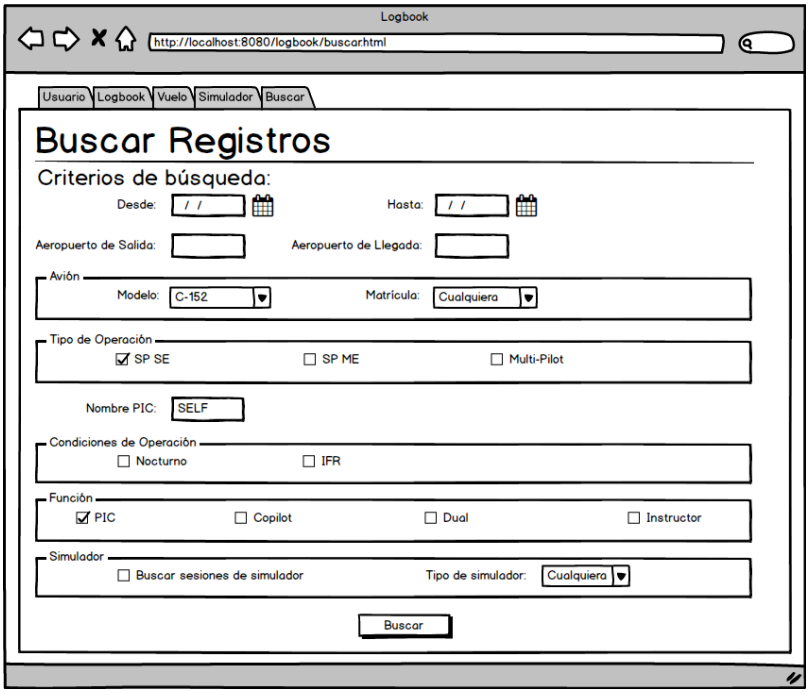


Figura 38: Mockup buscar.html.

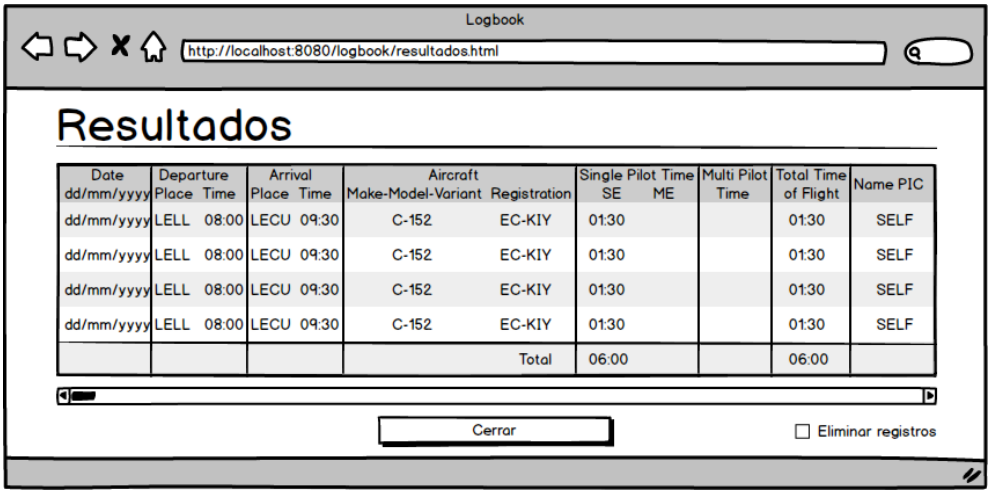


Figura 39: Mockup resultados.html

5.3. Diseño de la Base de Datos.

Utilizaremos la metodología descrita en el apartado 3.2 de esta memoria. Por lo que, para esta fase de diseño de la capa de persistencia de nuestra aplicación, seguiremos tres pasos: diseño conceptual, diseño lógico, incluyendo la normalización y, por último, diseño físico.

5.3.1. Diseño Conceptual.

El diseño conceptual es la fase del diseño de una base de datos cuyo objetivo es "obtener una representación de la realidad que capture las propiedades estáticas y dinámicas de la misma, que son necesarias para satisfacer los requisitos; esta representación debe suponer una imagen fiel del comportamiento del mundo real".

5.3.1.1. Propiedades Estáticas.

A) Modelo Entidad-Relación.

Los modelos de datos conceptuales son las herramientas que se utilizan para realizar este diseño. En el caso que nos ocupa, usaremos el modelo Entidad-Relación, una de las muchas propuestas que se han hecho sobre el modelo original de Peter Pin-Shan Chen [5]. El modelo Entidad Relación, en adelante ER, permite representar, en el llamado diagrama ER, las estructuras que constituyen el contenido del sistema de información junto con restricciones de distintos tipos que limitan las ocurrencias válidas de las mismas. Para ello hace uso, fundamentalmente, de tres conceptos: entidad, atributo y relación.

- Entidad: La observación de la realidad permite detectar el conjunto de objetos, físicos o conceptuales, de los que se quiere almacenar información para, mediante el uso de la clasificación, uno de los mecanismos de abstracción más primario que existe, descubrir el conjunto de clases o tipos de objetos que son de interés, permitiendo no prestar atención a las ocurrencias concretas sino al conjunto de ocurrencias. En nuestro caso podemos identificar pilotos, habilitaciones, vuelos, aviones, tipos de aviones, simuladores y tipos de simuladores.
- Relación: Los objetos de nuestro sistema de información se asocian unos con otros, siendo también de interés modelar estas conexiones. Para ello se utilizarán los tipos de relaciones. Con estas se representarán las posibles asociaciones existentes entre los objetos de nuestro logbook. Cada ocurrencia de una relación asociará una ocurrencia de cada uno de los objetos relacionados.

- Atributos: nos van a permitir representar las propiedades de los objetos de nuestro sistema de información así como las asociaciones entre ellos. Estos pueden ser simples o compuestos, monovaluados o multivaluados, básicos o derivados; y tener restricciones de dominio, de valor no nulo, unicidad o identificación.

B) Modelo Chen.

Basado en el modelo original de Peter Pin-Shan Chen [5], su mayor ventaja es la gran capacidad de expresión de la realidad. Ya que nuestras Entidades tienen numerosos atributos, por sencillez, se han representado únicamente las claves primarias, los atributos únicos y los atributos con restricción de valor no nulo. Realizado con la herramienta online para representar diagramas de flujo Gliffy[29].

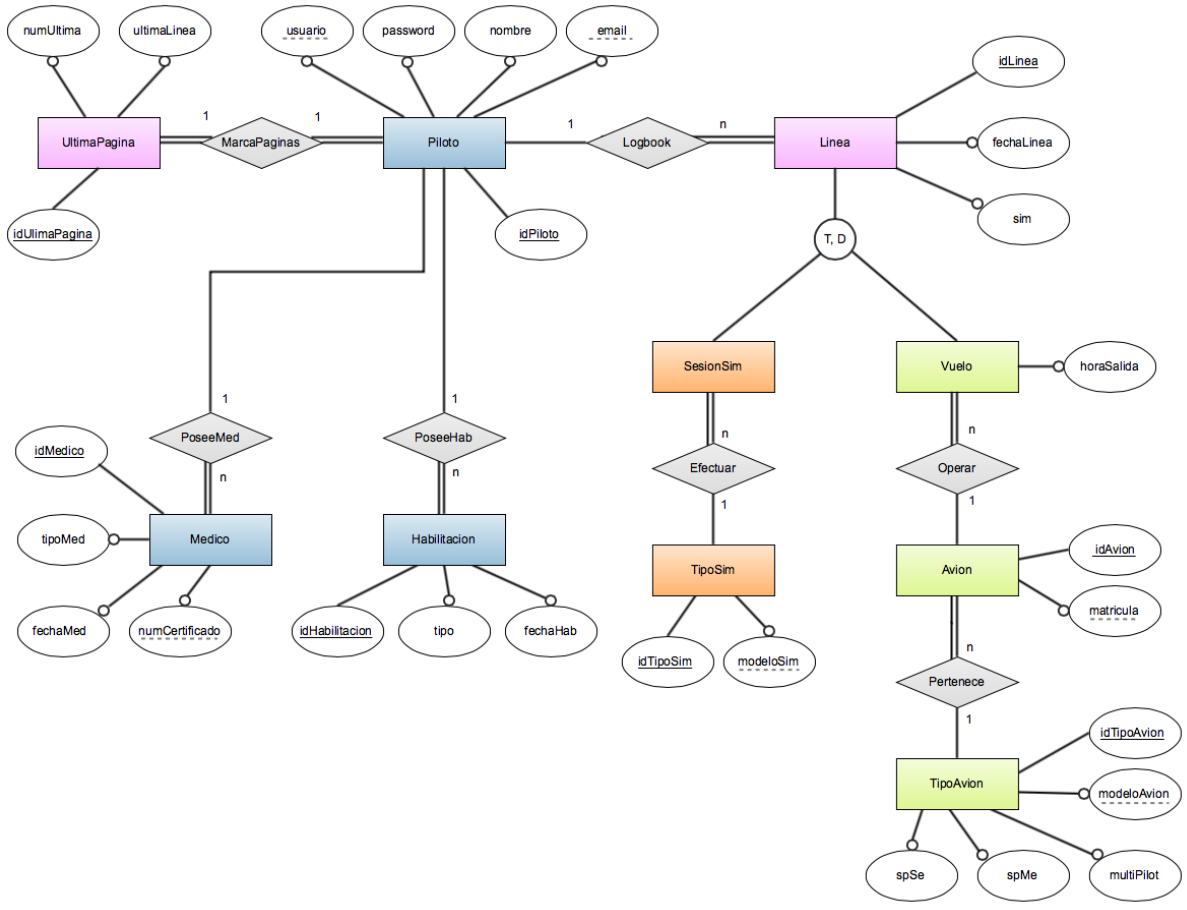


Fig 40: Modelo Entidad Relación.

[29] https://www.gliffy.com

C) Modelo Pata de Gallo.

Desarrollado por Richard Barker, entre otros. Éste lo implantó cuando empezó a trabajar para Oracle. Por ello, aún hoy en día, es el utilizado por la herramienta MySQLWorkbench[30] y otras muchas herramientas CASE[31]. Las ventajas frente al modelo Chen es una mejor lectura y aprovechamiento del espacio de dibujo. Además, usando MySQL Workbench, se crean automáticamente las claves ajenas y se pueden definir los tipos de datos, adelantando la tarea posterior.

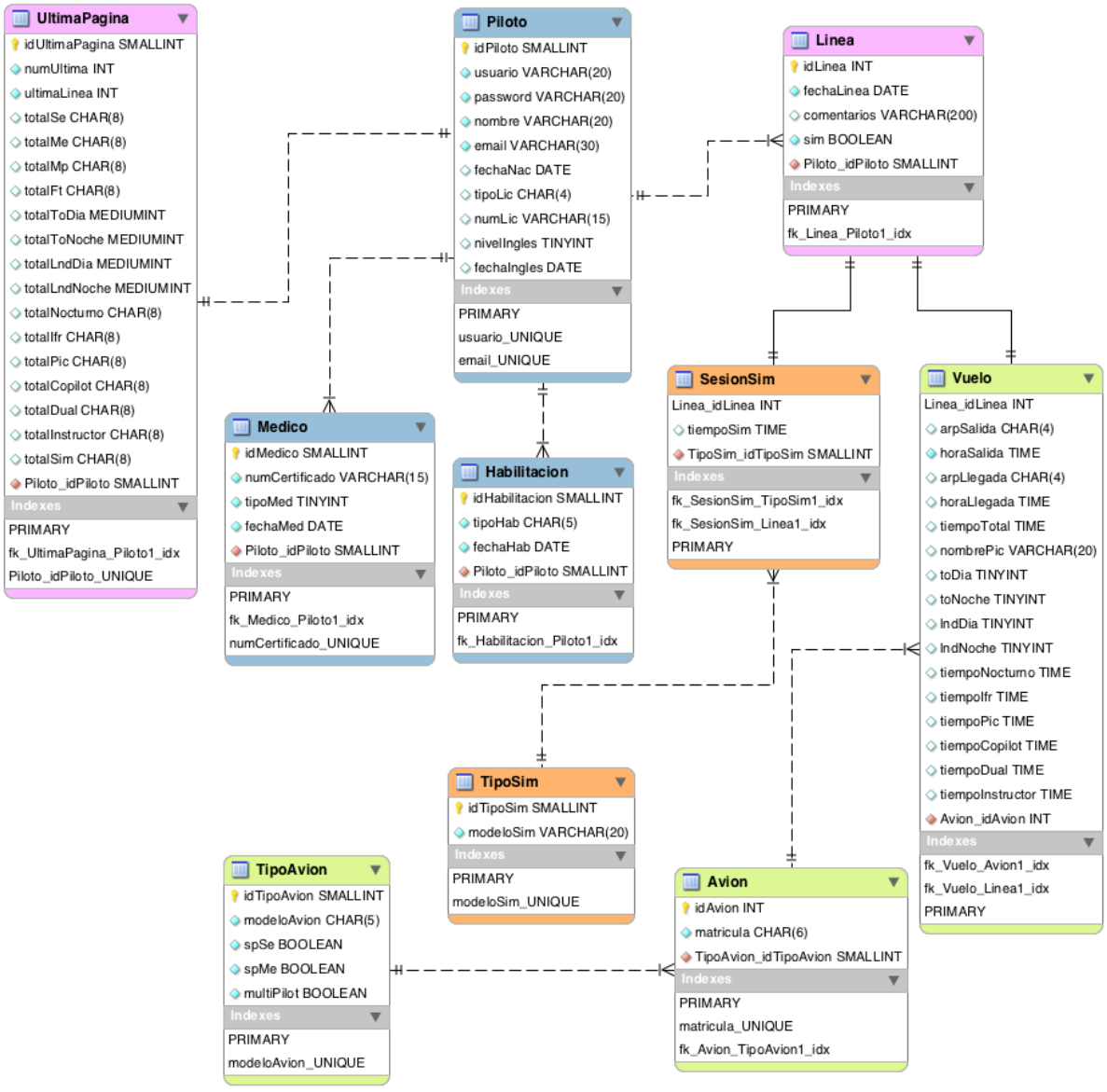


Fig 41: Modelo Pata de Gallo con Workbench.

[30] http://dev.mysql.com/downloads/workbench/

[31] Computer Aided Software Engineering.

Para la elección del tipo de datos se han seguido los siguientes criterios:

- **Datos numéricos:** Se han sobreestimado los rangos necesarios para no tener problemas en caso de numerosos usuarios. Aún así se han declarado todos los atributos posibles como SMALLINT o MEDIUMINT, en vez de optar por defecto por INT sin tener en cuenta los valores.
- **Cadenas de caractéres:** Para las cadenas cortras se ha usado CHAR y un valor de longitud definido teniendo en cuenta la cadena más larga posible. Se ha tomado esta decisión porque VARCHAR necesita un byte más para guardar la longitud y además es más fácil tratar con cadenas de longitud definida. Al ser cadenas cortas se desestima la posible eficiencia de VARCHAR en pos de las ventajas comentadas. Sin embargo en las cadenas largas sí se opta por el ahorro de espacio y se definen como VARCHAR.
- **Fechas:** Las fechas en mySQL tienen un formato YYYY/MM/DD. Habrá que tener en cuenta si las fechas en Java tienen ese mismo formato. En cualquier caso es probable que el usuario las introduzca como DD/MM/YYYY, por lo que habrá que implementar un algoritmo para hacer estas conversiones.
- **Tiempo:** Se representa en mySQL como HH:MM:SS. Su valor máximo es 838:59:59. En la tabla última página se guardan los totales de horas de un piloto. Dado que un piloto puede llegar a realizar hasta 40.000 horas a lo largo de su carrera profesional se ha optado por guardar estos totales en forma de cadena de ocho caracteres. Ejemplo: 40000:30. Implementaremos un algoritmo en Java que trabaje con String para transformar esta cadena en números y hacer los cálculos necesarios.

5.3.1.2. Propiedades Dinámicas.

A) Descripción de Transacciones.

"Una transacción es una secuencia de operaciones de acceso a los datos que constituye una unidad lógica de ejecución" [1]. Un correcto procesamiento de las transacciones debe asegurar las propiedades de atomicidad, consistencia, persistencia y aislamiento.

Para modelar la evolución dinámica del sistema hay que diseñar un conjunto de transacciones que permitan actualizar la información almacenada. Esta actualización se consigue añadiendo información (con la inserción de ocurrencias de entidad o relación) o modificando la información ya almacenada (con la modificación del valor de los atributos de ocurrencias de entidad o relación). Se asume por tanto que para cada tipo de entidad, de relación o de especialización existe una operación de inserción, una de borrado y una de modificación.

Se propone una metodología de diseño de transacciones que consiste en obtener un conjunto de transacciones, a partir del diagrama ER, a las que se denomina transacciones mínimas. Este conjunto incluirá, para cada entidad y para cada relación del diagrama una transacción de inserción, una de borrado y varias de modificación[32], aunque hay algunas excepciones.

Las transacciones mínimas deben considerarse como la única forma de modificar la información almacenada y podrán ser utilizadas en transacciones más complejas diseñadas a medida de la aplicación.
Para simplificar el diseño, asumiremos que todas las restricciones expresadas en el diagrama son controladas por el sistema por lo que no es necesario comprobarlas en las propias transacciones; sin embargo, deben de tenerse en cuenta o podrían definirse transacciones que siempre serían abortadas al conducir al sistema a un estado en el que se violase alguna restricción.

I) Inserción.

- Entidades: si una entidad tiene restricción de existencia en alguna relación, hay que insertar en esa relación. Si otras entidades que participan en la relación también tienen restricción de existencia, habrá que diseñar una transacción que incluya la inserción en la relación y también en todas las entidades con restricción de existencia. Dependiendo de las cardialidades máximas, esta transacción será única o no.
- Relaciones: Si la relación R en la que se está insertando participa como objeto agregado con restricción de existencia en otra relación S, hay que insertar en S. En nuestro diagrama no hay agregaciones, por lo que únicamente insertaremos en R.
- Para la inserción en la generalización "Linea", que es total y disjunta, habrá que diseñar tantas transacciones como entidades especializadas haya; cada transacción insertará en la entidad general y en una de las especializadas.

- Transacción para insertar en UltimaPagina.
  ⇒ Insertar en UltimaPagina.
  ⇒ Insertar en Piloto.
  ⇒ Insertar en MarcaPaginas.
- Transacción para insertar en Piloto.
  ⇒ Insertar en Piloto.

[32] Las transacciones para modificar el valor de atributos de ocurrencias de entidad o relación son triviales y se podría considerar que existe una por cada atributo que permitiría cambiar el valor "viejo" por uno "nuevo". Transacciones de modificación más complicadas dependerán del caso concreto que se esté considerando.

- Transacción para insertar en Linea tipo Vuelo.
    - ⇒ Insertar en Linea.
    - ⇒ Insertar en Vuelo.
        - ⇒ Insertar en Operar.
        - ⇒ Insertar en Logbook.
- Transacción para insertar en Linea tipo SesionSim.
    - ⇒ Insertar en Linea
    - ⇒ Insertar en SesionSim.
        - ⇒ Insertar en Efectuar.
        - ⇒ Insertar en Logbook.
- Transacción para insertar en Avion.
    - ⇒ Insertar en Avion.
        - ⇒ Insertar en Pertenece.
- Transacción para insertar en TipoAvión.
    - ⇒ Insertar en TipoAvión.
- Transacción para insertar en TipoSim.
    - ⇒ Insertar en TipoSim.
- Transacción para insertar en Medico.
    - ⇒ Insertar en Medico.
        - ⇒ Insertar en PoseeMed.
- Transacción para insertar en Habilitacion.
    - ⇒ Insertar en Habilitacion.
        - ⇒ Insertar en PoseeHab.
- Transacción para insertar en MarcaPáginas.
    - ⇒ Insertar en MarcaPáginas
- Transacción para insertar en Logbook.
    - ⇒ Insertar en Logbook.
- Transacción para insertar en Efectuar.
    - ⇒ Insertar en Efectuar.
- Transacción para insertar en Operar.
    - ⇒ Insertar en Operar.
- Transacción para insertar en Pertenece.
    - ⇒ Insertar en Pertenece.
- Transacción para insertar en PoseeMed.
    - ⇒ Insertar en PoseeMed.
- Transacción para insertar en PoseeHab.
    - ⇒ Insertar en PoseeHab.

II) Borrado.

- Entidades: Para el borrado de Entidades, se considerarán dos operaciones: restrictivo y en cascada.
    - Restrictivo respecto a una relación R: sólo incluye la operación de borrado de una entidad. Como el sistema controla que se cumplan las restricciones, cuando se ejecute dicha transacción, si la ocurrencia a borrar está participando en R, no podrá realizarse el borrado.
    - En cascada respecto a una relación R: se incluye, además de la operación de borrado de la entidad, el borrado de las ocurrencias de R en las que interviene la entidad. Si la entidad tiene restricción de existencia respecto a R, el borrado siempre tendrá que ser de este tipo.
- Relaciones: Como en el caso de las Entidades, también encontramos dos tipos:
    - Restrictivo respecto a una entidad o relación: no hay que añadir ninguna operación.
    - En cascada: si alguna de las entidades que participan en la relación tienen restricción de existencia en ella, hay que incluir, además del borrado de la relación, el borrado de dicha entidad para los casos en los que la ocurrencia de relación borrada sea la última en la que participaba la entidad.
- Para la generalización, total y disjunta, habrá que diseñar una transacción que debe borrar de la entidad general y de todas las especializadas aunque, al ser disjunta, realmente sólo se conseguirá borrar de una de ellas.

- Transacción para borrar en ÚltimaPágina.
    ⇒ Borrar en ÚltimaPágina.
        ⇒ Borrar en MarcaPáginas (cascada)
- Transacción para borrar en Piloto.
    ⇒ Borrar en Piloto.
    ⇒ Borrar en ÚltimaPágina (cascada).
    ⇒ Borrar en Línea (cascada).
    ⇒ Borrar en Médico (cascada).
    ⇒ Borrar en Habilitación (cascada).
        ⇒ Borrar en MarcaPáginas.
        ⇒ Borrar en Logbook.
        ⇒ Borrar en PoseeMed.
        ⇒ Borrar en PoseeHab.
- Transacción para borrar en Línea.
    ⇒ Borrar en Línea (cascada).

⇒ Borrar en SesiónSim.

⇒ Borrar en Vuelo.

⇒ Borrar en Logbook.

- Transacción para borrar en TipoSim.

⇒ Borrar en TipoSim (restrictivo respecto a SesiónSim).

- Transacción para borrar en Avión.

⇒ Borrar en Avión (restrictivo respecto a Vuelo).

- Transacción para borrar en TipoAvión.

⇒ Borrar en TipoAvión (restrictivo respecto a Avión).

- Transacción para borrar en Médico.

⇒ Borrar en Médico (cascada).

⇒ Borrar en PoseeMed.

- Transacción para borrar en Habilitación.

⇒ Borrar en Habilitación (cascada).

⇒ Borrar en PoseeHab.

- Transacción para borrar en MarcaPáginas.

⇒ Borrar en MarcaPáginas.

- Transacción para borrar en Logbook.

⇒ Borrar en Logbook.

- Transacción para borrar en Efectuar.

⇒ Borrar en Efectuar.

- Transacción para borrar en Operar.

⇒ Borrar en Operar.

- Transacción para borrar en Pertenece.

⇒ Borrar en Pertenece.

- Transacción para borrar en PoseeMed.

⇒ Borrar en PoseeMed.

- Transacción para borrar en PoseeHab.

⇒ Borrar en PoseeHab.

III) Modificación.

Dependiendo de las necesidades del sistema de información, puede haber muchas transacciones de modificación diferentes. Deben diseñarse las transacciones que posibiliten modificar cualquier atributo de cualquier objeto que sea necesario.

5.3.2. Diseño Lógico.

El diseño lógico consiste en la transformación del esquema conceptual, que se encuentra descrito con un cierto modelo de datos, en este caso el modelo relacional, en estructuras y transacciones descritas en términos del modelo de datos en el cual se base el sistema de gestión de bases de datos que se vaya a utilizar. Este proceso se dividirá en dos fases:

1. Transformación de los aspectos estáticos del esquema conceptual: para ellos se verá cómo es posible transformar cada una de las estructuras de un diagrama entidad-relación, modelo utilizado en el deseó conceptual, en relaciones. Esta transformación dará lugar a un primer esquema relacional. Posteriormente aplicaremos la teoría de la normalización, que permite refinar este primer esquema relaciona obteniendo un esquema relacional adecuado desde el punto de vista de su manipulación.
2. Transformación de los aspectos dinámicos del esquema conceptual: para ello se verá cómo se obtienen transacciones sobre las relaciones del esquema relacional a partir del análisis de transacciones realizado en la fase de diseño conceptual.

5.3.2.1. Transformación de los aspectos estáticos.

A) Esquema relacional.

La relación UltimaPagina - Piloto es de cardinalidad 1:1 con dos restricciones de existencia. Una de las soluciones sería establecer una única relación (MarcaPaginas) con los atributos de ambas y con idUltimaPagina como clave primaria y idPiloto como único y valor no nulo. Dado que de esta manera no es posible representar independientemente las dos entidades, se complica la manipulación de los objetos representados en Piloto. Por esto usaremos preferiblemente esta otra solución, aunque tenga más relaciones y restricciones de integridad:

```
UltimaPagina(idUltimaPagina: SMALLINT, numUltima: SMALLINT, ultimaLinea:
INT totalSe: CHAR 8, totalMe: CHAR 8, totalMp: CHAR 8, totalFt: CHAR 8,
totalToDia: MEDIUMINT, totalToNoche: MEDIUMINT, totalLndDia: MEDIUMINT,
totalLndNoche: MEDIUMINT, totalNocturno: CHAR 8, totalIfr: CHAR 8,
totalPic: CHAR 8, totalCopilot: CHAR 8, totalDual: CHAR 8,
totalInstructor: CHAR 8, totalSim: CHAR 8, idPiloto: SMALLINT )
      Clave Primaria: {idUltimaPagina}
      Único: {idPiloto}
      Valor No Nulo: {idPiloto, numUltima, ultimaLinea}
      Clave Ajena: {idPiloto} hace referencia a Piloto
```

La restricción de existencia de la entidad Piloto con respecto a la relación MarcaPaginas puede representarse con la siguiente expresión:

```
CREATE ASSERTION RestriccionExistencia1 CHECK
      NOT EXISTS (
            SELECT *FROM Piloto p
                  WHERE NOT EXISTS (
                        SELECT * FROM UltimaPagina u
                        WHERE p.idPiloto = u.idPiloto ))
```

```
Piloto(idPiloto: SMALLINT, usuario: VARCHAR 20, password: VARCHAR 20,
nombre: VARCHAR 20, email: VARCHAR 30, fechaNac: DATE, tipoLic CHAR 4,
numLic: VARCHAR 15, nivelIngles: TINYINT, fechaIngles: DATE)
      Clave Primaria: {idPiloto}
      Único: {usuario, email}
      Valor No Nulo: {usuario, password, nombre, email}
```

```
Habilitacion(idHabilitacion: SMALLINT, tipoHab: CHAR 5, fechaHab: DATE,
idPiloto: SMALLINT)
      Clave Primaria: {idHabilitacion}
      Clave Ajena: {idPiloto} hace referencia a Piloto
      Valor No Nulo: {idPiloto, tipoHab, fechaHab}
```

```
Medico(idMedico: SMALLINT, numCertificado: VARCHAR 15, tipoMed: TINYINT,
fechaMed: DATE, idPiloto: SMALLINT)
      Clave Primaria: {idMedico}
      Clave Ajena: {idPiloto} hace referencia a Piloto
      Único: {numCertificado}
      Valor No Nulo: {idPiloto, tipoMed, fechaMed, numCertificado}
```

Sólo cuando una especialización es parcial y solapada existe una transformación en relaciones totalmente adecuada; en los demás casos es necesario incluir ciertas restricciones de integridad que permitan la definición exacta de estos objetos. La transformación consiste en definir una relación para cada entidad especializada que incluye los atributos propios y también la clave primaria de la relación que representa a la entidad general, que pasa a ser también la clave primaria de la relación. Al ser total y disjunta es necesaria la inclusión de dos restricciones para obtener un esquema relaciona que represente exactamente este tipo tipo de objeto general.

```
Linea(idLinea: INT, fechaLinea: DATE, comentarios: VARCHAR 200,
sim: BOOLEAN)
```

Clave Primaria: {idLinea}

Clave Ajena: {idPiloto} hace referencia a Piloto

Valor no Nulo: {fechaLinea, idPiloto, sim}

```
CREATE ASSERTION Total CHECK
      NOT EXISTS (
            SELECT l.idLinea FROM Linea l
            WHERE l.idLinea NOT IN(
                  SELECT v.idLinea FROM Vuelo v UNION
                  SELECT s.idLinea FROM SesionSim s ))
```

```
CREATE ASSERTION Disjunta CHECK
      NOT EXISTS (      SELECT * FROM Vuelo v, SesionSim s
                        WHERE v.idLinea = s.idLinea )
```

Vuelo(idLinea: INT, arpSalida: CHAR 4, horaSalida: TIME, arpLlegada: CHAR 4, horaLlegada: TIME, tiempoTotal: TIME, nombrePic: VARCHAR 20, toDia: TINYINT, toNoche TINYINT, lndDia: TINYINT, lndNoche: TINYINT, tiempoNocturno: TIME, tiempoIfr: TIME, tiempoPic: TIME, tiempoCopilot: TIME, tiempoDual: TIME, tiempoInstructor: TIME, idAvion: INT)

Clave Primaria: {idLinea}

Clave Ajena: {idLinea} hace referencia a Linea

Clave Ajena: {idAvion} hace referencia a Avion

Valor No Nulo: {idAvion, horaSalida}

Avion(idAvion: INT, matricula: CHAR 6, idTipoAvion: SMALLINT)

Clave Primaria: {idAvion}

Clave Ajena: {idTipoAvion} hace referencia a TipoAvion

Único: {matricula}

Valor No Nulo: {idTipoAvion, matricula}

TipoAvion(idTipoAvion: SMALLINT, modeloAvion: CHAR 5, spSe: BOOLEAN, spMe: BOOLEAN, multiPilot: BOOLEAN)

Clave Primaria: {idTipoAvion}

Único: {modeloAvion}

Valor No Nulo: {modeloAvion, spSe, spMe, multiPilot}

Sesion_Sim(idLinea: INT, tiempoSim: TIME, idTipoSim: SMALLINT)

Clave Primaria: {idLinea}

Clave Ajena: {idLinea} hace referencia a Linea

Clave Ajena: {idTipoSim} hace referencia a TipoSim

Valor No Nulo: {idTipoSim}

```
TipoSim(idTipoSim SMALLINT: modeloSim: VARCHAR 20)
      Clave Primaria: {idTipoSim}
      Único: {modeloSim}
      Valor No Nulo: {modeloSim}
```

B) Normalización.

El esquema relacional obtenido con las transformaciones presentadas debe ser revisado para comprobar que se encuentra adecuadamente diseñado. Para ello se utilizarán los conceptos de la teoría de la normalización.

I) Primera forma normal - 1FN.

> Una relación R está en 1FN si sus atributos sólo pueden tomar valores atómicos, es decir, simples e indivisibles.

Nuestro esquema relacional no posee ningún atributo con dominios de este tipo, por lo que podemos decir que está en 1FN. Esto se debe a que la 1FN se considera hoy en día parte de la definición formal de relación, ya que los dominios asociados a los atributos deben ser simples.

II) Segunda forma normal con una sola clave - 2FN

> Una relación R está en 2FN si está en 1FN y todo atributo no-primo depende funcionalmente de forma completa de la clave primaria de R.

Toda relación cuya clave tenga sólo un atributo en su clave primaria está en segunda forma normal, por lo que podemos decir que nuestro esquema relacional está en 2FN desde la fase del diseño conceptual.

III) Tercera forma normal con una sola clave - 3FN.

> Una relación R está en 3FN si está en 2FN y ningún atributo no-primo depende funcionalmente de forma transitiva de la clave primaria.

Ya que hemos decidido usar claves primarias autoincrementadas para facilitar las operaciones de comparación, a lo largo de nuestro esquema relacional podemos encontrar atributos como usuario, numCertificado, matrícula, modeloAvión y modeloSim que hubieran sido las claves primarias de nuestras entidades. Por lo tanto,

```
        String totalMp = rs2.getString("totalMp");
        String totalFt = rs2.getString("totalFt");
        int totalToDia = rs2.getInt("totalToDia");
        int totalToNoche = rs2.getInt("totalToNoche");
        int totalLndDia = rs2.getInt("totalLndDia");
        int totalLndNoche = rs2.getInt("totalLndNoche");
        String totalNocturno = rs2.getString("totalNocturno");
        String totalIfr = rs2.getString("totalIfr");
        String totalPic = rs2.getString("totalPic");
        String totalCopilot = rs2.getString("totalCopilot");
        String totalDual = rs2.getString("totalDual");
        String totalInstructor = rs2.getString("totalInstructor");
        String totalSim = rs2.getString("totalSim");
        UltimaPagina u = new UltimaPagina (idUltimaPagina, numUltima, ultimaLinea,
                        totalSe, totalMe, totalMp, totalFt, totalToDia, totalToNoche,
                        totalLndDia, totalLndNoche, totalNocturno, totalIfr,
totalPic,
                        totalCopilot, totalDual, totalInstructor, totalSim);
        // Asignamos la última página al logbook del piloto y viceversa
        log.asignarUltimaPagina(u);
        u.asignarLogbook(log);

        // Leer certificados médicos de la base de datos
        String SSQL3 = "SELECT * FROM Medico WHERE Piloto_idPiloto=" +
               p.obtenerIdPiloto();
            ResultSet rs3 = BaseDatos.consultarBD(SSQL3);
            while (rs3.next()) {
               int idMedico = rs3.getShort("idMedico");
               String numCertificado = rs3.getString("numCertificado");
               int tipoMed = rs3.getByte("tipoMed");
               Date fechaMed = rs3.getDate("fechaMed");
               Medico m = new Medico(idMedico, numCertificado, tipoMed, fechaMed);
               // Asignamos Médico a Piloto y viceversa
               p.insertarMedico(m, tipoMed);
               m.asignarPiloto(p);
            }
            rs3.close();

        // Leer habilitaciones de la base de la base de datos
        String SSQL4 = "SELECT * FROM Habilitacion WHERE Piloto_idPiloto=" +
               p.obtenerIdPiloto();
        ResultSet rs4 = BaseDatos.consultarBD(SSQL4);
        while (rs4.next()) {
            int idHabilitacion = rs4.getShort("idHabilitacion");
            String tipoHab = rs4.getString("tipoHab");
            Date fechaHab = rs4.getDate("fechaHab");
            Habilitacion h = new Habilitacion(idHabilitacion, tipoHab, fechaHab);
               // Asignamos Habilitacion a Piloto y viceversa
               p.insertarHabilitacion(h);
               h.asignarPiloto(p);
            }
            rs4.close();

            // Guardar usuario en la sesión.
               HttpSession session = request.getSession();
               session.setAttribute("usuario", p);
        }
```

podemos decir que todos los atributos de las entidades dependen funcionalmente de ellos. Para que dejen de ser atributos no-primos aplicamos una restricción de unicidad, convirtiéndolos de esta manera en claves. Con esto ya podemos decir que nuestro esquema está en 3FN.

Con sólo este sencillo paso tenemos la base de datos normalizada. El modelo conceptual presentado en la figura x es la quinta versión realizada del modelado de la realidad, refinado en cada versión. Podemos afirmar por tanto, que dedicar tiempo y esfuerzo a realizar un buen modelo conceptual desde un principio hace que nuestro esquema resulte robusto y nos facilita el trabajo en fases posteriores.

5.3.2.2. Transformación de los aspectos dinámicos.

A) Representación de las transacciones.

El método que se seguirá para la transformación de un esquema de transacción, definido en el esquema conceptual, a una transacción sobre el esquema lógico se realizará en tres pasos:

1. Investigar la transacción para concretar sobre qué objetos, entidades o relaciones, actúa y con que operación: inserción, borrado o modificación).
2. Determinar en qué relaciones del esquema relaciona se encuentran representados esos objetos y qué operación debe utilizarse.
3. Analizar el esquema simplificado de transacciones del modelo conceptual y transformarlo en un conjunto de transacciones sobre las correspondientes relaciones del esquema relacional.

Haremos una transacción de alta y baja de un usuario como ejemplo, con pseudocódigo y SQL. Para la inserción únicamente tendremos que insertar en Piloto, como ya dijimos en el diseño de transacciones, mientras que para borrar un usuario deberemos de borrar su última página, todas sus líneas anotadas, su certificado médico y su habilitación.

```
TRANSACCIÓN altaPiloto (us: VARCHAR(20), pw: VARCHAR(20), nm:
VARCHAR(20), em: VARCHAR(30), fl: DATE, tl: CHAR(4), nl:
VARCHAR(15), ni: TINYINT, fi: DATE)
INICIO_TRANSACCIÓN
      INSERT INTO Piloto VALUES (us, pw, nm, em, fl, tl, nl, ni, fi)
FIN_TRANSACCIÓN
```

```
TRANSACCIÓN bajaPiloto (p: SMALLINT)
INICIO_TRANSACCIÓN
      DELETE FROM Piloto WHERE idPiloto = p
      DELETE FROM UltimaPagina WHERE idPiloto = p
      DELETE FROM Linea WHERE idPiloto = p
      DELETE FROM Medico WHERE idPiloto = p
      DELETE FROM Habilitacion WHERE idPiloto = p
FIN_TRANSACCIÓN
```

6. Desarrollo.

6.1. Carga de la Base de Datos.

Podemos hacer la carga de la base de datos siguiendo unos sencillos pasos con MySQLWorkbench. Abrimos el diseño de la base de datos de la figura x y vamos al menú `Database > Forward Engineer…` y se abrirá el asistente. En el primer paso, seleccionamos una conexión que habíamos creado previamente para probar las consultas que se realizarán a la base de datos. Para crear esta conexión simplemente hay que ir a `Home > MySQL Connections > +`.

El asistente da una gran cantidad de opciones de configuración, pero en este caso elegimos la configuración por defecto, siguiendo a través de las pantallas hasta que se crea la base de datos. Se puede encontrar el código MySQL para la creación del sistema de información en el Anexo B.

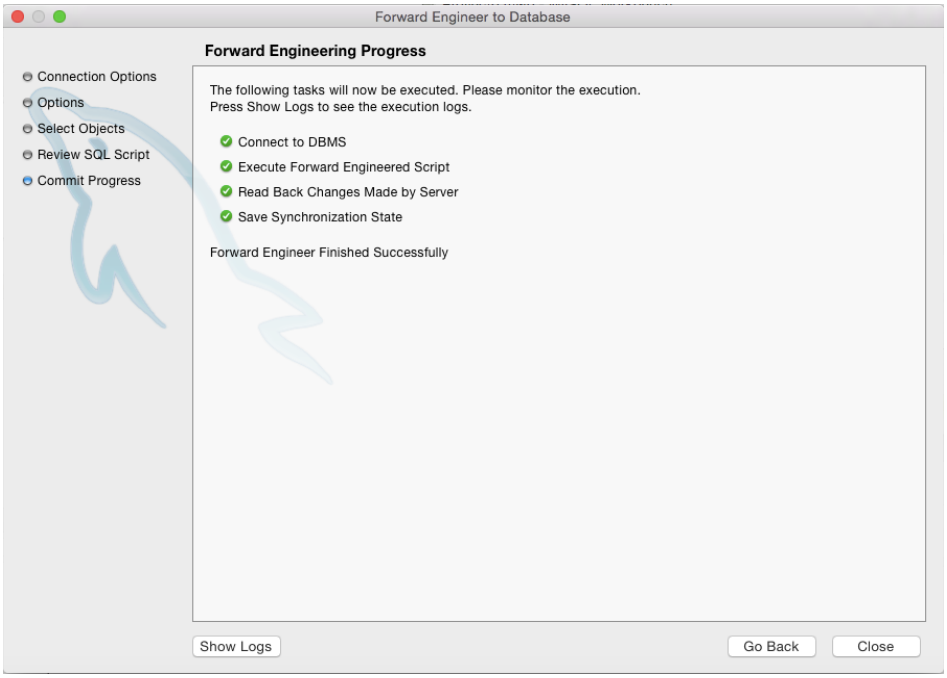


Figura 42: Resultado del Forward Engineer de MySQLWorkbench.

6.2. Creación de un Proyecto Web Dinámico.

Abrimos Eclipse y creamos un nuevo proyecto web dinámico: `File > New > Dynamic Web Project` y se abrirá el asistente. Seleccionamos como `Target runtime > Apache Tomcat v7.0` y nos aseguramos que en `Dynamic web module version` está seleccionado `3.0` y pulsamos `Next>`. En la siguiente pantalla podemos seleccionar el directorio en el que se guardará el código fuente. Lo dejamos por defecto en `src` y pulsamos `Next>`. En la última pantalla le ponemos un nombre a nuestro proyecto y no debemos de olvidarnos de marcamos `Generate web.xml deployment descriptor` y pulsamos `Finish`.

Haciendo clic con el botón derecho sobre el proyecto podemos ir creando los objetos del sistema de información seleccionando New > Class. Todas las clases pertenecerán al paquete logbook. Así pues, seguimos las guías en el Anexo A y creamos Piloto, Medico, Habilitación, Logbook, Pagina, UltimaPagina, Linea, Vuelo, SesionSim, Avion, TipoAvion y TipoSim.

Volviendo a hacer clic con el botón derecho sobre el proyecto, está vez seleccionamos `New > Servlet`. Llamaremos al servlet Controlador ya que se encargará precisamente de controlar las peticiones que llegan al servidor y responderlas de la forma deseada en cada caso.

6.3. Configuración de la Base de Datos en Eclipse.

Para poder realizar conexiones con el servlet a la base de datos MySQL que se ha creado en el el punto 5.1 debemos configurar Eclipse de la siguiente manera:

1. Descargar `mysql-connector-java-5.1.35.tar` de http
2. Descomprimir el archivo. En `docs > connector-j.pdf > Chapter 3 > Connector/J Installation` se pueden encontrar diferentes instrucciones para su instalación. En nuestro caso, movemos el archivo `mysql-connector-java-5.1.35-bin.jar` al directorio `WebContent/WEB-INF/lib` del proyecto en Eclipse.
3. Selecciona `Window > Open Perspective > Other > Database Development > OK` y se abrirá una vista de la base de datos, Database Development. Si quieres volver a la vista original selecciona Java EE en la parte superior derecha.
4. Botón derecho sobre `Database Connections > New > MySQL > Next> > MySQL JDBC Driver` y presionamos en el icono marcado en rojo en la siguiente figura:

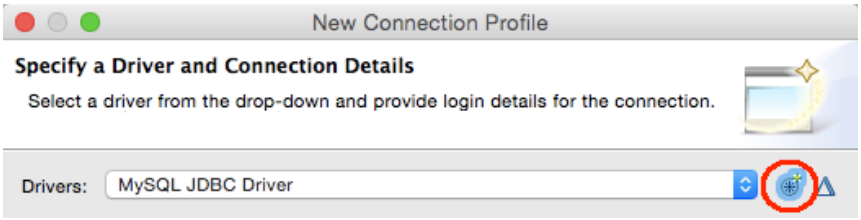


Figura 43: Configurar Driver MySQL en Eclipse.

5. En `Name/Type` seleccionamos `MySQL JDBC Driver – MySQL – 5.1.`
6. En la siguiente pestaña, `JAR List > Add JAR/Zip…` y agregamos `mysql-connector-java-5.1.35-bin.jar,` ubicado en `WEB-INF/lib`

Para acabar con los ajustes necesarios para trabajar con base de datos, vamos a agregar el Driver de MySQL a las librerías del proyecto.

7. Hacemos clic con el botón derecho sobre el proyecto y seleccionamos `Properties > Java Build Path > Libraries > Add External JARs`
8. Indicamos la ruta del archivo `mysql-connector-java-5.1.35-bin.jar` que colocamos en el paso 2 en `WEB-INF/lib.`
9. Comprobamos que el archivo aparece en `Libraries > Web App Libraries.`

Por último, ya en la implementación de la clase `BaseDatos`, leemos el driver específico de la base de datos que vamos a utilizar y, una vez lo tenemos cargado en memoria, tendremos que crear una conexión con el objeto de la clase **`Connection.`** En nuestro caso, será:

```
Class.forName("com.mysql.jdbc.Driver");
conn = DriverManager.getConnection("jdbc:mysql://127.0.0.1/mydb", "root", "");
```

6.4. El Servlet Controlador.

Como se ha explicado en el apartado 6.2, hemos creado en el proyecto web dinámico un Servlet llamado "Controlador". Para su correcto funcionamiento deberemos cambiar el fichero web.xml como sigue:

```
<servlet>
    <servlet-name>Controlador</servlet-name>
    <servlet-class>Controlador</servlet-class>
</servlet>
```

Creamos un método llamado ejecutarOpcion al que llamaremos tanto dentro del método POST, como en el GET. Recogeremos un parámetro de la petición llamado opción. Este parámetro será enviado como un campo oculto (hidden) en las páginas que utilicen formularios. Para las páginas que rellenamos de forma dinámica pero que no poseen formulario utilizaremos ?opcion=num al final de las direcciones html. El método ejecutarOpcion decidirá que es lo que se debe hacer en cada caso.

También guardaremos en el Servlet datos de la sesión del usuario, ya que en todo momento hay que acceder únicamente a su información y no es posible que vea la de otras usuarios, por tanto, la sesión nos es muy útil para no molestar al piloto introduciendo sus claves una y otra vez. Esto se hará durante la carga del sistema:

```
public void inicializarSistema (String us, HttpServletRequest request,
HttpServletResponse response)
                    throws IOException, SQLException, ServletException {

        // Obtener todos los datos del usuario de tabla Piloto
        String SSQL = "SELECT * FROM Piloto WHERE usuario='" + us + "'";
        ResultSet rs = BaseDatos.consultarBD(SSQL);

        // Guardar campos de la consulta y crear Piloto
        int idPiloto = rs.getShort("idPiloto");
        String usuario = rs.getString("usuario");
        String password = rs.getString("password");
        String nombre = rs.getString("nombre");
        String email = rs.getString("email");
        Date fechaNac = rs.getDate("fechaNac");
        String tipoLic = rs.getString("tipoLic");
        String numLic = rs.getString("numLic");
        int nivelIngles = rs.getByte("nivelIngles");
        Date fechaIngles = rs.getDate("fechaIngles");
        Piloto p = new Piloto (idPiloto, usuario, password, nombre, email,
                    fechaNac, tipoLic, numLic, nivelIngles, fechaIngles);
        rs.close();

        // Inicializar su Logbook vacío y asignar
        Logbook log = new Logbook();
        log.asignarPiloto(p);
        p.asignarLogbook(log);

        // Recuperar su última página
        String SSQL2 = "SELECT * FROM UltimaLinea WHERE Piloto_idPiloto=" +
                    p.obtenerIdPiloto();
        ResultSet rs2 = BaseDatos.consultarBD(SSQL2);

        // Guardar campos de la consulta y crear UltimaPagina
        int idUltimaPagina = rs2.getShort("idUltimaPagina");
        int numUltima = rs2.getInt("numUltima");
        int ultimaLinea = rs2.getInt("ultimaLinea");
        String totalSe = rs2.getString("totalSe");
        String totalMe = rs2.getString("totalMe");
```

6.5. La clase BaseDatos.

La clase de datos contienen métodos genérico para insertar, modificar, borrar y consultar de la base de datos, así como un método de conexión con la misma. Además, contiene el método de login:

```
public static int loginBD (String us, String pw) throws java.sql.SQLException {
	int login = 0;
	Connection conn = null;
	conn = conectarBD();
	if (conn == null)
		return login;
	String SSQL = "SELECT * FROM Piloto WHERE usuario='" + us + "' AND
			password='" + pw + "'";
	String SSQL2 = "SELECT * FROM Piloto WHERE usuario='" +us+ "'";
	Statement st = conn.createStatement();
	ResultSet rs = st.executeQuery(SSQL);
	if (rs.next()) {
		login = 3;
	} else {
		rs = st.executeQuery(SSQL2);
		if (rs.next()) {
			login = 2;
		} else login = 1;
	}
	return login;
}
```

Otro aspecto a tener en cuenta al trabajar con la base de datos será cual es el método apropiado para obtener los diferentes campos de uno o varios registros, obtenidos como un objeto **ResultSet** tras realizar una consulta. Usaremos la siguiente tabla:

Tipo de dato SQL	Tipo de dato Java devuelto por getObject ( )	Método get apropiado
BOOLEAN	Boolean	boolean getBoolean ( )
CHAR	String	String getString ( )
DATE	java.sql.Date	java.sql.Date getDate ( )
INTEGER	Integer	int getInt ( )
MEDIUMINT	Integer	int getInt ( )
SMALLINT	Integer	int getShort ( )
TIME	java.sql.Time	java.sql.Time getTime ( )
TINYINT	Integer	byte getByte ( )
VARCHAR	String	String getString ( )

Para el cifrado de la contraseña podemos utilizar las funciones proporcionadas por MySQL `AES_ENCRYPT(password, key_str)` al insertar en la base de datos y `AES_DECPRIPT(key_str)` al recuperar el atributo.

6.6. La clase Utilidades.

En la clase utilidades encontramos algoritmos que trabajan con el tiempo. Usamos las clases java.sql.Date y java.sql.Time, tipo en el que están almacenados los tiempos y fechas en la base de datos. Para poder operar con ellos los transformaremos a las clases LocalTime y LocalDate. Cuando finalicemos las operaciones los transformaremos de nuevo en tipo SQL.

```
public java.sql.Time tiempoVuelo (String to, String lnd) {
              // Pasamos los Strings a LocalTime
              java.time.LocalTime toT = LocalTime.parse(to);
              java.time.LocalTime lndT = LocalTime.parse(lnd);
              // Preparamos las horas y los minutos de tot a restar
              int horas = toT.getHour();
              int mins = toT.getMinute();
              // Restamos...
              java.time.LocalTime res = lndT.minusHours(horas).minusMinutes(mins);
              // Convertimos a formato sql para poder guardar en BD
              java.sql.Time resultado = java.sql.Time.valueOf(res);
              return resultado;
       }
```

```
public static java.sql.Time sumarTiempos(java.sql.Time a, java.sql.Time b) {
              // Pasamos los tiempos sql a  String
              String aS = a.toString();
              String bS = b.toString();
              // Y de String a LocalTime
              java.time.LocalTime aT = LocalTime.parse(aS);
              java.time.LocalTime bT = LocalTime.parse(bS);
              // Preparamos las horas y los minutos de bT
              int horas = bT.getHour();
              int mins = bT.getMinute();
              // Sumamos...
              java.time.LocalTime res = aT.plusHours(horas).plusMinutes(mins);
              // Convertimos a formato sql
              java.sql.Time resultado = java.sql.Time.valueOf(res);
              return resultado;
       }
```

```
public boolean restarDias(java.sql.Date fechaExpira, int dias) {
		boolean expira = false;
		// Pasamos el formato de Date de sql a LocalDate
		java.time.LocalDate f = fechaExpira.toLocalDate();
		// Le restamos los días
		LocalDate fRenovacion = f.minusDays(dias);
		// Obtenemos la fecha de hoy
		LocalDate hoy = java.time.LocalDate.now();
		// Si la fecha de renovación es mayor que hoy expira = true
		if (fRenovacion.isAfter(hoy)) {
			expira = true;
		}
		return expira;
	}
```

7. Conclusión y futuras ampliaciones.

Con este proyecto se ha intentado dar solución a un problema real, la tarea que tienen los pilotos de actualizar su logbook. Existen otras soluciones informáticas en el mercado para desarrollar esta labor, pero son económicamente muy costosas y la mayoría de ellas tienen demasiados detalles, por lo que rellenar el logbook electrónico se convierte en una tarea aún más tediosa que completar el formato físico.

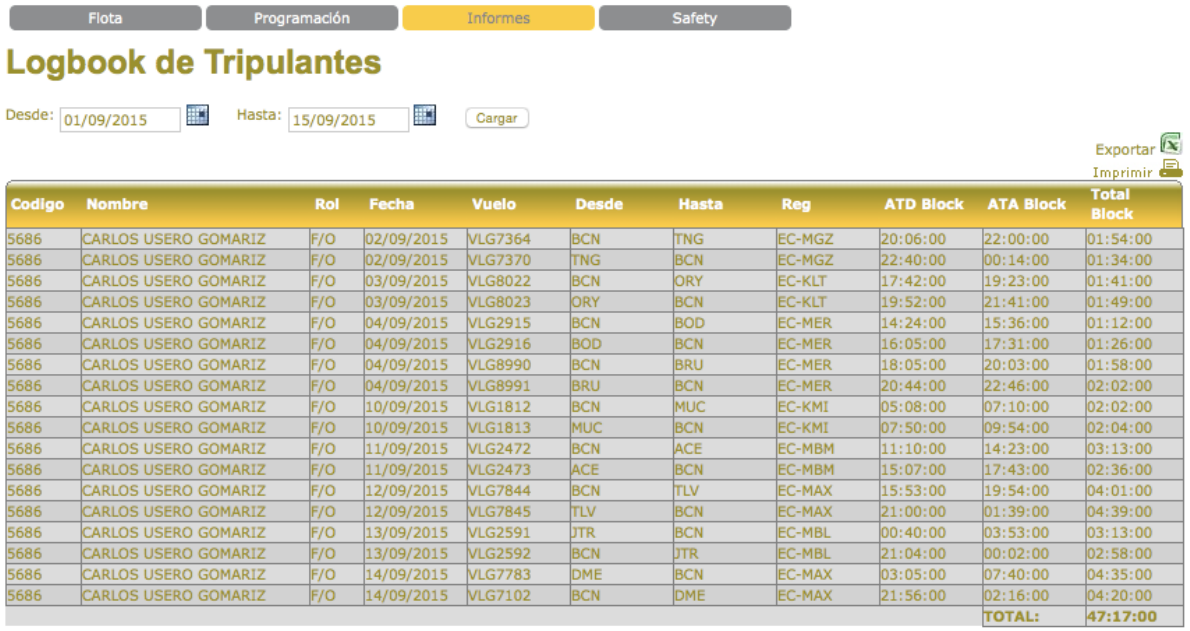
A lo largo de esta memoria se ha comentado en diversas ocasiones la importancia del análisis y el diseño. A medida que evolucionaba el desarrollo de la aplicación he podido comprobar la veracidad de esta afirmación. En un principio el diseño no estaba muy refinado, en parte por la falta de costumbre, ya que llevaba retirada del campo de la informática muchos años. Esto provocaba errores que se iban propagando y eran muy costosos de eliminar. Pero al ir avanzando en el desarrollo del mismo me propuse ser rigurosa con las metodologías escogidas. Gracias a esto, pasé de una idea vaga en mi cabeza a tener claro en mi mente el resultado deseado cuando escribía la primera línea de código. Para algún con mi inexperiencia y mi falta de práctica sin duda hubiera sido imposible realizar la aplicación sin la ayuda de estas metodologías.

A pesar de todo se han quedado algunas ideas en el tintero que sería interesante desarrollar en futuras versiones:

- **Soporte móvil:** Era uno de los puntos claves y una de las características que consideraba más interesantes, el poder actualizar el logbook desde tu móvil cada vez que finalices un vuelo, o tranquilamente sentado delante del ordenador, lo que mejor convenga al usuario en cada momento. Por este motivo se decidió el desarrollo de una aplicación web y podríamos desarrollar en el futuro una interfaz para el móvil o la tableta usando Bootstrap.

- **Aplicación para las compañías:** Se expone a continuación una captura real del logbook de un piloto, provisto por su compañía aérea. Se puede observar que no está toda la información a anotar en el cuaderno, aunque por supuesto son aspectos intrínsecos que un piloto conoce. Pero tratando de automatizar la tarea de completar el libro lo máximo posible para que el usuario no tenga que pensar, no está de más mostrarla de forma completa , tal y como realizamos en la aplicación. Además, no están totalizadas las horas, que es una de las tareas más pesadas. Se puede observar

que se puede exportar la información a un fichero .xls. Es fácil encontrar en la red scripts que trasforman este tipo de datos en JSON, que podría ser utilizado por la base de datos. Así que sería una buena idea tratar de automatizar la descarga de información periódica de este tipo de webs, de forma que el usuario no tuviera que introducir de forma manual cada línea.


Flota | Programación | Informes | Safety

Logbook de Tripulantes

Desde: 01/09/2015 Hasta: 15/09/2015 Cargar

Exportar
Imprimir

Codigo	Nombre	Rol	Fecha	Vuelo	Desde	Hasta	Reg	ATD Block	ATA Block	Total Block
5686	CARLOS USERO GOMARIZ	F/O	02/09/2015	VLG7364	BCN	TNG	EC-MGZ	20:06:00	22:00:00	01:54:00
5686	CARLOS USERO GOMARIZ	F/O	02/09/2015	VLG7370	TNG	BCN	EC-MGZ	22:40:00	00:14:00	01:34:00
5686	CARLOS USERO GOMARIZ	F/O	03/09/2015	VLG8022	BCN	ORY	EC-KLT	17:42:00	19:23:00	01:41:00
5686	CARLOS USERO GOMARIZ	F/O	03/09/2015	VLG8023	ORY	BCN	EC-KLT	19:52:00	21:41:00	01:49:00
5686	CARLOS USERO GOMARIZ	F/O	04/09/2015	VLG2915	BCN	BOD	EC-MER	14:24:00	15:36:00	01:12:00
5686	CARLOS USERO GOMARIZ	F/O	04/09/2015	VLG2916	BOD	BCN	EC-MER	16:05:00	17:31:00	01:26:00
5686	CARLOS USERO GOMARIZ	F/O	04/09/2015	VLG8990	BCN	BRU	EC-MER	18:05:00	20:03:00	01:58:00
5686	CARLOS USERO GOMARIZ	F/O	04/09/2015	VLG8991	BRU	BCN	EC-MER	20:44:00	22:46:00	02:02:00
5686	CARLOS USERO GOMARIZ	F/O	10/09/2015	VLG1812	BCN	MUC	EC-KMI	05:08:00	07:10:00	02:02:00
5686	CARLOS USERO GOMARIZ	F/O	10/09/2015	VLG1813	MUC	BCN	EC-KMI	07:50:00	09:54:00	02:04:00
5686	CARLOS USERO GOMARIZ	F/O	11/09/2015	VLG2472	BCN	ACE	EC-MBM	11:10:00	14:23:00	03:13:00
5686	CARLOS USERO GOMARIZ	F/O	11/09/2015	VLG2473	ACE	BCN	EC-MBM	15:07:00	17:43:00	02:36:00
5686	CARLOS USERO GOMARIZ	F/O	12/09/2015	VLG7844	BCN	TLV	EC-MAX	15:53:00	19:54:00	04:01:00
5686	CARLOS USERO GOMARIZ	F/O	12/09/2015	VLG7845	TLV	BCN	EC-MAX	21:00:00	01:39:00	04:39:00
5686	CARLOS USERO GOMARIZ	F/O	13/09/2015	VLG2591	JTR	BCN	EC-MBL	00:40:00	03:53:00	03:13:00
5686	CARLOS USERO GOMARIZ	F/O	13/09/2015	VLG2592	BCN	JTR	EC-MBL	21:04:00	00:02:00	02:58:00
5686	CARLOS USERO GOMARIZ	F/O	14/09/2015	VLG7783	DME	BCN	EC-MAX	03:05:00	07:40:00	04:35:00
5686	CARLOS USERO GOMARIZ	F/O	14/09/2015	VLG7102	BCN	DME	EC-MAX	21:56:00	02:16:00	04:20:00
									TOTAL:	47:17:00

Figura 44: Logbook de una compañía aérea.

- Uso de AJAX para el rellenado de los formularios: De esta forma, por ejemplo, no se listarían todas las matrículas en la base de datos, sino únicamente las que correspondan al tipo de avión introducido por el usuario, descargando a éste de información no útil. Hoy en día prácticamente no se concibe el desarrollo de una aplicación web sin el uso de tecnologías asíncronas.

- Mejora de la interfaz de usuario: Sustituir los botones y menús por iconos es la tendencia en todas las páginas web modernas, haciéndolas mucho más intuitivas y claras. Hoy en día todo el mundo sabe que una papelera eliminará un dato o que pulsando sobre tres líneas horizontales podrá ver un menú. En w3schools podemos encontrar tutoriales e incluso hojas de estilo disponibles de forma abierta para conseguir este aspecto más contemporáneo.

- Desarrollo de una ayuda en línea: Para facilitar la experiencia del usuario y saque el mayor posible partido a la aplicación.

8. Bibliografía.

8.1. Citas bibliográficas.

[1] CELMA, M. ; CASAMAYOR, J. C.; MOTA, L. Bases de datos relacionales. (2003). Pearson, Prentice Hall.

[2] BOOCH, G. *ULM. El lenguaje Unificado de Modelado. Guía de Usuario.* (2000). Addison-Wesley.

[3] OBJETC MANAGEMENT GROUP. *Introduction To Omg's Unified Modeling Language.* http://www.omg.org/gettingstarted/what_is_uml.htm [Consulta: 21 de Agosto 2015]

[4] SOMMERVILLE, I. *Ingeniería del Software (6ª edición).* (2002). Addison-Wesley.

[5] PIN-SHAN CHEN, P. *The Entity-Relationship Model - Toward a Unified View of Data.* (1976). ACM Transactions on Database Systems, Vol. 1, nº1

8.2. Referencias bibliográficas.

GARCÍA CASTELLANO, F. J. *Tutorial Servlets y JDBC.* http://flanagan.ugr.es/docencia/2005-2006/2/servlets/index.html [Consulta: 29 de Agosto 2015]

MOTA, L.; CELMA, M.; CASAMAYOR, J. C. *Bases de datos relacionales: teoría y diseño.* (1994). SPUPV 767.94.

MYSQL, ORACLE CORP. *5.7 Reference Manual.* En http://dev.mysql.com/doc/refman/5.7/en/ [Consulta: 4 de Agosto 2015]

MYSQL, ORACLE CORP. *Workbench Manual.* http://dev.mysql.com/doc/workbench/en/index.html [Consulta: 9 de Agosto 2015]

ORACLE CORP. *Java Platform Standar Edition 8 Documentation.*
https://docs.oracle.com/javase/8/docs/
[Consulta: 2 de Septiembre 2015]

PRESSMAN, R. (1998). *Ingeniería del Software. Un enfoque práctico (4ª edición).*
McGraw-Hill.

REAL ACADEMIA ESPAÑOLA. *Diccionario de la Legua Española.*
http://www.rae.es [Consulta el 28 de Agosto 2015]

SASTRE MIRALLES, N., NAVARRO LABOULAIS, C.T. *Cómo citar la bibliografía en los trabajos académicos.*
https://riunet.upv.es/handle/10251/31590 [Consulta: 7 de Agosto 2015]

W3SCHOOLS. *CSS 3 Tutorial*
http://www.w3schools.com/css/default.asp [Consulta el 5 de Junio 2015]

W3SCHOOLS. *HTML 5 Tutorial*
http://www.w3schools.com/html/default.asp [Consulta el 3 de Junio 2015]

W3SCHOOLS. *JavaScript Tutorial*
http://www.w3schools.com/js/default.asp [Consulta el 7 de Junio 2015]

WIKIPEDIA. *Wikipedia: La enciclopedia libre.*
https://es.wikipedia.org/ [Consulta el 29 de Agosto 2015]

Anexo A:

Representación de los objetos que conforman el sistema de información en Java (ver sección 5.1.)

```
public class Piloto {

	/* Atributos */
	private int idPiloto;
	private String usuario;
	private String password;
	private String nombre;
	private String email;
	private Date fechaNac;
	private String tipoLic;
	private String numLic;
	private int nivelIngles;
	private Date fechaIngles;

	/* Consultores y modificadores: Cada atributo tendrá su método de
	asignar y consultar. Ejemplo: asignarIdPiloto (int id) y
	obtenerIdPiloto() */

	/* Constructores */
		// Para recuperar objetos de la BBDD
	public Piloto (int id, String us, String pw, String nm, String em,
		Date fn, String tl, String nl, int ni, Date fi)
		// Para insertar objetos en la BBDD (sin el id, ya que es
		// autoincrement)
	public Piloto (String us, String pw, String nm, String em, Date fn,
		String tl, String nl, int ni, Date fi)
		// Constructor vacío
	public Piloto () {}

	/* Modificar objeto */
	public void modificarPiloto (String us, String pw, String nm, String
		em, Date fn, String tl, String nl, int ni, Date fi)

	/* Relaciones y cardinalidades */
		// Logbook 0..1
	private Logbook logbook;
```

```
        public void asignarLogbook (Logbook l)
        public Logbook obtenerLogbook ()
              // Medico 0..3
        private Medico[] medicos = new Medico[3];
        public Medico consultarMedico (int tipoMed)
        public void insertarMedico (Medico m, int tipoMed)
        public void borrarMedico (Medico m)
        public int tamanyoMedicos ()
              // Habilitacion 0..*
        private ArrayList<Habilitacion> habilitaciones = new
              ArrayList<Habilitacion>();
        public Habilitacion consultarHabilitacion (int index)
        public int consultarIndiceHabilitacion (Habilitacion h)
        public void insertarHabilitacion (Habilitacion h)
        public void borrarHabilitacion (Habilitacion h)
        public int tamanyohabilitaciones ()

        /* Base de datos */
        public static boolean insertarPilotoBD(Piloto p)
        public static boolean borrarPilotoBD(Piloto p)
        public static boolean modificarPilotoBD(Piloto p, String col, String
              nuevoValor)
        public static Piloto consultarPilotoBD(int id)

public class Medico {

        /* Atributos */
        private int idMedico;
        private String  numCertificado;
        private int tipoMed;
        private Date fechaMed;

        /* Consultores y modificadores: Cada atributo tendrá su método de
        asignar y consultar. Ejemplo: asignarIdMedico (int id) y
        obtenerIdMedico() */

        /* Constructores */
              // Para recuperar objetos de la BBDD
        public Medico (int id, String nc, int tm, Date fm)
              // Para insertar objetos en la BBDD (sin el id, ya que es
              // autoincrement)
        public Medico (String nc, int tm, Date fm)
```

```
            // Constructor vacío
        public Medico () {}

        /* Modificar objeto */
        public void modificarMedico (String nc, int tm, Date fm)

        /* Relaciones y cardinalidades */
            // Piloto 1
        private Piloto piloto;
        public void asignarPiloto (Piloto p)
        public Piloto obtenerPiloto ()

        /* Base de datos */
        public static boolean insertarMedicoBD (Medico m)
        public static boolean borrarMedicoBD (Medico m)
        public static boolean modificarMedicoBD (Medico m, String col,
            String nuevoValor)
        public static Medico consultarMedicoBD (int id)
}

public class Habilitacion {

        /* Atributos */
        private int idHabilitacion;
        private String tipoHab;
        private Date fechaHab;

        /* Consultores y modificadores: Cada atributo tendrá su método de
        asignar y consultar. Ejemplo: asignarIdHabilitacion (int id) y
        obtenerIdHabilitacion() */

        /* Constructores */
            // Para recuperar objetos de la BBDD
        public Habilitacion Habilitacion (int id, String th, Date fh)
            // Para insertar objetos en la BBDD (sin id, ya que es
            // autoincrement)
        public Habilitacion (String th, Date fh)

        /* Modificar objeto */
        public void modificarHabilitacion (String th, Date fh)
```

```
	/* Relaciones y cardinalidades */
		// Piloto 1
	private Piloto piloto;
	public void asignarPiloto (Piloto p)
	public Piloto obtenerPiloto ()

	/* Base de datos */
	public static boolean insertarHabilitacionBD (Habilitacion h)
	public static boolean borrarHabilitacionBD (Habilitacion h)
	public static boolean modificarHabilitacionBD (Habilitacion h,
		String col, String nuevoValor)
	public static Habilitacion consultarHabilitacion (int id)
}

public class Logbook {

	/* Todos los atributos de Logbook vienen dados por sus relaciones */

	/* Constructores */
	public Logbook (Piloto p, UltimaPagina up, ArrayList<Pagina> pags)
	public Logbook (Piloto p, UltimaPagina up)
	public Logbook () {}

	/* Relaciones y cardinalidades */
		// Piloto 1
	private Piloto piloto;
	public void asignarPiloto (Piloto p)
	public Piloto obtenerPiloto ()
		// Pagina 0..*
	private ArrayList<Pagina> paginas = new ArrayList<Pagina>();
	public Pagina consultarPagina (int index)
	public int consultarIndicePagina (Pagina p)
	public void borrarPagina (Pagina p)
	public int tamanyoPaginas ()
		// UltimaPagina 1
	private UltimaPagina ultima;
	public void asignarUltima (UltimaPagina u)
	public void UltimaPagina obtenerUltima ()

	public Logbook generarLogbook(Logbook log, Piloto p)
	// Rellena y estructura las últimas líneas del usuario
```

```
}
public class Pagina {

	/* Atributos */
	protected int numPagina;
	protected int numLineas;
	protected String totalSe;
	protected String totalMe;
	protected String totalMp;
	protected String totalFt;
	protected int totalToDia;
	protected int totalToNoche;
	protected int totalLndDia;
	protected int totalLndNoche;
	protected String totalNocturno;
	protected String totalIfr;
	protected String totalPic;
	protected String totalCopilot;
	protected String totalDual;
	protected String totalInstructor;
	protected String totalSim;

	/* Consultores y modificadores: Cada atributo tendrá su método de
	asignar y consultar. Ejemplo: asignarNumPagina (int id) y
	obtenerNumPagina() */

	/* Constructores */
	public Pagina (int np, int nl, String tse, String tme, String tmp,
		String tft, int ttod, int tton, int tldd, int tldn, String
		ttno, String ttif, String ttpi, String ttco, String ttdu,
		String ttin, String ttsi) {
		// Para trabajar con UltimaLinea, que no tiene número de
		// líneas porque es siempre una página completa
	public Pagina (int np, String tse, String tme, String tmp, String
		tft, int ttod, int tton, int tldd, int tldn, String ttno,
		String ttif, String ttpi, String ttco, String ttdu, String
		ttin, String ttsi)
		// Constructor vacío
	public Pagina () {}

	/* Modificar objeto */
	public void modificarPagina (int np, int nl, String tse, String tme,
		String tmp, String tft, int ttod, int tton, int tldd, int
```

```
            tldn, String ttno, String ttif, String ttpi, String ttco,
            String ttdu, String ttin, String ttsi)
            // Para trabajar con UltimaLinea, que no tiene número de
            // líneas porque es siempre una página completa
      public void modificarPagina (int np, String tse, String tme, String
            tmp, String tft, int ttod, int tton, int tldd, int tldn,
            String ttno, String ttif, String ttpi, String ttco, String
            ttdu, String ttin, String ttsi)

      /* Relaciones y cardinalidades */
            // Agregación: Navegación restringuida a Logbook
            // Linea 0..18
      private Linea[] lineas = new Linea[18];
      public Linea consultarLinea (int idLi)
      public Linea consultarPosLinea (int pos)
      public void insertarLinea (Linea l, int pos)
      public void borrarLinea (Linea l)
      public int tamanyoLineas ()
}

public class UltimaPagina extends Pagina {

      /* Atributos */
      private int idUltimaPagina;
      private int numUltima;
      private int ultimaLinea;

      /* Consultores y modificadores: Cada atributo tendrá su método de
      asignar y consultar. Ejemplo: asignarIdUltimaPagina (int id) y
      obtenerIdUltimaPagina() */

      /* Constructores */
            // Para recuperar objetos de la BD
      public UltimaPagina (int id, int nu, int idu, String tse, String
            tme, String tmp, String tft, int ttod, int tton, int tldd, int
            tldn, String ttno, String ttif, String ttpi, String ttco,
            String ttdu, String ttin, String ttsi) {
                  super (nu, tse, tme, tmp, tft, ttod, tton, tldd, tldn,
                  ttno, ttif, ttpi, ttco, ttdu, ttin, ttsi);
                  idUltimaPagina = id;
                  ultimaLinea = idu;
      }
```

```
            // Para insertar objetos en la BBDD (sin el id, ya que es
            // autoincrement)
      public UltimaPagina (int nu, int idu, String tse, String tme, String
            tmp, String tft, int ttod, int tton, int tldd, int tldn,
            String ttno, String ttif, String ttpi, String ttco, String
            ttdu, String ttin, String ttsi) {
                  super (nu, tse, tme, tmp, tft, ttod, tton, tldd, tldn,
                  ttno, ttif, ttpi, ttco, ttdu, ttin, ttsi);
                  ultimaLinea = idu;
      }
            // Constructor vacío
      public UltimaPagina () { super(); }

      /* Modificar objeto */
      public void modificarUltimaPagina (int nu, int idu, String tse,
            String tme, String tmp, String tft, int ttod, int tton, int
            tldd, int tldn, String ttno, String ttif, String ttpi, String
            ttco, String ttdu, String ttin, String ttsi) {
                  super.modificarPagina(nu, tse, tme, tmp, tft, ttod,
                  tton, tldd, tldn, ttno, ttif, ttpi, ttco, ttdu, ttin,
                  ttsi);
                  ultimaLinea = idu;
      }

      /* Relaciones y cordialidades */
            // Logbook 1
      private Logbook logbook;
      public void asignarLogbook (Logbook l)
      public void obtenerLogbook ()

      /* Base de datos */
      public static boolean insertarUltimaLineaBD (UltimaLinea u)
      public static boolean borrarUltimaLineaBD (UltimaLinea u)
      public static boolean modificarUltimaLineaBD (UltimaLinea u, String
            col, String nuevoValor)
      public static boolean consultarUltimaLineaBD (int id)
}

public class Linea {

      /* Atributos */
      protected int idLinea;
      protected Date fechaLinea;
```

```
        protected String comentarios;
        protected boolean sim;

        /* Consultores y modificadores: Cada atributo tendrá su método de
        asignar y consultar. Ejemplo: asignarIdLinea (int id) y
        obtenerIdLinea() */

        /* Constructores */
              // Para recuperar objeto de la BBDD
        public Linea (int id, Date fl, String cm, boolean s)
              // Para insertar objeto en la BBDD (sin id, ya que es
              // autoincrementado)
        public Line (Date fl, String cm, boolean s)
              // Constructor vacío
        public Linea () {}

        /* Modificar objeto */
        public modificarLinea (Datefl, String cm, boolean s)

        /* Relaciones y cardinalidades */
              // Agregación: Navegación restringida a Página

        /* Base de datos */
        public static boolean insertarLineaBD (Linea l)
              // Inserta en Línea y en Vuelo o SesionSim, según corresponda
        public static boolean borrarLineaBD (Linea l)
              // Borra la Línea y el Vuelo o SesionSim con el mismo id
        public static boolean modificarLineaBD (Linea l, String col, String
              nuevoValor)
              // Modifica las tablas Linea y Vuelo o SesionSim
        public static Linea consultarLineaBD (int id)
              // Recupera un Vuelo o una Sesion, que devuelve como Linea
}

public class Vuelo extends Linea {

        /* Atributos */
        private String arpSalida;
        private Time horaSalida;
        private String arpLlegada;
        private Time horaLlegada;
```

```
private Time tiempoTotal;
private String nombrePic;
private int toDia;
private int toNoche;
private int lndDia;
private int lndNoche;
private Time tiempoNocturno;
private Time tiempoIfr;
private Time tiempoPic;
private Time tiempoCopilot;
private Time tiempoDual;
private Time tiempoInstructor;

/* Consultores y modificadores: Cada atributo tendrá su método de
asignar y consultar. Ejemplo: asignarArpSalida (String as) y
obtenerArpSalida() */
      // Hereda los atributos y métodos de la clase Linea

/* Consultores */
      // Recuperar objeto de la BBDD. Llama a constructor de super
public Vuelo (int id, Date fv, String as, Time hs, String all, Time
      hll, Time tt, String pic, int tod, int ton, int ld, int ln,
      Time tn, Time ti, Time tp, Time tco, Time tdu, Time tfi,
      String com, boolean s) {
            super (id, fv, com, s)
            //Asignar el resto de parámetros
}
      // Insertar objeto en la BBDD (sin id, ya que es
      // autoincrementado)
public Vuelo (Date fv, String as, Time hs, String all, Time hll,
      Time tt, String pic, int tod, int ton, int ld, int ln, Time
      tn, Time ti, Time tp, Time tco, Time tdu, Time tfi,
      String com, boolean s) {
            super (fv, com, s)
            //Asignar el resto de parámetros
}
      // Constructor vacío
public Vuelo () { super(); }

/* Modificar un objeto */
public void modificarVuelo (Date fv, String as, Time hs, String all,
      Time hll, Time tt, String pic, int tod, int ton, int ld, int
```

```
            ln, Time tn, Time ti, Time tp, Time tco, Time tdu, Time tfi,
            String com, boolean s) {
                  super.modificarLinea(fs, com, s);
                  // Asignar el resto de parámetros
      }
      /* Relaciones y cardinalidades */
            // Avion 1
      private Avion avion;
      public void asignarAvion (Avion a)
      public void obtenerAvion ()

      /* Base de datos */
            // Estos métodos reciben una Linea, ya que son llamados desde
            // los de esta clase padre
      public static boolean insertarVueloBD (Linea l)
      public static boolean borrarVueloBD (Linea l)
      public static boolean modificarVueloBD (Linea l, String col, String
            nuevoValor)
      public static Vuelo consultarVueloBD (int id)
}

public class SesionSim extends Linea {

      /* Atributos */
      private Time tiempoSim;

      /* Consultores y modificadores: Cada atributo tendrá su método de
      asignar y consultar. Ejemplo: asignarTiempoSim (String ts) y
      obtenerTiempoSim() */
            // Hereda los atributos y métodos de la clase Linea

      /* Constructores */
            // Recuperar objeto de la BBDD
      public SesionSim (int id, Date fs, Time ts, String com, boolean s) {
            super (id, fs, com, s)
            tiempoSim = ts;
      }
            // Insertar objeto en la BBDD (sin id, ya que es
            // autoincrementado)
      public SesionSim (Date fs, Time ts, String com, boolean s) {
            super (fs, com, s)
            tiempoSim = ts;
```

```
	}
		// Constructor vacío
	public SesionSim () { super(); }

	/* Modificar objeto
	public void modificarSesionSim (Date fs, Time ts, String com,
		boolean s) {
			super (fs, com, s)
			tiempoSim = ts;
	}

	/* Relaciones y cardinalidades */
		// TipoSim 1
	private TipoSim sim;
	public void asignarTipoSim (TipoSim s)
	public void obtenerTipoSim ()

	/* Base de datos */
		// Estos métodos reciben una Linea, ya que son llamados desde
		// los de esta clase padre
	public static boolean insertarSesionSimBD (Linea l)
	public static boolean borrarSesionSimBD (Linea l)
	public static boolean modificarSesionSimBD (Linea l, String col,
		String nuevoValor)
	public SesionSim consultarSesionSimBD (int id)

}

public class Avion {

	/* Atributos */
	private int idAvion;
	private String matricula;

	/* Consultores y modificadores: Cada atributo tendrá su método de
	asignar y consultar. Ejemplo: asignarIdAvion (int id) y
	obtenerIdAvion() */

	/* Constructores */
		// Para recuperar de la BBDD
	public Avion (int id, String ma)
```

```
		// Para insertar en la BBDD (sin id, ya que es
		// autoincrementado)
	public Avion (String ma)
		// Constructor vacío
	public Avion () {}

	/* Modificar objeto */
	public void modificarAvion (String ma)

	/* Relaciones: */
		// Tipo Avión 1
	private TipoAvion tipoAv;
	public TipoAvion obtenerTipoAv()
	public void asignarTipoAv(TipoAvion ta)

	/* Base de datos */
	public static boolean insertarAvionBD (Avion a)
	public static boolean borrarAvionBD (Avion a)
	public static boolean modificarAvionBD (Avion a, String col, String
		nuevoValor)
	public static Avion consultarAvionBD (int id)

}

public class TipoAvion {

	/* Atributos */
	private int idTipoAvion;
	private String modeloAvion;
	private boolean spSe;
	private boolean spMe;
	private boolean multiPilot;

	/* Consultores y modificadores: Cada atributo tendrá su método de
	asignar y consultar. Ejemplo: asignarIdTipoAvion (int id) y
	obtenerIdTipoAvion() */

	/* Constructores */
		// Recuperar objeto de la BBDD
	public TipoAvion (int id, String ma, boolean se, boolean me,
		boolean mp)
```

```
            // Insertar objeto en la BBDD (sin id, ya que es
            // autoincrementado)
      public TipoAvion (String ma, boolean se, bolean me, boolean mp)

      /* Modificar objeto */
      public void modificarTipoAvion (String ma, boolean se, boolean me,
            boolean mp)

      /* Relaciones y cardinalidades
            // Avion 0..*
      private ArrayList<Avion> aviones = new ArrayList<Avion>();
      public Avion consultarAvion (int index)
      public int consultarIndiceAvion (Avion a)
      public boolean insertarAvion (Avion a) {
      public boolean borrarAvion (Avion a)
      public int tamanyoAviones ()

      /* Base de datos */
      public static boolean insertarTipoAvionBD (tipoAvion ta)
      public static boolean borrarTipoAvionBD (tipoAvion ta)
      public static boolean modificarTipoAvionBD (tipoAvion ta, String
            col, String nuevoValor)
      public static Avion consultarTipoAvionBD (int id)
}

public class TipoSim {

      /* Atributos */
      private int idTipoSim;
      private String modeloSim;

      /* Consultores y modificadores: Cada atributo tendrá su método de
      asignar y consultar. Ejemplo: asignarIdTipoSim (int id) y
      obtenerIdTipoSim() */

      /* Constructores */
            // Recuperar de la BBDD
      public TipoSim (int id, String ms)
            // Insertar en la BBDD (sin id, ya que es autoincrementado)
      public TipoSim (String ms)
            // Constructor vacío
      public TipoSim () {}
```

```
	/* Relaciones y cardinalidades */
		// SesionSim 0..*
	private ArrayList<SesionSim> sesiones = new ArrayList<SesionSim>();
	public SesionSim consultarSesionSim (int index)
	public int consultarIndiceSesionSim (SesionSim s)
	public boolean insertarSesionSim (SesionSim s)
	public boolean borrarSesionSim (SesionSim s)
	public int tamanyoSesiones ()

	/* Base de datos */
	public static boolean insertarTipoSimBD (TipoSim ts)
	public static boolean borrarTipoSimBD (TipoSim ts)
	public static boolean modificarTipoSimBD (TipoSim ts, String col,
		String nuevoValor)
	public static TipoSim consultarTipoSim (int id)
}
```

Anexo B:

Creación de la base de datos en código MySQL.

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 COLLATE
utf8_general_ci ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`Piloto`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Piloto` (
  `idPiloto` SMALLINT NOT NULL AUTO_INCREMENT COMMENT '',
  `usuario` VARCHAR(20) NOT NULL COMMENT '',
  `password` VARCHAR(20) NOT NULL COMMENT '',
  `nombre` VARCHAR(20) NOT NULL COMMENT '',
  `email` VARCHAR(30) NOT NULL COMMENT '',
  `fechaNac` DATE NULL COMMENT '',
  `tipoLic` CHAR(4) NULL COMMENT '',
  `numLic` VARCHAR(15) NULL COMMENT '',
  `nivelIngles` TINYINT NULL COMMENT '',
  `fechaIngles` DATE NULL COMMENT '',
  PRIMARY KEY (`idPiloto`)  COMMENT '',
  UNIQUE INDEX `usuario_UNIQUE` (`usuario` ASC)  COMMENT '',
  UNIQUE INDEX `email_UNIQUE` (`email` ASC)  COMMENT '')
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`Habilitacion`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Habilitacion` (
  `idHabilitacion` SMALLINT NOT NULL AUTO_INCREMENT COMMENT '',
  `tipoHab` CHAR(5) NOT NULL COMMENT '',
  `fechaHab` DATE NOT NULL COMMENT '',
  `Piloto_idPiloto` SMALLINT NOT NULL COMMENT '',
  PRIMARY KEY (`idHabilitacion`)  COMMENT '',
  INDEX `fk_Habilitacion_Piloto1_idx` (`Piloto_idPiloto` ASC)  COMMENT '',
  CONSTRAINT `fk_Habilitacion_Piloto1`
    FOREIGN KEY (`Piloto_idPiloto`)
    REFERENCES `mydb`.`Piloto` (`idPiloto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Medico`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Medico` (
  `idMedico` SMALLINT NOT NULL AUTO_INCREMENT COMMENT '',
  `numCertificado` VARCHAR(15) NOT NULL COMMENT '',
  `tipoMed` TINYINT NOT NULL COMMENT '',
  `fechaMed` DATE NOT NULL COMMENT '',
  `Piloto_idPiloto` SMALLINT NOT NULL COMMENT '',
  PRIMARY KEY (`idMedico`)  COMMENT '',
  INDEX `fk_Medico_Piloto1_idx` (`Piloto_idPiloto` ASC)  COMMENT '',
  UNIQUE INDEX `numCertificado_UNIQUE` (`numCertificado` ASC)  COMMENT '',
  CONSTRAINT `fk_Medico_Piloto1`
    FOREIGN KEY (`Piloto_idPiloto`)
    REFERENCES `mydb`.`Piloto` (`idPiloto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`TipoAvion`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`TipoAvion` (
  `idTipoAvion` SMALLINT NOT NULL AUTO_INCREMENT COMMENT '',
  `modeloAvion` CHAR(5) NOT NULL COMMENT '',
  `spSe` TINYINT(1) NOT NULL COMMENT '',
  `spMe` TINYINT(1) NOT NULL COMMENT '',
  `multiPilot` TINYINT(1) NOT NULL COMMENT '',
  PRIMARY KEY (`idTipoAvion`)  COMMENT '',
  UNIQUE INDEX `modeloAvion_UNIQUE` (`modeloAvion` ASC)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Avion`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Avion` (
  `idAvion` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `matricula` CHAR(6) NOT NULL COMMENT '',
  `TipoAvion_idTipoAvion` SMALLINT NOT NULL COMMENT '',
  PRIMARY KEY (`idAvion`)  COMMENT '',
  UNIQUE INDEX `matricula_UNIQUE` (`matricula` ASC)  COMMENT '',
   INDEX `fk_Avion_TipoAvion1_idx` (`TipoAvion_idTipoAvion` ASC)   COMMENT
'',
  CONSTRAINT `fk_Avion_TipoAvion1`
    FOREIGN KEY (`TipoAvion_idTipoAvion`)
    REFERENCES `mydb`.`TipoAvion` (`idTipoAvion`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Linea`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Linea` (
  `idLinea` INT NOT NULL COMMENT '',
  `fechaLinea` DATE NOT NULL COMMENT '',
  `comentarios` VARCHAR(200) NULL COMMENT '',
  `sim` TINYINT(1) NOT NULL COMMENT '',
```

```
  `Piloto_idPiloto` SMALLINT NOT NULL COMMENT '',
  PRIMARY KEY (`idLinea`)  COMMENT '',
  INDEX `fk_Linea_Piloto1_idx` (`Piloto_idPiloto` ASC)  COMMENT '',
  CONSTRAINT `fk_Linea_Piloto1`
    FOREIGN KEY (`Piloto_idPiloto`)
    REFERENCES `mydb`.`Piloto` (`idPiloto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Vuelo`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Vuelo` (
  `Linea_idLinea` INT NOT NULL COMMENT '',
  `arpSalida` CHAR(4) NULL COMMENT '',
  `horaSalida` TIME NOT NULL COMMENT '',
  `arpLlegada` CHAR(4) NULL COMMENT '',
  `horaLlegada` TIME NULL COMMENT '',
  `tiempoTotal` TIME NULL COMMENT '',
  `nombrePic` VARCHAR(20) NULL COMMENT '',
  `toDia` TINYINT NULL COMMENT '',
  `toNoche` TINYINT NULL COMMENT '',
  `lndDia` TINYINT NULL COMMENT '',
  `lndNoche` TINYINT NULL COMMENT '',
  `tiempoNocturno` TIME NULL COMMENT '',
  `tiempoIfr` TIME NULL COMMENT '',
  `tiempoPic` TIME NULL COMMENT '',
  `tiempoCopilot` TIME NULL COMMENT '',
  `tiempoDual` TIME NULL COMMENT '',
  `tiempoInstructor` TIME NULL COMMENT '',
  `Avion_idAvion` INT NOT NULL COMMENT '',
  INDEX `fk_Vuelo_Avion1_idx` (`Avion_idAvion` ASC)  COMMENT '',
  INDEX `fk_Vuelo_Linea1_idx` (`Linea_idLinea` ASC)  COMMENT '',
  PRIMARY KEY (`Linea_idLinea`)  COMMENT '',
  CONSTRAINT `fk_Vuelo_Avion1`
    FOREIGN KEY (`Avion_idAvion`)
    REFERENCES `mydb`.`Avion` (`idAvion`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Vuelo_Linea1`
    FOREIGN KEY (`Linea_idLinea`)
    REFERENCES `mydb`.`Linea` (`idLinea`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`TipoSim`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`TipoSim` (
  `idTipoSim` SMALLINT NOT NULL AUTO_INCREMENT COMMENT '',
  `modeloSim` VARCHAR(20) NOT NULL COMMENT '',
  PRIMARY KEY (`idTipoSim`)  COMMENT '',
  UNIQUE INDEX `modeloSim_UNIQUE` (`modeloSim` ASC)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`SesionSim`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`SesionSim` (
  `Linea_idLinea` INT NOT NULL COMMENT '',
  `tiempoSim` TIME NULL COMMENT '',
  `TipoSim_idTipoSim` SMALLINT NOT NULL COMMENT '',
  INDEX `fk_SesionSim_TipoSim1_idx` (`TipoSim_idTipoSim` ASC)  COMMENT '',
  INDEX `fk_SesionSim_Linea1_idx` (`Linea_idLinea` ASC)  COMMENT '',
  PRIMARY KEY (`Linea_idLinea`)  COMMENT '',
  CONSTRAINT `fk_SesionSim_TipoSim1`
    FOREIGN KEY (`TipoSim_idTipoSim`)
    REFERENCES `mydb`.`TipoSim` (`idTipoSim`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_SesionSim_Linea1`
    FOREIGN KEY (`Linea_idLinea`)
    REFERENCES `mydb`.`Linea` (`idLinea`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`UltimaPagina`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`UltimaPagina` (
  `idUltimaPagina` SMALLINT NOT NULL AUTO_INCREMENT COMMENT '',
  `numUltima` INT NOT NULL COMMENT '',
  `ultimaLinea` INT NOT NULL COMMENT '',
  `totalSe` CHAR(8) NULL COMMENT '',
  `totalMe` CHAR(8) NULL COMMENT '',
  `totalMp` CHAR(8) NULL COMMENT '',
  `totalFt` CHAR(8) NULL COMMENT '',
  `totalToDia` MEDIUMINT NULL COMMENT '',
  `totalToNoche` MEDIUMINT NULL COMMENT '',
  `totalLndDia` MEDIUMINT NULL COMMENT '',
  `totalLndNoche` MEDIUMINT NULL COMMENT '',
  `totalNocturno` CHAR(8) NULL COMMENT '',
  `totalIfr` CHAR(8) NULL COMMENT '',
  `totalPic` CHAR(8) NULL COMMENT '',
  `totalCopilot` CHAR(8) NULL COMMENT '',
  `totalDual` CHAR(8) NULL COMMENT '',
  `totalInstructor` CHAR(8) NULL COMMENT '',
  `totalSim` CHAR(8) NULL COMMENT '',
  `Piloto_idPiloto` SMALLINT NOT NULL COMMENT '',
  PRIMARY KEY (`idUltimaPagina`)  COMMENT '',
  INDEX `fk_UltimaPagina_Piloto1_idx` (`Piloto_idPiloto` ASC)  COMMENT '',
   UNIQUE INDEX `Piloto_idPiloto_UNIQUE` (`Piloto_idPiloto` ASC)   COMMENT
'',
  CONSTRAINT `fk_UltimaPagina_Piloto1`
    FOREIGN KEY (`Piloto_idPiloto`)
    REFERENCES `mydb`.`Piloto` (`idPiloto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Anexo C:
Manual de usuario.

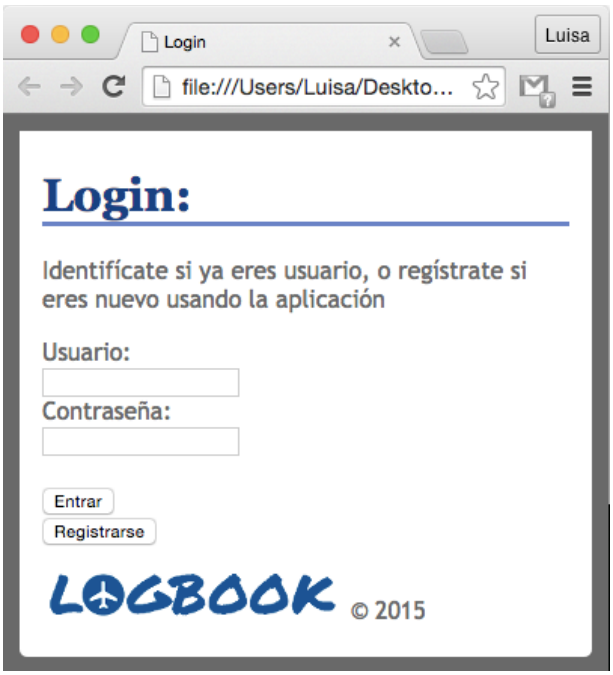
Logbook: Una herramienta de supervisión para las actividades de pilotos de aeronaves.

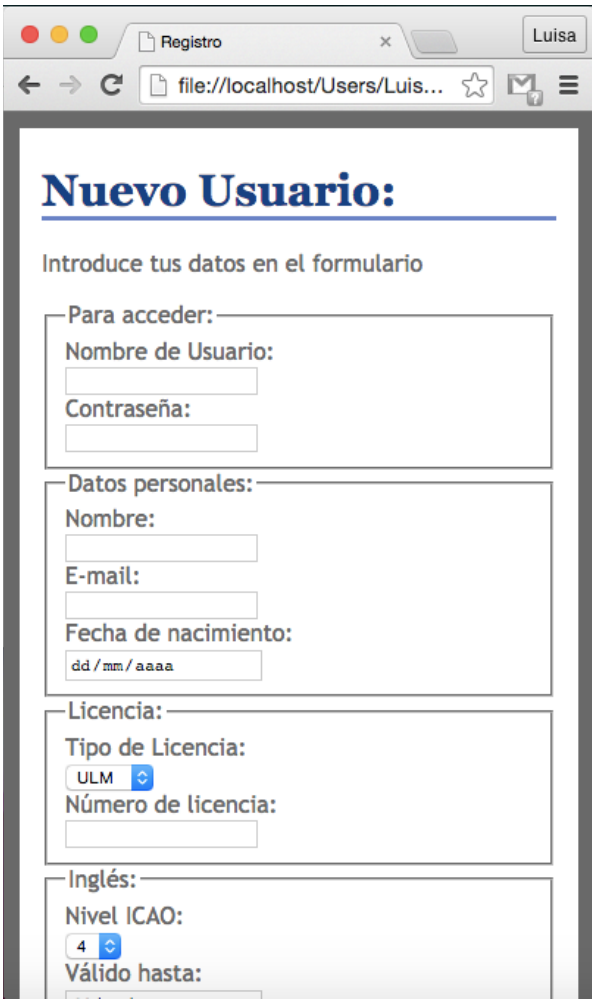
Con esta aplicación se pretenden dar solución a dos tareas fundamentales:

- Vista logbook: Esta vista presentará sus registros, tanto de vuelos como de simuladores, de una forma muy similar a como está representada en el libro físico. Es importante que el usuario tenga en cuenta que una página que se muestre debe de ser transcrita al documento original, pues sólo se generará una vez. La aplicación recuerda cual fue la última página que el usuario relleno, así que lo único que el piloto tiene que hacer es coger un boli y anotar, calcularemos todos los totales para usted, para que no tenga que perder el tiempo nunca más.
- Generar informes con todo tipo de patrones de búsqueda: Si el usuario quiere ver una página antigua de su logbook, como se ha explicado en el punto anterior, no podrá. Pero no tiene más que seleccionar los registros entre las fechas de la primera y la última línea de esa página en concreto. También puede saber cuantas horas de instrumental realizó con un avión, o quien fue el comandante el día de su cumpleaños, las posibilidades son infinitas.

1. Soy nuevo usando la aplicación.

El único requisito para empezar a utilizar logbook es registrarse aportando una dirección de correo electrónico válida. En la página de login pulse “Registro" y rellene el formulario.





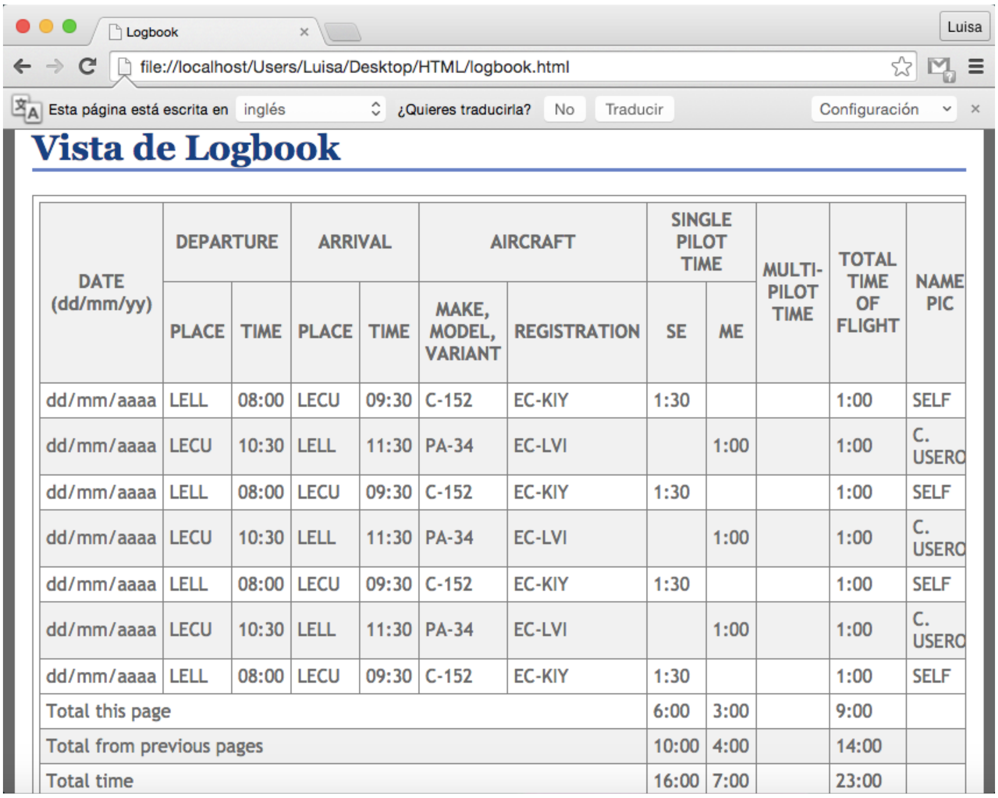
2. Menús.

La aplicación dispone de cinco pestañas:

Usuario: Es la página de entrada y muestra toda su información. Además le recordará si tiene que coger cita con su médico aeronáutico y si expira alguna de sus licencias o el certificado ICAO de inglés. Desde esta página puede, además:

- Modificar sus datos.
- Modificar su certificado de inglés.
- Modificar/Borrar/Crear un certificado médico.
- Modificar/borrar/Crear una habilitación.
- Dar de baja su cuenta de usuario.

Logbook: Le muestra la información lista para copiar en su libro físico. Recuerde que sólo podrá ver esta información una vez.



DATE (dd/mm/yy)	DEPARTURE		ARRIVAL		AIRCRAFT		SINGLE PILOT TIME		MULTI-PILOT TIME	TOTAL TIME OF FLIGHT	NAME PIC
	PLACE	TIME	PLACE	TIME	MAKE, MODEL, VARIANT	REGISTRATION	SE	ME			
dd/mm/aaaa	LELL	08:00	LECU	09:30	C-152	EC-KIY	1:30			1:00	SELF
dd/mm/aaaa	LECU	10:30	LELL	11:30	PA-34	EC-LVI		1:00		1:00	C. USERO
dd/mm/aaaa	LELL	08:00	LECU	09:30	C-152	EC-KIY	1:30			1:00	SELF
dd/mm/aaaa	LECU	10:30	LELL	11:30	PA-34	EC-LVI		1:00		1:00	C. USERO
dd/mm/aaaa	LELL	08:00	LECU	09:30	C-152	EC-KIY	1:30			1:00	SELF
dd/mm/aaaa	LECU	10:30	LELL	11:30	PA-34	EC-LVI		1:00		1:00	C. USERO
dd/mm/aaaa	LELL	08:00	LECU	09:30	C-152	EC-KIY	1:30			1:00	SELF
Total this page							6:00	3:00		9:00	
Total from previous pages							10:00	4:00		14:00	
Total time							16:00	7:00		23:00	

Vuelo: Puede introducir un nuevo vuelo, así como:

- Modificar/Borrar/Crear un nuevo tipo de avión.
- Modificar/Borrar/Crear un nuevo avión con su matrícula.

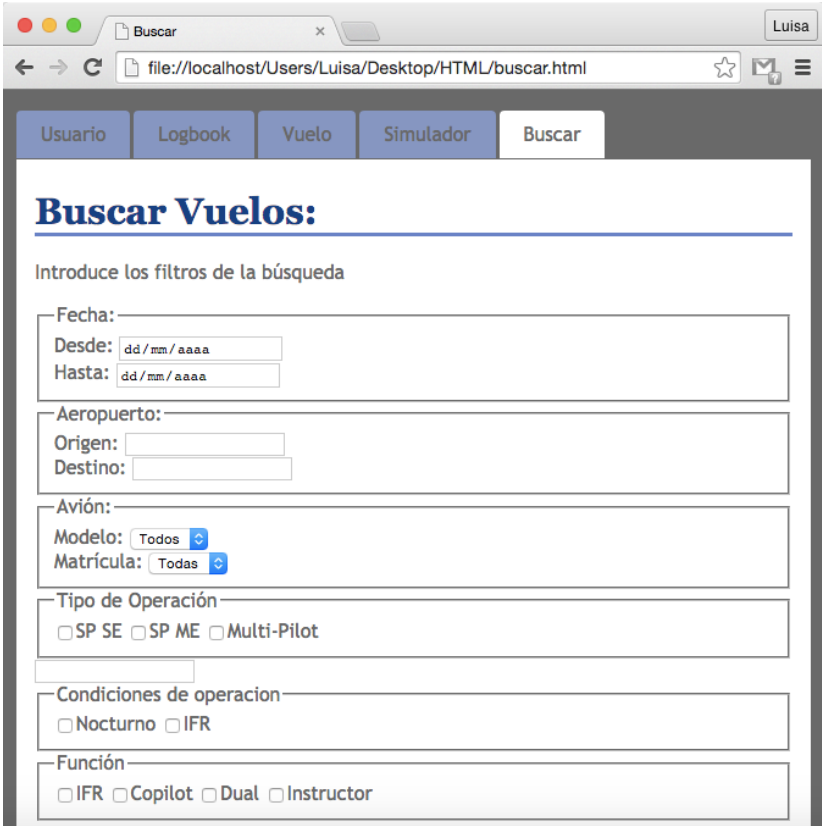
Recuerde que tiene que dar de alta los aviones antes de poder introducir un vuelo. Después de la primera vez le aparecerá en la lista desplegable.

Simulador: Para anotar sus entrenamientos en simuladores, además:

- Modificar/Borrar/Crear un nuevo tipo de simulador.

Recuerde que tiene que dar de alta el tipo de simulador antes de poder introducir una sesión. Después de la primera vez le aparecerá en la lista desplegable.

Buscar: Combine los parámetros de búsqueda de cualquier forma que imagine. Cuando se muestre el resultado podrá cerrar la ventana o borrar los registros obtenidos. Tenga en cuenta que los registros no se pueden modificar, sólo borrar. Y este será el único sitio en el que pueda hacerlo.



3. Modificar/Borrar/Crear

Se muestran juntos en la misma ventana y siempre de la misma forma:

- Crear: Opción seleccionada por defecto. Complete la información y envíe el formulario.

- Modificar: Elija el item que desea modificar en la lista desplegable. El formulario se autorellena con los valores actuales y únicamente tiene que cambiar el que desee.

- Borrar: Elija el item que desea borrar en la lista desplegable y marque la opción “Eliminar” en la esquina inferior derecha.

Happy flights!