Trabajo Fin de Grado en
Ingeniería de las Tecnologías de Telecomunicación

Aplicación web conforme a patrones para la docencia

Autor: Fabián Muñoz Peñas

Tutor: Isabel Román Martínez

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo fin de Grado en
Ingeniería de las Tecnologías de Telecomunicación

Aplicación web conforme a patrones para la docencia

Autor:
Fabián Muñoz Peñas

Tutor:
Isabel Román Martínez
Profesora colaboradora

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo Fin de Grado: Aplicación web conforme a patrones para la docencia

Autor: Fabián Muñoz Peñas

Tutor: Isabel Román Martínez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mis padres y abuelos, sin los que nada de esto hubiera sido posible y a los que nunca seré capaz de agradecerle todo lo que han hecho por mí

A mi compañera de viaje, Cristina, por estar SIEMPRE ahí

A esos dos locos, David y Gonzalo, que comenzaron este camino conmigo y han llegado a ser como hermanos para mí

A mi tutora del TFG, Isabel Román, por su apoyo y consejos tanto personales como profesionales y por toda la "lata" que le he dado

A mi socio, esa persona que siempre me apoyó incondicionamente aunque, en algunas ocasiones, no llevase razón, con la que tan bien me compenetraba y a la que tanto echo de menos, mi abuelo Fernando. Siento que te lo debía y, por tanto, esto va por ti

Resumen

La mayor dificultad que presentan los alumnos que cursan Ingeniería en Telecomunicaciones en nuestra escuela está relacionada con la asimilación de los conceptos impartidos en las distintas clases teóricas. Esto puede deberse a motivos muy diversos, comprendidos entre la carencia de dominio de conceptos estudiados previamente en otras asignaturas, la complejidad de las distintas asignaturas impartidas o la falta de tiempo para impartir dichas asignaturas.

Con la necesidad de ayudar a solucionar los problemas antes expuestos, surge la aplicación web explicada en esta memoria.
Ésta es una aplicación creada con un objetivo totalmente didáctico. Resumido muy brevemente, con la aplicación se pretende que los alumnos comprendan de forma práctica los conceptos explicados previamente en la parte teórica y que ésta les sirva de ayuda para asimilarlos más fácilmente.

Para ello, la aplicación creada estará bien estructurada, comentada y documentada, haciendo uso de algunos de los paradigmas de programación explicados en las distintas asignaturas en las que será usada.

Debido al tiempo tan reducido con el que se encuentran los docentes a la hora de impartir sus asignaturas, la aplicación se desarrolla con el fin de poder ser utilizada en varias asignaturas, ayudando a reducir el tiempo de adaptación de los alumnos con el dominio de la aplicación. También, el usar la misma aplicación en varias asignaturas, puede llevar a los alumnos a descubrir la relación entre los contenidos impartidos en ellas.

Como herramientas que han servido de ayuda en la realización de la aplicación, ha sido utilizado el *IDE Eclipse* para el desarrollo del grueso de la aplicación, *pgAdmin III PostgreSQL* para la gestión y manejo de la Base de Datos, *MagicDraw UML* para el diseño de la aplicación y los distintos *navegadores web* para realizar pruebas y depuraciones de código, entre otras.

Si nos centramos en el contenido de esta memoria, además de lo que ha sido citado anteriormente, en primer lugar, se puede ver una explicación a nivel general de todos y cada uno de los paradigmas de diseño y desarrollo software, lenguajes de programación y programas usados.
Tras esto, se explica el desarrollo completo del proyecto, comenzando por el análisis y definición de actores, requisitos y restricciones que ayudan a definir de forma clara lo necesario; y finalizando con el diseño y desarrollo del mismo. En ésta parte, se explica cómo se ha realizado todo lo definido en el paso anterior.
Posteriormente, se encuentran apartados explicativos de los métodos de trabajo seguidos, de los problemas más importantes surgidos durante la realización del proyecto y de los posibles desarrollos futuros.
Para finalizar, se encuentran varios anexos que servirán de ayuda tanto en el despliegue de la aplicación en el servidor, en el arranque y manejo del servicio de PostgreSQL y en la gestión de la Base de Datos gracias a un tutorial creado para ello.

Tras la finalización del desarrollo de la aplicación, considero que ésta, aunque pueda sufrir alguna modificación para ajustarse a lo realmente requerido, será de gran utilidad en el aprendizaje del alumnado de las prácticas que hagan uso de ella.

A modo de breve conclusión personal, el desarrollo de esta aplicación me ha llevado a enfrentarme a un proyecto completo, desde el análisis y la definición de objetivos y requisitos hasta la documentación del mismo.
Considero esto como una aproximación a la forma de abordar un proyecto en la vida "real", ayudándome, a mejorar, además de asimilando conceptos y aprendiendo más sobre programación, diseño y arquitectura software, en planificación y organización, independencia y autonomía, adaptabilidad y flexibilidad, capacidad de investigación y de resolver problemas.

Abstract

The biggest difficulty that the students who study Telecommunications Engineering in our school have is related to the assimilation of the concepts taught in the different theoretical classes. This can be due to very different reasons, included between the lack of knowledge of previously studied concepts in other subjects, the complexity of the different subjects taught or the lack of time to teach these subjects.

With the need to help solve the problems that we have mentioned before, the web application explained in this report arises.
This is an application created with a didactic objective. With this application, we want that the students understand in a practical way the concepts previously explained in the theoretical part and so it will help them to assimilate these concepts more easily.

For this, the application will be well structured, commented and documented, making use of some of the programming paradigms explained in the different subjects in which it will be used.

The application is developed in order to be used in various subjects, helping to reduce the students' adaptation time with the domain of the application. Also, using the same application in several subjects, can lead students to discover the relationship between the contents taught in them.

Some of tools that have helped in the execution of the application are: *Eclipse IDE* which has been used for the development of the bulk of the application, *pgAdmin III PostgreSQL* for the management of the Database, *MagicDraw UML* for the design of the application and the different *web browsers* to test and debug code, among others.

In this memory, we can find:

- An explanation of all paradigms of design and software development, programming languages and used programs in the development of the web application.
- Analysis and definition of actors, requirements and restrictions that help define clearly what is necessary.
- Design and development of the project.
- An explanation of the work methods followed, of the most important problems that have arisen during the realization of the project and of possible future developments.
- Several annexes that will help both in the deployment of the application on the server, in the startup and management of the PostgreSQL service and in the management of the Database.

I believe that the web application will be very useful in the learning of students of the practices that make use of it.

As a personal conclusion, the development of this application has led me to face a complete project, from the analysis and definition of objectives and requirements to the documentation of it.
I consider this as an approach to the way of approaching a project in "real" life, helping me to improve in design and software architecture, in planning and organization, independence and autonomy, adaptability and flexibility, research capacity and problem solving.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Motivación y Objetivos

En el tercer curso del Grado de Ingeniería en Tecnologías de Telecomunicación se imparte la asignatura Ingeniería del software. Los alumnos, en años anteriores, cursarán varias asignaturas que les aportarán unos conocimientos previos para poder comprender dicha asignatura.

Las asignaturas relacionadas con Ingeniería del software que los alumnos han cursado previamente son Fundamentos de Programación I, Fundamentos de Programación II y Fundamentos y Aplicaciones de Servicios Telemáticos. Los conocimientos previos de programación que aportan dichas asignaturas son: manejo de lenguaje C y creación de algoritmos y programas en dicho lenguaje, manejo del lenguaje Java, resolución de algoritmos y técnicas de Programación Orientada a Objetos, diseño y desarrollo de aplicaciones webs de distinto tipo (estáticas y dinámicas) en distintos lenguajes de programación (HTML, CSS, XML, AJAX, CGI o JSP entre otros), acceso desde las aplicaciones web a las bases de datos y nociones muy básicas de manejo de bases de datos relacionales (en concreto de SQL).

Pero, gran parte del alumnado cursa Ingeniería del Software sin tener bien asimilados los conocimientos previos estudiados en otras asignaturas y comentados anteriormente. Este problema puede dificultar mucho el aprendizaje de los conceptos impartidos en la asignatura, pero no es el único. En la asignatura se desarrollan conceptos que presentan cierta complejidad para alumnos con poca experiencia. Actualmente, se utilizan diferentes ejemplos para desarrollar las prácticas de la asignatura, utilizando Java como lenguaje de programación y UML para los modelos. Se propone utilizar un único ejemplo, que se pueda estudiar desde distintas perspectivas según el objetivo de cada práctica, y que permita además revisar otros lenguajes de programación vistos en cursos anteriores.

El objetivo del proyecto es el diseño de una aplicación bien estructurada, unificada para todas las prácticas,autoexplicativa y con una clara orientación didáctica para el aprendizaje de los alumnos. Se quiere que esta aplicación utilice los paradigmas y lenguajes de programación aprendidos en las asignaturas ya cursadas por los alumnos para ejemplificar de forma práctica los aspectos tratados en Ingeniería del Software. Esta aplicación será utilizada a lo largo de las diferentes prácticas de la asignatura, poniendo en cada una de ellas el foco en un tema de interés. También, se busca que la aplicación pueda ser utilizada en otras asignaturas, con la finalidad de facilitar a los alumnos la identificación de conexiones entre los contenidos estudiados en distintos cursos y minimizar el tiempo que deben dedicar a familiarizarse con el dominio de la aplicación, que tiene poco interés docente.

Así, esta aplicación se crea con el objetivo de:

- Reforzar los conocimientos previos de los alumnos.
- Servir de nexo de unión entre asignaturas para facilitar a los alumnos la creación de estos vínculos.
- Optimizar el tiempo de aprendizaje de la asignatura por parte de los alumnos intentando que éstos pierdan el menor tiempo posible en familiarizarse con el dominio de la aplicación.
- Facilitar la comprensión de los nuevos aspectos tratados en la asignatura.

1.2 Material / Recursos utilizados

En este apartado procederemos a listar el material utilizado en nuestro proyecto, tanto hardware como software.

En cuanto al **hardware**, hemos usado un equipo *Dell XPS Studio 1645 Laptop* con un procesador Intel Core 2 Duo a una velocidad de reloj de 2.40 GHz y una memoria RAM de 4 GB.

En cuanto al **software**, hemos utilizado:

- **Sistema Operativo**:
  - *Windows 7 Professional 64 bits*.
  - *Linux Ubuntu 16.0 64 bits*. Este sistema operativo lo hemos hecho correr de forma virtual sobre Windows a través del software de virtualización VMWare Player Versión 6.0.0.

  Hemos decidido realizar la mayor parte de nuestra aplicación sobre el sistema operativo Windows 7 Professional usando Linux Ubuntu para pruebas de funcionalidad de la aplicación y de gestión de la Base de Datos. El uso de Linux Ubuntu viene motivado por la necesidad de replicar el mismo escenario (estamos usando la máquina virtual del departamento de Ingeniería Telemática, la cual usan los alumnos) que se encontrarán los alumnos cuando estén realizando sus prácticas.
- Para la realización de nuestra aplicación, hemos trabajado con un **Entorno de Desarrollo (o IDE)**. En nuestro caso, éste ha sido *Eclipse Mars Versión 2*.
- Para la **gestión y configuración de la Base de Datos** hemos usado:
  - En Windows 7, el programa *pgAdmin III PostgreSQL 9.5* a través de su interfaz gráfica.
  - En Linux Ubuntu, *psql* como consola de PostgreSQL.
- Para el diseño **de esquemas UML explicativos**, hemos utilizado *MagicDraw UML 16.9*.
- En cuanto a los **navegadores web** utilizados para la depuración y pruebas de la aplicación:
  - *Navegador de Eclipse*.
  - *Google Chrome*.
  - *Mozilla Firefox*.

2 ESTADO DE LA TÉCNICA

Para desarrollar el proyecto, se han usado una serie de lenguajes de programación tanto para la parte del cliente como para la del servidor, así como para el acceso y gestión de la base de datos. Toda la programación anteriormente citada se ha realizado cumpliendo patrones de diseño y arquitectura, cuyo aprendizaje es uno de los pilares fundamentales de la asignatura Ingeniería del Software. Además, para llevar a cabo el diseño y desarrollo se ha usado un programa de diseño UML (MagicDraw) y un Entorno de Desarrollo (Eclipse).

En este apartado explicaremos, entre otras cosas, a nivel de conceptos generales todo lo anteriormente mencionado.

2.1 Lenguajes de Programación

2.1.1 Parte Cliente – FrontEnd

2.1.1.1 HTML

A modo de definición podemos decir que HTML *(HyperText Markup Language o Lenguaje de Marcas de Hipertexto en su traducción española)* es un lenguaje de marcado descriptivo utilizado en la elaboración de páginas web.

El lenguaje HTML sirve para describir la estructura básica de una página y organizar la forma en que dicha página mostrará su contenido. Éste está basado en etiquetas, las cuales son interpretadas por el navegador y sirven para describir el formato del texto y otros elementos que compondrán una página web, así como el "layout". Dispone de etiquetas para imágenes, listas, tablas, botones, enlaces que permiten la redirección a otras páginas, estilo, etc.

Anteriormente, hemos indicado que HTML es un lenguaje de marcado. Básicamente, podemos diferenciar entre dos tipos de marcado: marcado estructural y marcado representacional. Por un lado, el marcado estructural es el que estipula la disposición de los elementos dentro de la página, aunque no define cómo se verán éstos. De definir cómo se verán dichos elementos se encarga el marcado representacional. Es decir, este tipo de marcado se encarga del estilo sin influir en lo relacionado con la estructura del elemento.

Volviendo a las etiquetas, las cuales hemos definido anteriormente, en HTML se representan siguiendo el formato de XML, es decir, se rodean de corchetes angulares (<,>) teniendo un inicio de etiqueta representado de la forma *<etiqueta>* y un cierre de etiqueta que es de la forma *</etiqueta>* donde *etiqueta* es el nombre de la etiqueta. Es obligatorio que toda etiqueta tenga una apertura y un cierre. Además, una etiqueta puede tener atributos (para poder identificarla, de estilo y diseño, etc).

Si nos fijamos en su estructura, un documento HTML empieza y acaba con las etiquetas *<html>* y *</html>* respectivamente. Todo lo incluido entre estas dos etiquetas formará la página web. Dentro de estas dos etiquetas, se observan dos partes diferenciadas. La primera de ellas es la cabecera de la página representada por la etiqueta *<head>* y dentro de la cual va incluido el

título de la página representado por *<title>*. La segunda es el cuerpo de la página representado por la etiqueta *<body>* y dentro del cual va incluido el contenido de la página.

Es decir, el esquema básico de la estructura de un fichero HTML es el siguiente:

```
<HTML>

    <HEAD>
        Definiciones de la cabecera
    <HEAD>

  <BODY>
        Instrucciones HTML
<BODY>

<HTML>
```

Figura 1 - Estructura de un fichero HTML

2.1.1.2 CSS

CSS (*Cascading StyleSheets u Hojas de Estilo en Cascada*) es un lenguaje de estilo encargado de definir la apariencia de un documento escrito en un lenguaje de marcado (como por ejemplo HTML, XML o XHTML).

Una de las filosofías principales sobre las que está cimentado CSS es la de separar el contenido del documento de la presentación del mismo. Es decir, por un lado tendremos el contenido del documento (*por ejemplo en un fichero HTML*) y por otro lado, en un archivo .css por ejemplo, tendremos todo lo orientado con el diseño del mismo como pueden ser características como colores, fuentes, capas, espaciado entre elementos, márgenes, bordes entre otras muchas cosas. Explicado de otra forma, la página web sería lo que hay debajo (el contenido) y CSS sería una especie de filtro que hace que el contenido se muestre de una forma u otra.

Esta separación entre la estructura y la presentación es muy útil, ya que permite que sólo modificando los CSS se modifique completamente el aspecto de una página web. Esto supone una serie de ventajas. Entre ellas, podemos destacar: mayor accesibilidad al documento, mayor control de las características representacionales, reduce la complejidad de mantenimiento, permite visualizar el mismo documento en infinidad de dispositivos diferentes y, en el caso deque varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css, se reduce la complejidad y la repetición de código en la estructura del documento.

El lenguaje *CSS* es muy sencillo e intuitivo. Está basado en una serie de reglas encargadas de regular y configurar el estilo de los elementos que forman parte de los documentos a los que está vinculada dicha hoja de estilo. Cada regla consiste en un selector y una declaración, la cual va entre corchetes e incluye una propiedad del elemento y el valor a asignarle a dicha propiedad, separados por dos puntos. El selector especifica qué elementos HTML van a estar afectados por esa declaración y

la declaración es la información de estilo que indica cómo se va a ver el selector.

Cuando se crea una página web, se utiliza en primer lugar el lenguaje de marcado para "marcar" los contenidos, es decir, para designar la función de cada elemento dentro de la página (párrafo, titular, texto destacado, tabla, lista de elementos, etc).

Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento (color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc).

2.1.1.3 JavaScript

JavaScript es un lenguaje de programación interpretado, es decir, las instrucciones en código Javascript se interpretan directamente, en tiempo de ejecución, sin necesidad de que el código del programa tenga que ser compilado para utilizarlo. Funciona como una extensión de HTML. El código de un programa fuente JavaScript se inserta directamente en el documento HTML (también puede estar en un fichero aparte y ser llamado por el documento HTML) y es el propio navegador el que se encarga de ir interpretándolo.

Se usa principalmente en programación web y surgió con la necesidad de darle dinamismo e interactividad a los documentos HTML. Aunque nos lo podemos encontrar en la parte del servidor (uso menos frecuente), lo más común es trabajar con él en la parte del cliente, donde se ejecutan directamente los programas sin necesidad de acceder al servidor.

Como características de este lenguaje podemos destacar: su ligereza, sencillez, que es imperativo, estructurado y multiplataforma. Además, dispone de gran versatilidad para integrarse a la perfección tanto en cualquier sistema operativo como en la mayoría de navegadores web.

Tiene usos muy diversos, pero entre los más comunes podríamos destacar:

- Interacciones del usuario con el contenido de la página mediante eventos en botones, enlaces, bloques, acciones, etc.
- Transferir datos desde el servidor sin tener que recargar la página usando AJAX.
- Ofrecer distintos tipos de respuesta según el navegador y sistema operativo utilizado.
- Modificar código HTML en respuesta a la acción del usuario sin necesidad de establecer conexión con el servidor.
- Creación de widgets o pequeños aplicaciones como relojes, cronómetros, calculadoras, galerías de imágenes con movimiento, etc.
- Convertir las páginas en aplicaciones web con las que es posible interactuar como haríamos en una aplicación de escritorio (jugar, reproducir música y vídeo, editar imágenes, etc.).
- Validación de datos en formularios.

En nuestro proyecto, JavaScript se utiliza principalmente para la validación de datos en formularios, es decir, como verificación en el front-end de que los datos introducidos son correctos y cumplen nuestras normas de seguridad. Si los datos no son correctos, se le notifica al usuario a través de una ventana emergente y los datos no se procesan (en ningún momento llegan al servidor) quedando la aplicación a la espera de que éste introduzca unos datos válidos. Con esto, conseguimos minimizar el envío de información no válida al servidor y que nuestro servicio sea más seguro y robusto.

2.1.1.4 JSP

Aunque no pertenezca a la parte cliente, para poder explicar JSP (Java Server Page) de forma correcta, primero es necesario definir el concepto de Servlet y cómo funciona.

Un Servlet es una tecnología que permite crear aplicaciones web dinámicas, permitiendo al usuario interactuar con la aplicación. El Servlet es una extensión de los servidores orientados a petición-respuesta. Éste admite peticiones realizadas desde un navegador web, las procesa y las devuelve al navegador.

Un Servlet está realizado en Java. Está formado por una serie de métodos que generan callbacks. La forma de comunicación de los Servlets con el servidor es a través del contenedor web o motor de Servlets.

Tras la explicación de lo que es un Servlet, se procederá a explicar JSP.

JSP (Java Server Page) es un lenguaje de programacióncuyo objetivo es crear páginas web dinámicas.

JSP ofrece una forma de agregar contenido dinámico (generado por código Java) a un archivo HTML. Esto se puede lograr introduciendo el código escrito en Java (entre las etiquetas especiales <% %>) en el documento escrito en HTML. Así, cuando una página JSP es solicitada por un usuario, el código HTML pasará directamente al usuario, mientras que el código Java es procesado por el servidor web antes de enviar su resultado a dicho usuario.

Todo JSP genera un Servlet Java. Si se modifica el código de la página JSP se vuelve a generar el Servlet y se recompila automáticamente.

Para desplegar y correr JSP se requiere un servidor web compatible con contenedores Servlet como Apache Tomcat o Jetty. Este servidor web mantiene una máquina virtual de Java (JVM) en ejecución que se encarga de compilar los archivos JSP y ejecutar los Servlets. En nuestra aplicación usamos Apache Tomcat. Lo explicaremos más adelante en este mismo apartado.

No nos extenderemos más en explicar este lenguaje, sólo señalar que la aplicación ha sido desarrollada según el patrón MVC, de modo que los ficheros .jsp los usamos en la capa de la Vista del MVC, es decir, estos ficheros son utilizados como interfaz gráfica para mostrar o solicitar datos a los usuarios. Basándonos en los conceptos de dicho patrón, en la capa de la Vista no incluimos ninguna lógica de negocio, operaciones ni ningún tipo de dinamismo (excepto el que le aporte el código JavaScript), por tanto, en nuestra aplicación, dichos ficheros, salvo en algún momento puntual, realizarán prácticamente la misma función que un documento escrito en lenguaje HTML.

2.1.1.5 EL y JSTL

EL (Expression Language) es un lenguaje de propósito especial utilizado principalmente en aplicaciones webpara incrustar expresiones en páginas web.

Este lenguaje se ha utilizado en *JSP* para permitir en la Vista el uso de lógica, pero respetando el patrón MVC, es decir, sin implementar Java en la interfaz encargada de mostrar los datos al usuario. Esto lo logra realizando llamadas a la librería *JSTL (JSP Standard Tag Library) Core*. La finalidad de esto es la de construir toda la lógica de las páginas jsp de forma más rápida y eficaz, y realizar todo esto sólo haciendo llamadas a la librería JSTL, no ejecutando lógica en la capa de la Vista.

JSTL (JSP Standard Tag Library) es un conjunto de librerías de etiquetas simples y estándares que encapsulan funcionalidad, es decir, consiste en una colección de funciones de uso común, escritas en lenguaje XML y fácilmente invocadas desde páginas JSP. Esto facilita la integración de estas funciones con el lenguaje HTML. Las etiquetas JSTL están organizadas en cinco librerías:

- core: Comprende las funciones script básicas como bucles, estructuras de control y entrada/salida.
- xml: Comprende el procesamiento de lenguaje XML.
- fmt: Comprende la internacionalización y formato de valores como de moneda y fecha.
- sql: Comprende el acceso a base de datos.
- functions: Comprende las funciones relacionadas con Strings y Arrays.

Para poder usar JSTL en un proyecto web, es necesario copiar en nuestro proyecto las bibliotecas *jstl.jar* y *standard.jar*. Estos ficheros tenemos que copiarlos en la carpeta *lib* dentro del directorio *WEB-INF*.
Para poder usar una librería en una página, es necesario importarla usando su URI, y dar un nombre de prefijo, el cual lo usaremos para llamar luego a las funciones disponibles.
En nuestra página JSP lo haremos de la siguiente manera:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

2.1.2 Parte Servidor – BackEnd

2.1.2.1 Java

Podemos definir Java como un lenguaje de programación de propósito general orientado a objetos. Fue desarrollado por Sun Microsystems.

Además,podemosdecir que Java es más que un simple lenguaje de programación,es una tecnología que provee de una máquina virtual Java (Java Virtual Machine) que permite ejecutar código compilado Java, sea cual sea la plataforma que exista por debajo (tanto software como hardware).

Para definirlo de una forma más amplia, mostraremos sus características:

- **Orientado a Objetos:** Es un paradigma de programación más cercano a como tratamos las cosas en nuestra vida cotidiana. Como dice su nombre, este paradigma de programación se centra en los objetos en los cuales se combinan los datos (“estado” del objeto) y sus funciones o métodos (“comportamiento del objeto”). La POO, tratando a un objeto como una entidad genérica, nos permite reutilizar código. Otra ventaja de la POO es una gran organización y control sobre el código ya que basta con escribir una vez los métodos y las propiedades de un tipo de objeto y se pueden usar tantas veces como se quiera.

- **Es un lenguaje compilado e interpretado:**El compilador es el encargado de convertir el código fuente de un programa en un código intermedio (ByteCode). Éste es una especie de código máquina, independiente de la plataforma en que se trabaje y ejecutado/interpretado por el intérprete de Java que forma parte de la JVM.
  Es útil combinar la compilación y la interpretación ya que de esta forma se facilita la seguridad y la estabilidad y se reducen los problemas de versiones.

- **Multiplataforma o independencia de la plataforma**: Las aplicaciones desarrolladas en Java funcionan en cualquier entorno ya que no es el sistema quien las ejecuta sino la máquina virtual de Java o JVM. Así, las aplicaciones escritas en Java funcionarán igual en los distintos sistemas operativos (Windows, Unix, Mac OS…), diversos entornos de red y distintas arquitecturas hardware.

- **Distribuido**: Este lenguaje es muy útil en la creación de aplicaciones distribuidas ya que ofrece un módulo de clases destinadas a tal fin. Algunas de las funciones de estas clases son: establecer conexiones de forma remota con clientes y servidores, manejo de sockets…

  Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos.

- **Dinámico:** Se puede considerar que Java es un lenguaje dinámico en la fase de enlazado, ya que, en su ejecución, únicamente hace uso de sus clases cuando le es necesario utilizarlas. Con este dinamismo, ahorra memoria en uso y disminuye el tiempo de ejecución de la aplicación.

- **Portable:** Podemos considerar que el lenguaje Java es portable por ser transparente a la arquitectura sobre la quefunciona, es decir, los programas escritos en Java son iguales en cualquiera de las plataformas sobre las que pueden correr.

- **Robusto y fiable:** Para asegurar su robustez y fiabilidad, Java realiza una serie de verificaciones, tanto en tiempo de compilación como en tiempo de ejecución. Algunas de estas verificaciones son: Comprobación de tipos Java para detectar errores en el ciclo de desarrollo, comprobación de punteros y límites de arrays para evitar que se escriba en zonas de memoria erróneas, verificación de ByteCodes para asegurar el funcionamiento de la aplicación y uso de manejadores de excepciones para poder controlar errores en tiempo de ejecución.

  El manejo de la memoria por parte de Java y la presencia de un recolector de basura también ayudan a la robustez de este lenguaje. Esto libera a los programadores de la obligación de tener que manejar la liberación de memoria y los problemas que ello conlleva.

- **Seguridad:**Por su uso distribuido, en Java se ha prestado especial atención a la seguridad. A nivel del lenguaje, al eliminar los punteros y el casting implícito, utilizados en otros lenguajes, se evitan accesos ilegales a las zonas delicadas de memoria. En cuanto a las barreras de seguridad en el sistema de ejecución en tiempo real, podemos destacar un filtrado de verificación de los ByteCodes usado para detectar los fragmentos de código ilegal, una separación del espacio de nombres del sistema de ficheros local del de los recursos procedentes de la red (realizado por el cargador de clases), imposibilidad de abrir ningún fichero de la máquina local ni de ejecutar ninguna aplicación nativa de una plataforma.

- **Multihilo:**Al ser multihilo, Java puede llevar a cabo varias actividades simultáneamente (una actividad se ejecuta en cada hilo o proceso). Esta característica proporciona una gran mejora de rendimiento y es muy útil sobretodo en la ejecución de aplicaciones distribuidas.

- **Recolector de basura:** El recolector de basura o garbage collector se encarga de eliminar los objetos que previamente creó el programador y hacia los que ya no existe ninguna referencia. Esto se realiza con la finalidad de liberar memoria (que estará disponible para poder ser utilizada justo después de la liberación) para que ésta no se agote. El uso del recolector de basura es una gran ayuda para el programador ya que éste no tendrá la responsabilidad de encargarse de la liberación de memoria.

Podemos encontrarnos el lenguaje Java en una gran variedad de entornos de funcionamiento. Es muy usado en aplicaciones de escritorio, en aplicaciones móviles y en sistemas embebidos. También podemos encontrarlo en el navegador web en forma de applets. Éstas son unas pequeñas aplicaciones que van incrustadas en las páginas HTML y que el navegador se encarga de descargar y ejecutar. Pero, dónde más utilidad ha tenido el lenguaje de programación Java es en la parte del servidor web, lugar en el es utilizado en nuestro proyecto. Esta popularidad de Java en la parte del servidor comienza con la aparición de JSP y Servlets, que sustituyen a los CGI y otros lenguajes interpretados y se convierte en uno de los estándares de facto para el desarrollo de aplicaciones web dinámicas de servidor.

En nuestro proyecto, además de ser utilizado en los Servlets que forman parte del controlador, Java es utilizado en la capa del modelo, es decir, dónde se definen los componentes para la representación de la información.

2.1.3 Base de Datos

2.1.3.1 SQL

SQL (Structured Query Language o Lenguaje de Consulta Estructurada en su traducción al español) es un lenguaje declarativo (basado en la lógica relacional y el álgebra) específico del dominio de acceso a los sistemas de gestión de bases de datos relacionales, permitiendo especificar diversos tipos de operaciones sobre ellas.

Al ser un lenguaje declarativo, una de sus características principales es el manejo del álgebra y el cálculo relacional, los cuales, a través de una serie de comandos, cláusulas, operadores y funciones de agregado, nos dan la posibilidad de gestionar bases de datos; modificándolas, actualizándolas o efectuando consultas para obtener información de éstas.

En SQL coexistentres tipos de lenguajes: DDL (Data Definition Language o Lenguaje de Definición de Datos), DML (Data Manipulation Languaje o Lenguaje de Manipulación de Datos) y DCL (Data Control Languaje o Lenguaje de Control de Datos).

DDL es el lenguaje encargado de modificar la estructura de los objetos contemplados por la base de datos por medio de cuatro operaciones básicas. Las cuatro operaciones son: CREATE (Crear tabla), DROP (Eliminar datos y estructura de la tabla), ALTER (Modificar la estructura de la tabla) y TRUNCATE (Eliminar contenido de la tabla sin eliminar su estructura).

DML es un lenguaje que contribuye a la gestión de las bases de datos y que permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de los datos contenidos en éstas. Tiene cuatro operaciones básicas, las cuales son comúnmente definidas como CRUD (Create (INSERT),Read (SELECT), Update (Update) y Delete (DELETE)).

DCL es un lenguaje que incluye una serie de comandos SQL que permiten al administrador controlar el acceso a las bases de datos.

2.2 Diseño

2.2.1 MagicDraw UML

MagicDraw UML es una herramienta CASE (Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadora en su traducción al español). Es compatible con el estándar UML 2.3. Como una herramienta CASE que es, MagicDraw UML tiene como objetivo aumentar la productividad en el desarrollo software reduciendo los costes.

MagicDraw UML también facilita el modelado de datos y proporciona un gran mecanismo de ingeniería de código para una gran variedad de lenguajes de programación como C++, C#, Java, CORBA o .NET. Además, este programa facilita el análisis y modelado orientado a objetos de los sistemas y bases de datos.

Como ventajas de usar MagicDraw UML podemos destacar su integración en Entornos de Desarrollo (IDEs) como Eclipse IDE, Netbeans, Oracle Workshop o Sun Java Studio, disponibilidad para una gran cantidad de sistemas operativos y plataformas, posibilidad de trabajo en grupo favoreciendo el desarrollo colaborativo haciendo uso del Team Work Server, el cual permite trabajar a más de un desarrollador sobre el mismo proyecto en el mismo momento, posibilidad de visualización del proyecto desde diferentes perspectivas, una interfaz intuitiva y fácil de usar y posibilidad de generación automática de informes.

Esta herramienta es habitualmente utilizada por analistas de software, analistas de sistemas, programadores, analistas de negocio, escritores de documentación e ingenieros de gestión de calidad (QA, Quality Assurance) entre otros.

En la siguiente imagen se muestra una captura de pantalla del diseño de la aplicación web en MagicDraw UML:

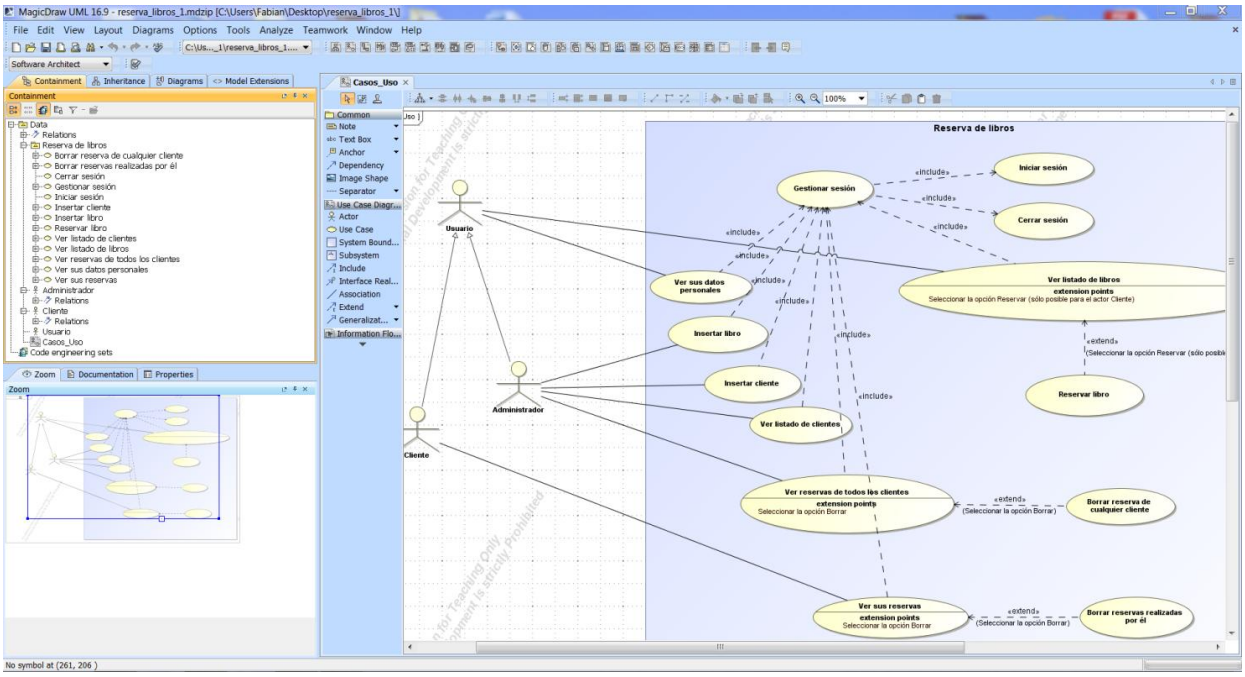

Figura 2 - Ejemplo del diseño de la aplicación en MagicDraw

2.3 Servidor:Apache Tomcat

Apache Tomcat es un contenedor web que soporta Servletsy JSPs. Está desarrollado por la Apache Software Foundation.

Tomcat puede funcionar como servidor web autónomo soportando altos niveles de tráfico y se integra muy bien en el IDE Eclipse.

Prestando atención a la estructura de Apache Tomcat, se tiene la siguiente jerarquía de directorios:

- bin: arranque, cierre, scripts y ejecutables.
- common: clases comunes que puede utilizar Catalina (contenedor de servlets) y las aplicaciones web.
- conf: ficheros XML y la correspondiente DTD para la configuración de Apache Tomcat.
- logs: informes del contenedor de servlets y de las aplicaciones.
- server: clases usadas por el contenedor de servlets.
- shared: clases compartidas por todas las aplicaciones web.
- webapps: directorio que contiene las aplicaciones web.
- work: almacenamiento temporal de ficheros y directorios.

Explicado de forma muy básica, el funcionamiento de Apache Tomcat es el siguiente:

El navegadorsolicita una página weba Apache Tomcat. Éste, delega la petición a un Servlet particular elegido de entre los Servlets que contiene.El Servletse encarga de generar el texto de la página web que se entrega a Tomcat. Finalmente, Apache Tomcat devuelve la página web al navegador que le solicitó la página.

2.4 Gestión de Base de Datos:PostgreSQL – pgAdmin III (Windows) y psql (Linux Ubuntu)

PostgreSQL es un Sistema de gestión de Bases de Datos relacional orientado a objetos y de código libre. Este sistema de gestión de Bases de Datos está manejado por el PGDG (PostgreSQL Global Development Group), una comunidad de desarrolladores que trabajan de forma altruista y libre.

Este sistema gestor de Bases de Datos presenta varias ventajas:

- Gran estabilidad sin tener caídas en años de operación con alta actividad.
- Extensibilidad pudiendo extenderse o personalizarse con poco esfuerzo y bajo coste.
- Multiplataforma.
- Facilidad de uso.
- Gran popularidad, sobretodo integrado con tecnologías web.

Además, gracias al uso de un sistema denominado MVCC (Multiversion Concurrency Control o Acceso Concurrente Multiversión en su traducción al español), PostgreSQL permite una alta concurrencia pudiendo modificarse una tabla a la vez que ésta está siendo consultada por otros usuarios sin producirse ningún tipo de bloqueo.

Para facilitarnos la interacción con PostgreSQL, en el desarrollo de nuestra Base de Datos estamos

usando una aplicación gráfica desde la cual manejamos este gestor de Bases de Datos. Esta herramienta se llama PgAdmin III. Es de licencia OpenSource y es la herramienta oficial de PostgreSQL. Desde ella se puede realizar de forma gráfica todo lo que PostgreSQL nos permite a través de comandos. Es muy fácil e intuitiva de usar.

Aunque nosotros sólo la estamos usando en Windows (para gestionar PostgreSQL desde Linux Ubuntu usamos la consola de comandos de PostgreSQL llamada psql) está disponible para todas las plataformas.

En la siguiente imagen se muestra una captura de pantalla de la gestión de la Base de Datos de la aplicación a través de la interfaz gráfica pgAdmin III:

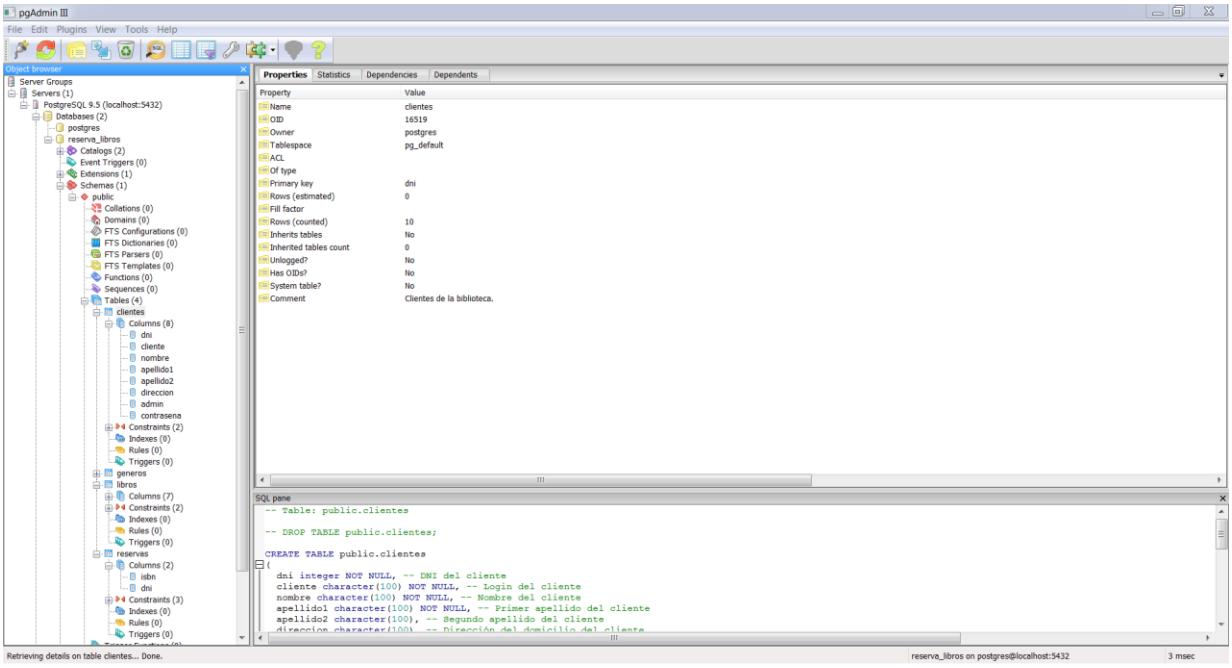

Figura 3 - Ejemplo del diseño de la Base de Datos de la aplicación en PostgreSQL (pgAdmin III)

2.5 IDE (Entorno de Desarrollo Integrado):Eclipse

Eclipse es una plataforma de desarrollo de software de código abierto (diseñado por IBM y cuyo código fuente fue puesto a disposición de los usuarios) basada en Java. De forma básica, podríamos definir Eclipse como un marco sobre el cual se pueden montar herramientas de desarrollo para cualquier lenguaje, extendiendo esta plataforma de desarrollo a través de la implementación de plugins. La finalidad de lo anteriormente citado es la de ayudar al usuario a programar de manera rápida y fácil.

Este entorno de desarrollo presenta las siguientes características:

- Multiplataforma, estando disponible para una gran variedad de plataformas.
- Soporte para diversos lenguajes de programación, siendo los más usados Java, C, C++, JSP, sh, Perl y PHP.
- Integración con Sistemas de Control de Versiones como GitHub, Subversion (SVN) o CVN (Concurrent Version System).
- Edición de textos con reconocimiento de sintaxis. Esta característica sirve de guía al programador, así como de indicador de errores antes de la compilación.

- Integración con una gran variedad de frameworks.
- Ampliación de las funcionalidades a través de extensiones y plugins.
- Depurador de errores.
- Módulo de pruebas unitarias a través de JUnit.
- Integración con Ant, útil en la realización de tareas mecánicas y repetitivas. Su mayor uso es en la automatización de la compilación.
- Refactorización o conocido de forma informal como "limpieza de código", con la finalidad de hacer el código más claro, manejable, fácil de mantener y consistente.

En Eclipse, una perspectiva es un contenedor sobre el cual el programador monta las distintas herramientas y vistas que necesita.Hay perspectivas configuradas por defecto. Con la finalidad de facilitarnos el trabajo en el desarrollo de la aplicación, hemos usado la perspectiva "Java", ya que el lenguaje base de programación de dicha aplicación es Java.

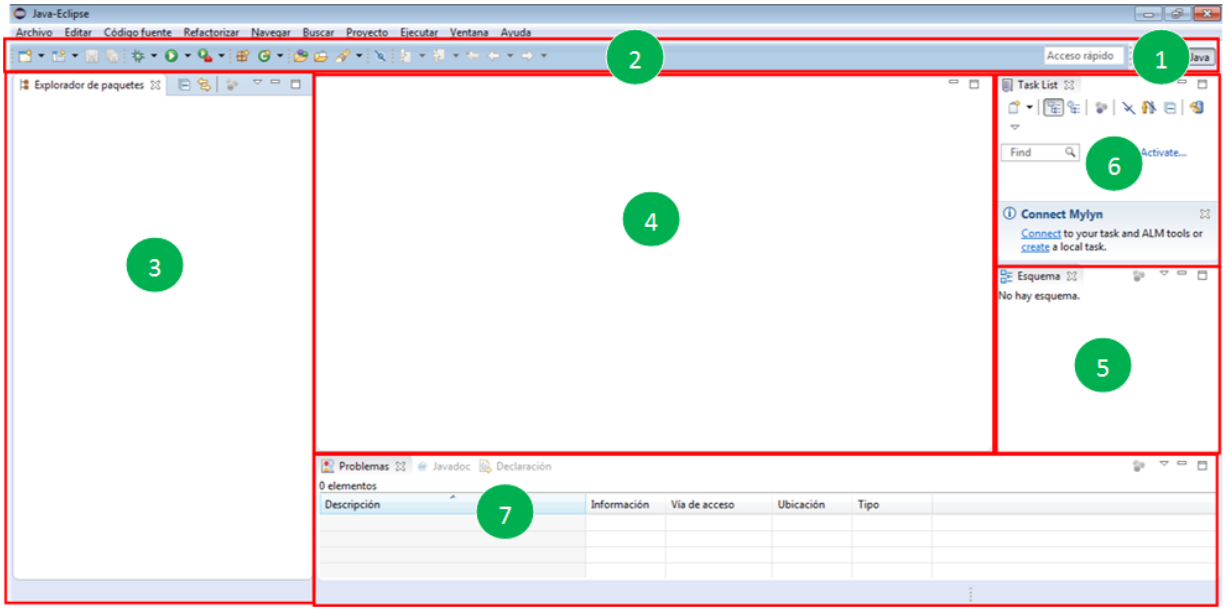

Figura 4 - Perspectiva Java en Eclipse

En la imagen, podemos ver la perspectiva "Java" por defecto. Ésta consta de:

1. **Perspectivas:** En esta zona, se pueden ver las perspectivas usadas en la aplicación. Para desarrollar el TFG, además de la perspectiva "Java", se usa la perspectiva "Debug".
2. **Barra de herramientas.**
3. **Explorador de paquetes:** Permite navegar por los diferentes proyectos y seleccionar los elementos que deseemos abrir en un editor.
4. **Editor de Java**: Editor con corrección de sintaxis y ayuda a la programación.
5. **Esquema**: Se muestra la estructura del archivo de código fuente seleccionado actualmente.
6. **Task list**: Permite crear listas de tareas asociadas a un cierto archivo.
7. **Problemas**: Aparece en la barra inferior. Muestra los errores y mensajes de advertencia relacionados con los proyectos. Los mensajes mostrados se pueden configurar en el menú desplegable de la vista.

En el desarrollo del TFG, en la barra inferior también disponemos, entre otras pestañas, de "Servidores", "Consola" y "Depurador".

Un plugin de Eclipse es un módulo que se monta en Eclipse y añade a la plataforma de trabajo una funcionalidad específica. La gran ventaja que presenta el trabajar con plugin es la optimización de la memoria y de los recursos, ya que sólo se instala lo que se va a usar.

Agrupándolos por funcionalidad, los plugins que se han usado en el desarrollo de la aplicación son (se ha considerado sólo explicar los plugins que actúan de forma directa sobre la aplicación y que son diferentes de otras aplicaciones):

- **Eclipse Java Development Tools o JDTy Java EE:** Es el conjunto de plugins básico de Java y proporciona multitud de funcionalidades como corrección de errores, ayudas y sugerencias en la programación, compilación y ejecución de código.
- **Apache Tomcat:** Este grupo de plugins es añadido a Eclipse al realizar la instalación del servidor Apache Tomcat. Entre los plugins de este grupo podemos nombrar algunos como Apache Web Services, Apache HTTPCLient y Apache Tomcat Support .
- **PostgreSQL:** Este grupo de plugins se encarga de todo lo relacionado con la base de datos.
- **Aptana:** Este grupo de plugins es un asistente de código HTML, CSS y Javascript.
- **Eclipse GITHub y Git Team Provider:** Se utiliza para trabajar con el sistema de control de versiones Git. Hasta donde llega este proyecto no se usará, pero se ha probado e investigado con él y se espera que sea utilizado en un plazo corto de tiempo.

Además de los plugins anteriormente definidos, a continuación se hará una reseña especial al driver JDBC (Java DataBase Conectivity).

JDBC (Java DataBase Conectivity) es la interfaz que proporciona Java para la conexión con las Bases de Datos. Explicado de forma básica, su uso es el de establecer y mantener una conexión con la Base de Datos, es decir, el de hacer la función de *conector*entre las clases Java y la Base de Datos.

JDBC puede ser utilizado con cualquier Base de Datos relacional. En el proyecto, como se ha comentado anteriormente, será usado con PostgreSQL.

2.6 Patrones de Diseño Software

2.6.1 Cliente – Servidor

Explicado de forma sencilla, el modelo Cliente-Servidor se puede definir como una arquitectura distribuida en la que un cliente solicita las capacidades ofrecidas por un servidor, normalmente intercambiando mensajes de solicitud/respuesta o invocando métodos de forma remota.

Por tanto, este patrón de diseño software define una comunicación basada en una serie de solicitudes y respuestas entre Cliente y Servidor. Para entender de forma más sencilla lo anteriormente definido, se procederá a definir los actores participantes en el modelo, así como la comunicación existente entre ellos.

Los actores del modelo cliente – servidor se pueden diferenciar en:

- Cliente: Actor encargado de solicitar servicios a uno o más servidores. Es el encargado de iniciar la comunicación, normalmente es dirigido por el usuario. Un ejemplo de cliente es el navegador web.
- Servidor: Actor encargado de ofrecer un conjunto de servicios a los clientes y de responder a las solicitudes de éstos. Está en constante ejecución. Hay diferentes categorías de servidores: de archivos, de bases de datos, de impresión, web, de correo…
  Como hemos mencionado en apartados anteriores, en la aplicación se ha usado el servidor Web Apache Tomcat, de modo que se utiliza HTTP como protocolo de comunicación entre cliente y servidor.

Además de los actores explicados anteriormente, a continuación se va a explicar un elemento que juega un papel fundamental en el desarrollo del patrón MVC. Este elemento es la Interfaz de Usuario, es decir, el medio de comunicación entre el usuario y la aplicación. Por parte del usuario, a través de la interfaz, éste realiza consultas e introduce datos en la aplicación y, por parte de la aplicación, con la ayuda de la interfaz, ésta responde a las distintas solicitudes del usuario, mostrando los datos solicitados de la forma requerida.

Paso a paso la comunicación entre cliente y servidor se realiza de la siguiente forma:

1. El cliente realiza una solicitud al servidor.
2. El servidor procesa la petición que le ha llegado por parte del cliente.
3. El servidor envía la respuesta al cliente.
4. El cliente procesa la respuesta anteriormente enviada por el servidor.

Figura 5 - Proceso de comunicación entre Cliente y Servidor

En cuanto a las ventajas que proporciona el uso de este modelo de diseño software, podemos destacar:

- **Control centralizado de los datos:** El servidor es el encargado de controlar el acceso y la integridad de los datos. Aunque esta característica puede presentar en el servidor un cuello de botella que congestione el tráfico, presenta las ventajas de que cualquier cliente dañino o defectuoso no consiga dañar el sistema, así como una gran facilidad en las tareas de actualización de los datos.
- **Escalabilidad y fácil mantenimiento:** Al funcionar de forma independiente el cliente y el servidor (sistemas distribuidos), cualquier elemento puede ser reparado, actualizado, reemplazado, así como añadir nuevos nodos, sin afectar a los demás.

- **Independencia de desarrollo**: Siempre que la Interfaz se respete, Cliente y Servidor pueden evolucionar por separado. Se pueden reemplazar de forma totalmente transparente al otro e incluso puede haber diferentes implementaciones (versiones) de cada uno.
  Así, publicada la interfaz de un servicio, diferentes desarrolladores pueden implementar clientes "a medida" y, del mismo modo, el servidor puede reemplazarse cuando sea necesario sin tener que modificar los clientes.

2.6.2 Modelo – Vista – Controlador

Modelo – Vista – Controlador es un patrón de diseño software basado en la separación de la lógica de negocio y lo relacionado con los datos de una aplicación, de la interfaz de usuario y del gestor de las comunicaciones y los eventos. Esto se realiza mediante un modelo de tres capas diferentes, las cuales serán explicadas a continuación:

- **Modelo:** Representa la parte de la aplicación encargada del manejo de datos. Entre otras funciones, el modelo se encarga de la recuperación de los datos, convirtiéndolos en unidades "entendibles" para la aplicación, así como del almacenamiento de éstos en la base de datos, de su procesamiento y validación.
- **Vista:** Es la parte encargada de la representación visual de los datos, es decir, de todo lo relacionado con la interfaz gráfica. Es la capa que está en relación directa con el usuario.
- **Controlador:** Este componente se encarga de representar la lógica, siendo considerado como una especie de "intermediario" entre el modelo y la vista, es decir, sirve de enlace entre la vista y el modelo. Se encarga de controlar a qué parte del modelo solicitar los datos y qué vista los muestra al usuario.

A continuación, a modo de ejemplo, detallaremos un proceso de comunicación normal entre las tres capas para mostrar a un usuario una página web tras acceder a ella:

1. El usuario intenta acceder al sitio web. Para ello, realiza una solicitud a dicho sitio web. Esta solicitud llega al controlador.
2. El controlador le solicita al modelo los datos necesarios para mostrar la página.
3. El modelo consulta los datos a la base de datos y devuelve al controlador dichos datos de una forma en la que puedan ser entendidos por este último.
4. El controlador invoca a la vista correspondiente, enviándole los datos anteriormente solicitados al modelo.
5. La vista se encargará de mostrar dichos datos al usuario.

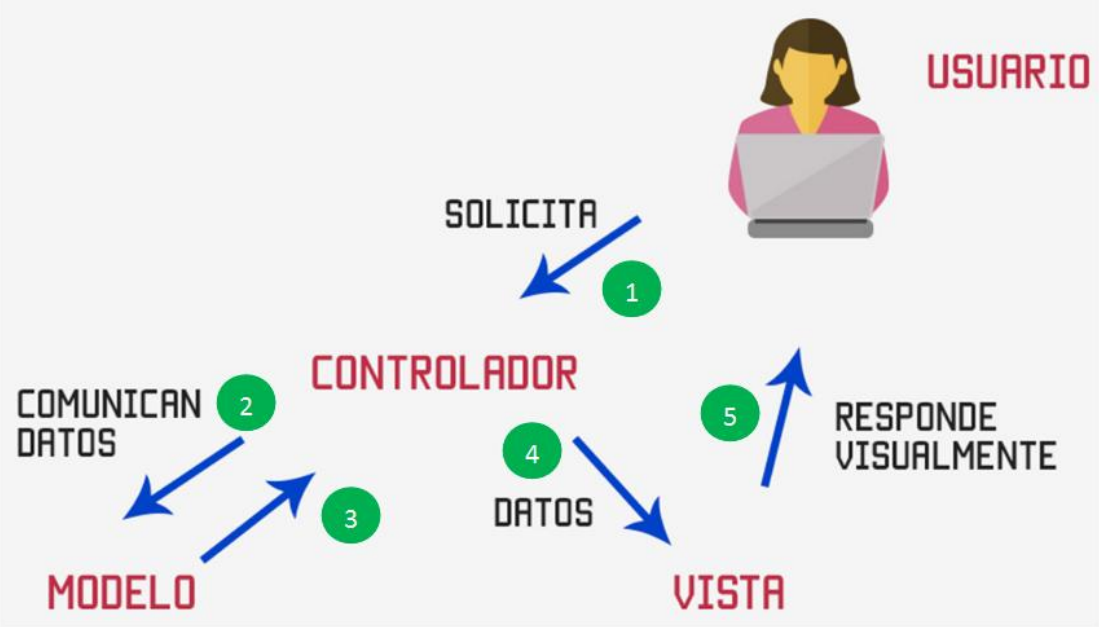

Figura 6 - Proceso de comunicación entre Modelo, Vista y Controlador

En cuanto a las ventajas que proporciona el uso de este modelo de arquitectura software, podemos destacar:

- **Mayor escalabilidad:** Al separase la lógica de negocio y el manejo de los datos del diseño y la representación de los mismos, las aplicaciones ganan en escalabilidad.
- **Menos alteraciones antes cambios:**Debido a la independencia entre los tres componentes del MVC, se pueden realizar modificaciones en una capa sin afectar al resto. Por ejemplo, se puede modificar la vista y la forma de mostrar los datos al usuario sin necesidad de modificar la obtención de dichos datos de la base de datos.
- **Mantenimiento ante errores más rápido y sencillo:** Por el motivo mencionado anteriormente, una aplicación implementada bajo el patrón MVC facilita el mantenimiento ante errores.
- Posibilidad de agregar **múltiples representaciones de los datos**, así como **múltiples vistas**.
- **Código más adaptable:**En el caso de proyectos donde hay varios desarrolladores, el cumplir todos la metodología de programar siguiendo el patrón MVC hace que el código sea más entendible entre estos, pudiendo uno continuar el trabajo de otro de forma relativamente sencilla.

3 DESARROLLO DEL PROYECTO

3.1 Objetivos

El objetivo principal es el de diseñar y desarrollar una aplicación que englobe la mayor cantidad posible de los conceptos de diseño y programación que los alumnos de GITT estudian en las asignaturas de Fundamentos de Aplicaciones y Servicios Telemáticos e Ingeniería del Software, así como el uso de las herramientas y lenguajes de programación que en ellas son impartidos.Una aplicación que, como comentamos en el apartado "Motivación", tiene que estar bien estructurada y servir de herramienta y marco de trabajo para las prácticas de las asignaturas en las que ésta sea utilizada. Además, esta aplicación debe hacer uso, de forma clara y sencilla, de paradigmas de programación que sean de utilidad para llevar a la práctica lo explicado en la parte teórica.

Haciendo referencia nuevamente al uso de la aplicación tanto en la asignatura Fundamentos de Aplicaciones y Servicios Telemáticos como en Ingeniería del Software, esta aplicación servirá de nexo de unión entre ambas, facilitando a los alumnos la adaptación a ellas y el aprendizaje de los conceptos transversales que en ellas se estudian.

Con la idea de no romper con las aplicaciones que se encuentran en uso en las prácticas de la asignatura Ingeniería del Software, se decide continuar con la misma temática, una aplicación que se base en una biblioteca y la gestión de la misma, en cuanto a libros y usuarios se refiere. Además, dicha temática resulta de gran utilidad para poder implementar un modelo que haga uso de las funciones CRUD de forma sencilla.

Con el uso de esta aplicación, los alumnos podrán:

- Conocer varios modelos de diseño y arquitectura software como los patrones Cliente – Servidor o Modelo – Vista – Controlador.
- Analizar el ciclo de vida de una aplicación.
- Manejar varias clases y cómo éstas se interrelacionan.
- Aprender acerca de la gestión de datos y manejar un gestor de base de datos concreto.
- Perfeccionar el manejo de varios lenguajes de programación, tanto del lado del cliente como del servidor.
- Ser capaces de depurar código para detectar errores u optimizarlo.

3.2 Análisis – Requisitos

3.2.1 Contexto

Aunque el objetivo de la aplicación es puramente el de la docencia, se procederá a explicar brevemente el uso real que podría tener dicha aplicación.

La aplicación consiste en un software enfocado en la reserva de libros por parte de los usuarios de la Biblioteca.

Estas reservas podrán ser realizadas por el usuario desde cualquier equipo con acceso a Internet. Tras llevarse a cabo, el usuario que realizó la reserva podrá recoger en el mostrador de la Biblioteca el

libro reservado anteriormente.

También podrán realizarse reservas en el mostrador de la Biblioteca, presentando el usuario su DNI. Para ello, el/la Bibliotecario/a entrará en la aplicación con las credenciales del usuario en concreto y realizará la reserva de libro deseado.

3.2.2 Actores

Se pueden definir a los actores como las entidades que interactúan con la aplicación. En función de los diferentes roles que pueden adquirir los usuarios de la aplicación, podemos distinguir los siguientes actores:

- **Cliente:** Representa a la persona que utiliza los libros de la Biblioteca.
  Sus requisitos principales están relacionados con la gestión del préstamo de libros. En concreto, este actor puede realizar reservas sobre los libros disponibles para, posteriormente, recogerlos en el mostrador de la Biblioteca.
- **Administrador:** Usuario encargado de la gestión de la aplicación y de los datos asociados a ella.
  Debido a su rol como gestor de la aplicación, este actor tiene acceso a la mayoría de las funcionalidades de la aplicación. Entre las distintas funciones de las que dispone este usuario, podemos destacar la visualización e inserción tanto de libros como de clientes o la visualización de las reservas de todos los clientes y el borrado de éstas.

  El/la Bibliotecario/a dispondrá de una cuenta de usuario con rol *Administrador*. Le será de gran utilidad para gestionar libros y clientes, así como para obtener las credenciales de los usuarios a los que les tenga que realizar reservas de forma presencial en el mostrador de la Biblioteca.
- **Usuario:** Usuario abstracto encargado de englobar a los dos diferentes actores.
  Este usuario es puramente teórico y es usado para definir las funciones comunes a los diferentes actores.

3.2.3 Requisitos funcionales

3.2.3.1 Casos de Uso

Antes de explicar los distintos Casos de Uso, será de gran utilidad visualizar su diagrama de Casos de Uso. Éste se muestra a continuación:

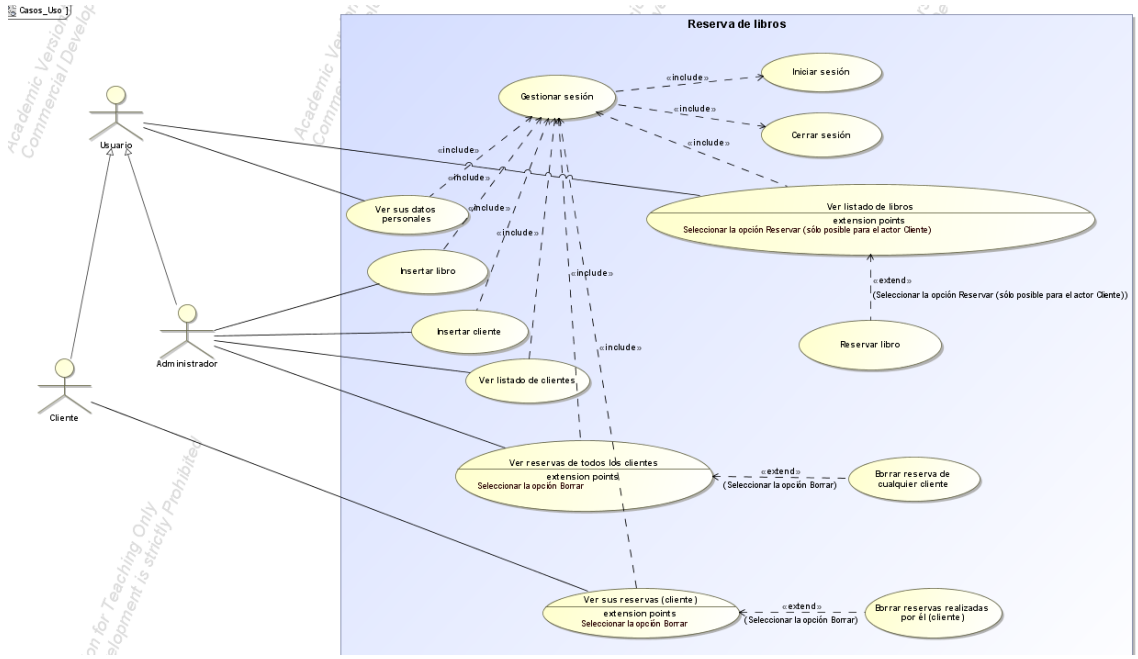

Figura 7 - Diagrama de Casos de Uso de la aplicación

Tras esto, se detallan de forma individual los distintos Casos de Uso:

CU-001	**Gestionar sesión**
Versión	1.0 (15/09/2016)
Da soporte a	•*CU-004 Ver sus datos personales* • *CU-005 Ver listado de clientes* • *CU-006 Ver listado de libros* •*CU-007 Ver reservas de todos los clientes* • *CU-008 Ver sus reservas (cliente)* • *CU-012 Insertar libro* • *CU-013 Insertar cliente*
Utiliza	•*CU-002 Iniciar sesión* • *CU-003 Cerrar sesión*
Actor	*Usuario*
Precondición	Para poder utilizar la aplicación, es necesario para el usuario disponer de una sesión. Además, el usuario, con cualquier rol, necesita de un menú que englobe todas las opciones a elegir en la aplicación y con el que se pueda mover por ellas.
Descripción	Cada vez que un usuario entra en la aplicación a través del *CU-002 Iniciar sesión*, se inicia una sesión. Ésta se mantiene mediante el *CU-001 Gestionar sesión* y, cuando el usuario desea salir de la aplicación, se cierra mediante el *CU-003 Cerrar sesión.* Además de mantener la sesión, el *CU-001 Gestionar sesión* realiza lo siguiente: En cualquier pantalla de la aplicación, excepto en la pantalla de inicio, aparecerá un menú de selección que permita moverse por las distintas páginas. Este menú permitirá acceder a las distintas funcionalidades de la aplicación y será

	diferente dependiendo del rol de usuario que tenga iniciada la sesión. Para un usuario Cliente el menú mostrado será: o Listado de las reservas en activo realizadas por él. o Listado de libros. o Sus datos personales. o Cerrar sesión. Para un usuario Administrador el menú mostrado será: o Formulario para insertar un nuevo libro. o Formulario para insertar un nuevo usuario. o Listado de reservas actuales de todos los usuarios. o Listado de clientes. o Listado de libros. o Sus datos personales. o Cerrar sesión. Además, será necesario desde la autenticación del usuario en la aplicación, que se le apliquen los privilegios acordes a sus rol. Para esto, el sistema debe ser capaz de comprobar el rol de cada usuario.
Postcondición	Desde su autenticación, el usuario tiene una sesión abierta , la cual se mantiene hasta que el usuario decide salir de la aplicación, así como disponibles las funcionalidades de la aplicación acordes a su rol. El usuario puede acceder y hacer uso de cada una de las funcionalidades habilitadas para él sin ningún tipo de problema.
Comentarios	-

Tabla 1 - CU-001 Gestionar sesión

CU-002	**Iniciar sesión**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-001 Gestionar sesión.* • *ReqInf-001 Cliente.* • *Restr-001 Campos Autenticación No Vacíos.* • *Restr-005 Cliente.* • *Restr-010 Contraseña.*
Actor	*Usuario*
Precondición	El usuario de la biblioteca, con cualquier rol, tiene pensado acceder a la aplicación.
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario intente iniciar sesión en la aplicación.
Secuencia normal	1. El usuario procede a autenticarse introduciendo sus credenciales (usuario y contraseña). 2. Se cumple la restricción *Restr-001 Campos Autenticación No Vacíos*. 3. Se cumple la restricción *Restr-005 Cliente*. 4. Se cumple la restricción *Restr-010 Contraseña*. 5. El sistema comprueba en la Base de Datos si los datos introducidos por el usuario corresponden a un usuario correcto. 6. El usuario accede a la aplicación. 7. Al acceder a ésta, comienza el caso de uso *CU-004 Ver sus datos personales*.

Postcondición	El usuario de la biblioteca accede con éxito a la aplicación.
Excepciones	*1.* Si no se cumple la restricción *Restr-001 Campos Autenticación No Vacíos* debido a que el campo Cliente está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo Cliente no puede estar vacío. E2. No permitirá al usuario acceder al sistema en ese momento. E3. Da la opción al usuario de volver a introducir nuevamente los datos. *2.* Si no se cumple la restricción *Restr-001 Campos Autenticación No Vacíos* debido a que el campo *Contraseña* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo Contraseña no puede estar vacío. E2. No permitirá al usuario acceder al sistema en ese momento. E3. Da la opción al usuario de volver a introducir nuevamente los datos *3.* Si no se cumple la restricción *Restr-005 Cliente*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo Cliente tiene que tener entre 5 y 12 caracteres. E2. No permitirá al usuario acceder al sistema en ese momento. E3. Da la opción al usuario de volver a introducir nuevamente los datos *4.* Si no se cumple la restricción *Restr-010 Contraseña*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Contraseña* tiene que tener entre 5 y 12 caracteres. E2. No permitirá al usuario acceder al sistema en ese momento. E3. Da la opción al usuario de volver a introducir nuevamente los datos *5.* Si los datos introducidos en el campo *Cliente* o *Contraseña* cumplen con las especificaciones anteriores, pero no son correctos: E1. Se cargará la misma página indicando al usuario que los datos introducidos en los campos *Usuario* o *Contraseña* no son correctos e invitándolo a introducir los datos correctamente. E2. No permitirá al usuario acceder al sistema en ese momento. E3. Da la opción al usuario de volver a introducir nuevamente los datos
Comentarios	El número máximo de intentos no ha sido considerado como un punto relevante, por lo que no ha sido tenido en cuenta.

Tabla 2 - CU-002 Iniciar sesión

CU-003	**Cerrar sesión**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-001 Gestionar sesión*
Actor	*Usuario*
Precondición	Cualquier tipo de usuario con una sesión abierta desea cerrarla.
Descripción	Para cerrar la sesión: 1. El usuario en cuestión pulsará el botón "Salir" del menú. 2. La sesión abierta de dicho usuario es cerrada. 3. Se produce una redirección hacia la página de inicio para que otro usuario pueda acceder a la aplicación.

Postcondición	La sesión abierta del usuario es cerrada, saliendo éste de la aplicación.
Comentarios	-

Tabla 3 - CU-003 Cerrar sesión

CU-004	**Ver sus datos personales**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-001 Gestionar sesión* • *ReqInf-001 Cliente*
Actor	*Usuario*
Precondición	Cualquier tipo de usuario quiere acceder a ver sus datos personales.
Descripción	El usuario puede acceder a la visualización de los datos de tres formas: 1. Tras iniciar sesión, es decir, tras autenticarse. 2. Al pulsar en el botón "Datos" del menú. 3. Al pulsar sobre el logotipo del departamento de Ingeniería Telemática que está en la cabecera de la página (una vez que el usuario está dentro de la aplicación). Cuando el usuario intenta acceder a sus datos personales, el sistema realiza una consulta a la Base de Datos que le devuelve los datos solicitados.
Postcondición	El usuario de la biblioteca visualiza por pantalla sus datos personales.
Comentarios	No se ha considerado introducir la opción de que cada usuario modifique sus propios datos debido al carácter puramente académico que inicialmente tendrá la aplicación.

Tabla 4 - CU-004 Ver sus datos personales

CU-005	**Ver listado de clientes**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-001 Gestionar sesión* • *ReqInf-001 Cliente*
Actor	*Administrador*
Precondición	El usuario de la biblioteca con rol *Administrador* quiere ver el listado de clientes registrados en la aplicación.
Descripción	Cuando el usuario Administrador desee ver el listado de clientes puede realizarlo seleccionando la opción del menú Listado de clientes. Tras esto, el sistema realizará una consulta a la Base de Datos y mostrará por pantalla el listado de clientes. Este listado devolverá todos los clientes con todos los datos referentes a cada uno de ellos que se encuentran en la Base de Datos, sin excepción alguna, según lo especificado en el *ReqInf-001 Cliente*.
Postcondición	El *Administrador* visualiza un listado de todos los clientes registrados en la aplicación.
Comentarios	Se ha decidido no añadir la opción de que el *Administrador* pueda modificar o

	eliminar clientes debido a que esto sería replicar lo que ya se realiza con las reservas. Se deja como sugerencia para posteriores modificaciones o para que los alumnos lo puedan realizar para asentar su aprendizaje.

Tabla 5 - CU-005 Ver listado de clientes

CU-006	**Ver listado de libros**
Versión	1.0 (15/09/2016)
Da soporte a	• *CU-011 Reservar libro*
Utiliza	• *CU-001 Gestionar sesión* • *ReqInf-002 Libro*
Actor	*Usuario*
Precondición	Cualquier usuario de la biblioteca quiere ver el listado de todos los libros que contiene la aplicación.
Descripción	Cuando cualquier usuario autenticado en el sistema desee ver el listado de libros puede realizarlo eligiendo la opción del menú *Listado de libros*. Tras esto, el sistema realizará una consulta a la Base de Datos y mostrará por pantalla el listado de libros. Este listado devolverá todos los libros que se encuentren en la Base de Datos según lo explicado en el *ReqInf-002 Libro*. Además, dependiendo del rol del usuario, para cada libro aparecen distintas opciones. 1. Si el usuario tiene rol de cliente, aparecen, para cada libro: a) *Reservado*: Esto ocurre en el caso de que el libro se encuentre reservado por éste u otro cliente. No permite poder reservar el libro. b) *Reservar*: Elemento que indica que el libro está disponible y que, pichando sobre él, procederá a realizar la reserva del artículo en cuestión. 2. Si el usuario tiene rol de administrador, éste no tiene el privilegio de poder reservar un libro. Para cada libro, este usuario se encuentra con las siguientes opciones: a) *Reservado*: Esto ocurre en el caso de que el libro se encuentre reservado por cualquier cliente. b) *Disponible*: Esto ocurre en el caso de que el libro no se encuentre reservado por ningún cliente.
Postcondición	El usuario visualiza un listado de todos los libros registrados en la aplicación y, en el caso de que su rol sea Cliente, puede realizar una reserva de cualquier libro que se encuentre disponible.
Comentarios	-

Tabla 6 - CU-006 Ver listado de libros

CU-007	**Ver reservas de todos los clientes**

Versión	1.0 (15/09/2016)
Da soporte a	• *CU-009 Borrar reserva de cualquier cliente.*
Utiliza	• *CU-001 Gestionar sesión* • *ReqInf-001 Cliente* • *ReqInf-002 Libro*
Actor	*Administrador*
Precondición	El usuario de la biblioteca con rol *Administrador* quiere ver el listado de reservas vigentes de todos los clientes.
Descripción	Cuando el usuario *Administrador* desee ver las reservas actuales de cada cliente, puede realizarlo seleccionando la opción del menú *Reservas*. Tras esto, el sistema realizará una consulta a la Base de Datos y mostrará por pantalla las reservas que cada cliente tenga actualmente. Por pantalla se mostrará una tabla con todos los elementos relativos al libro reservado, según lo especificado en el *ReqInf-002 Libro* y el DNI del cliente que tiene realizada la reserva, según lo especificado en el *ReqInf-001 Cliente*. Además, el *Administrador* también dispondrá, por cada libro reservado, de un botón para poder eliminar la reserva del cliente sobre dicho libro. Si se realizase esta acción, se pasaría al *CU-009 Borrar reserva de cualquier cliente*.
Postcondición	El *Administrador* visualiza un listado de todos los libros reservados y los clientes que poseen dichas reservas.
Comentarios	-

Tabla 7 - CU-007 Ver reservas de todos los clientes

CU-008	**Ver sus reservas (cliente)**
Versión	1.0 (15/09/2016)
Da soporte a	• *CU-010 Borrar reservas realizadas por él (cliente).*
Utiliza	• *CU-001 Gestionar sesión* • *ReqInf-002 Libro*
Actor	*Cliente*
Precondición	El usuario de la biblioteca con rol *Cliente* quiere ver el listado de las reservas que tiene realizadas en la actualidad.
Descripción	Cuando el usuario Cliente desee ver las reservas que tiene en su poder, puede realizarlo seleccionando la opción del menú *Mis Reservas*. Tras esto, el sistema realizará una consulta a la Base de Datos y mostrará por pantalla las reservas que tiene dicho cliente. Por pantalla se mostrará una tabla con todos los elementos relativos al libro reservado, según lo especificado en el *ReqInf-002 Libro*. Además, el cliente también dispondrá, por cada libro reservado, de un botón para poder eliminar la reserva que tiene sobre dicho libro. Si se realizase esta acción, se pasaría al *CU-009 Borrar reservas realizadas por él (cliente)*.
Postcondición	El *Cliente* visualiza un listado de todos los libros que tiene reservados en la actualidad.

Comentarios	-

Tabla 8 - CU-008 Ver sus reservas (cliente)

CU-009	**Borrar reserva de cualquier cliente**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-007 Ver reservas de todos los clientes* • *ReqInf-001 Cliente* • *ReqInf-002 Libro*
Actor	*Administrador*
Precondición	El usuario *Administrador* desea eliminar la reserva de un cliente para un libro.
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario intente borrar la reserva.
Secuencia normal	1. El usuario *Administrador* se encuentra visualizando el listado de reservas de todos los clientes, según lo especificado en el *CU-007 Ver reservas de todos los clientes*. 2. El *Administrador* decide eliminar una reserva. 3. Este usuario selecciona la opción *Borrar* con el fin de eliminar la reserva de un libro concreto para un cliente concreto. 4. El sistema procede a eliminar la reserva. 5. La reserva se ha borrado correctamente. 6. Al usuario le aparece una nueva página indicándole que la reserva del libro "XXXX" con ISBN "YYYY" se ha eliminado correctamente.
Postcondición	El *Administrador* de la biblioteca borra con éxito la reserva de un cliente para un libro y éste recibe la notificación del sistema.
Excepciones	El sistema no ha podido borrar la reserva correctamente. La reserva no se elimina de la Base de Datos del sistema. Al Administrador se le redirecciona a otra página, la cual le indica que no se ha podido borrar la reserva.
Comentarios	-

Tabla 9- CU-009 Borrar reserva de cualquier cliente

CU-010	**Borrar reservas realizadas por él (cliente)**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-008 Ver sus reservas (cliente)* • *ReqInf-001 Cliente* • *ReqInf-002 Libro*
Actor	*Cliente*
Precondición	El usuario *Cliente* desea eliminar su reserva en un libro.

Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario intente borrar la reserva.
Secuencia normal	1. El usuario *Cliente* se encuentra visualizando una tabla con sus reservas actuales, según lo especificado en el *CU-008 Ver sus reservas (cliente)*. 2. El *Cliente* decide eliminar una reserva. 3. Este usuario selecciona la opción de *Borrar* con el fin de eliminar la reserva de un libro concreto. 4. El sistema procede a eliminar la reserva. 5. La reserva se ha borrado correctamente. 6. Al usuario le aparece una nueva página indicándole que la reserva del libro "XXXX" con ISBN "YYYY" se ha realizado correctamente.
Postcondición	El cliente borra con éxito la reserva para un libro que tenía previamente reservada y recibe un mensaje de éxito como notificación.
Excepciones	*1.* El sistema no ha podido borrar la reserva correctamente. *2.* La reserva no se elimina de la Base de Datos del sistema. *3.* Al *Cliente* se le redirecciona a otra página, la cual le indica que no se ha podido borrar la reserva.
Comentarios	-

Tabla 10 - CU-010 Borrar reserva realizadas por él (cliente)

CU-011	**Reservar libro**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-006 Ver listado de libros* • *ReqInf-001 Cliente* • *ReqInf-002 Libro*
Actor	*Cliente*
Precondición	El usuario *Cliente* desea realizar la reserva de un libro disponible.
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario intente borrar la reserva.
Secuencia normal	1. El usuario *Cliente* se encuentra visualizando el listado de libros, según lo indicado en el *CU-006 Ver listado de libros*. 2. El *Cliente* decide realizar una reserva de un libro disponible. Un libro disponible para ser reservado es aquel que no se encuentra reservado en el momento en que se quiere realizar la reserva, es decir, el que en el último campo de la tabla aparece la opción *Reservar*. 3. Este usuario selecciona la opción *Reservar* con el fin de reservar el libro en cuestión. 4. El sistema procede a realizar la reserva. 5. La reserva se ha realizado correctamente. 6. Al usuario le aparece una nueva página indicándole que la reserva del libro "XXXXXXXXXX" con ISBN "YYYYYYYYYY" se ha realizado correctamente.
Postcondición	El *Cliente* realiza la reserva de un libro concreto con éxito y recibe un mensaje de éxito como notificación.

Excepciones	*1.* El sistema no ha podido realizar la reserva correctamente. *2.* La nueva reserva no es añadida en la Base de Datos del sistema. *3.* Al *Cliente* se le redirecciona a otra página, la cual le indica que no se ha podido realizar la reserva del libro "XXXXXXXXXX" con ISBN "YYYYYYYYYY".
Comentarios	-

Tabla 11 - CU-011 Reservar libro

CU-012	**Insertar libro**
Versión	1.0 (15/09/2016)
Utiliza	• *CU-001 Gestionar sesión.* • *ReqInf-002 Libro.* • *Restr-003 Campos Libro No Vacíos.* • *Restr-011 ISBN.* • *Restr-012 Título.* • *Restr-013 Autor.* • *Restr-014 Editorial.* • *Restr-015 Año de publicación.*
Actor	*Administrador*
Precondición	El usuario Administrador desea realizar la inserción de un nuevo libro en el sistema.
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario intente iniciar sesión en la aplicación.
Secuencia normal	1. El usuario con rol *Administrador* accede a la página que contiene el formulario para la inserción de un nuevo libro en el sistema. *2.* El usuario procede a rellenar los campos del formulario. Estos campos están definidos en el *ReqInf-002 Libro.* 2.1. Se cumple la restricción *Restr-003 Campos Libro No Vacíos.* 2.2. Se cumple la restricción *Restr-11 ISBN.* 2.3. Se cumple la restricción *Restr-12 Título.* 2.4. Se cumple la restricción *Restr-13 Autor.* 2.5. Se cumple la restricción *Restr-14 Editorial.* 2.6. Se cumple la restricción *Restr-15 Año de publicación.* 3. Tras rellenar el formulario, el usuario lo envía. 4. Éste es procesado por el sistema y el libro insertado en la base de datos. 5. La información sobre el nuevo libro ya se encuentra disponible para poder gestionarla como se requiera. 6. El usuario es redireccionado a otra página en la que se le notifica de que que el libro con ISBN "XXXXXXXXXX" ha sido insertado.
Postcondición	El *Administrador* ha insertado el libro en la aplicación correctamente y el sistema lo notifica por la acción realizada con éxito.
Excepciones	*1.* Si no se cumple la restricción*Restr-003 Campos Libro No Vacíos,* debido a que el campo *ISBN* está vacío:

	E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *ISBN* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *2.* Si no se cumple la restricción *Restr-003 Campos Libro No Vacíos*, debido a que el campo *Título* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Título* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *3.* Si no se cumple la restricción *Restr-003 Campos Libro No Vacíos*, debido a que el campo *Autor* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Autor* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *4.* Si no se cumple la restricción *Restr-003 Campos Libro No Vacíos*, debido a que el campo *Editorial* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Editorial* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *5.* Si no se cumple la restricción *Restr-003 Campos Libro No Vacíos*, debido a que el campo *Año de publicación* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Año de publicación* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *6.* Si no se cumple la restricción *Restr-003 Campos Libro No Vacíos*, debido a que para el campo *Género* no se ha seleccionado ninguna opción: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando al usuario que es necesario seleccionar un género. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *7.* Si no se cumple la restricción *Restr-11 ISBN*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *ISBN* tiene que estar formado por 10 caracteres numéricos. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *8.* Si no se cumple la restricción *Restr-12 Título*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Título* tiene que estar formado por menos de 50 caracteres. E2. No permitirá al usuario insertar el nuevo libro.

	E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *9.* Si no se cumple la restricción *Restr-13 Autor*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Autor* tiene que estar formado por menos de 50 caracteres. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *10.* Si no se cumple la restricción *Restr-14 Editorial*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Editorial* tiene que estar formado por menos de 50 caracteres. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *11.* Si no se cumple la restricción *Restr-15 Año de publicación*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Año de publicación* tiene que estar formado por 4 caracteres numéricos. E2. No permitirá al usuario insertar el nuevo libro. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión.
Comentarios	-

Tabla 12 - CU-012 Insertar libro

CU-013	**Insertar cliente**
Versión	1.0 (15/09/2016)
Utiliza	• *ReqInf-001 Cliente.* • *Restr-002 Campos Cliente No Vacíos.* • *Restr-004 DNI.* • *Restr-005 Cliente.* • *Restr-006 Nombre.* • *Restr-007 Primer Apellido.* • *Restr-008 Segundo Apellido.* • *Restr-009 Dirección.* • *Restr-010 Contraseña.*
Actor	*Administrador*
Precondición	El usuario *Administrador* desea realizar la inserción de un nuevo cliente en el sistema.
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario intente iniciar sesión en la aplicación.
Secuencia normal	1. El usuario con rol *Administrador* accede a la página que contiene el formulario para la inserción de un nuevo cliente en el sistema.

	2. El usuario procede a rellenar los campos del formulario. Estos campos están definidos en el *ReqInf-001 Cliente*. 2.1. Se cumple la restricción *Restr-002 Campos Cliente No Vacíos*. 2.2. Se cumple la restricción *Restr-004 DNI*. 2.3. Se cumple la restricción *Restr-005 Cliente*. 2.4. Se cumple la restricción *Restr-006 Nombre*. 2.5. Se cumple la restricción *Restr-007 Primer Apellido*. 2.6. Se cumple la restricción *Restr-008 Segundo Apellido*. 2.7. Se cumple la restricción *Restr-009 Dirección*. 2.8. Se cumple la restricción *Restr-010 Contraseña*. 3. Tras rellenar el formulario, el usuario lo envía. 4. Éste es procesado por el sistema y el nuevo cliente insertado en la base de datos. 5. El nuevo cliente ya se encuentra disponible para ser usado. 6. El usuario es redireccionado a otra página en la que se le notifica de que que el libro con DNI "XXXXXXXX" ha sido insertado correctamente.
Postcondición	El *Administrador* ha insertado el nuevo cliente en la aplicación correctamente y el sistema se lo notifica por la acción realizada con éxito.
Excepciones	*1.* Si no se cumple la restricción *Restr-002 Campos Cliente No Vacíos*, debido a que el campo *DNI* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *DNI* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *2.* Si no se cumple la restricción *Restr-002 Campos Cliente No Vacíos*, debido a que el campo *Cliente* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Cliente* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *3.* Si no se cumple la restricción *Restr-002 Campos Cliente No Vacíos*, debido a que el campo *Nombre* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Nombre* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *4.* Si no se cumple la restricción *Restr-002 Campos Cliente No Vacíos*, debido a que el campo *Primer Apellido* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Primer Apellido* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *5.* Si no se cumple la restricción *Restr-002 Campos Cliente No Vacíos*, debido a que el campo *Segundo Apellido* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Segundo Apellido* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión.

	6. Si no se cumple la restricción *Restr-002 Campos Cliente No Vacíos*, debido a que el campo *Dirección* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Dirección* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *7.* Si no se cumple la restricción *Restr-002 Campos Cliente No Vacíos*, debido a que el campo *Contraseña* está vacío: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Contraseña* no puede estar vacío. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *8.* Si no se cumple la restricción *Restr-004 DNI*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *DNI* tiene que estar formado por 8 caracteres numéricos. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *9.* Si no se cumple la restricción *Restr-005 Cliente*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Cliente* tiene que contener entre 5 y 12 caracteres. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *10.* Si no se cumple la restricción *Restr-006 Nombre*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Nombre* no puede contener más de 20 caracteres. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *11.* Si no se cumple la restricción *Restr-007 Primer Apellido*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Primer Apellido* no puede contener más de 20 caracteres. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *12.* Si no se cumple la restricción *Restr-008 Segundo Apellido*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Segundo Apellido* no puede contener más de 20 caracteres. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *13.* Si no se cumple la restricción *Restr-009 Dirección*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Dirección* no puede contener más de 20 caracteres. E2. No permitirá al usuario insertar el nuevo cliente.

	E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión. *14.* Si no se cumple la restricción *Restr-010 Contraseña*: E1. La aplicación lanzará un mensaje contenido en una ventana emergente indicando que el campo *Contraseña* tiene que contener entre 5 y 12 caracteres. E2. No permitirá al usuario insertar el nuevo cliente. E3. Da la opción al usuario de volver a introducir unos datos correctos para el campo en cuestión.
Comentarios	-

Tabla 13 - CU-013 Insertar cliente

3.2.3.2 Requisitos de Información

ReqInf-001	**Cliente**
Versión	*1.0 (15/09/2016)*
Utilizado por	•*CU-001 Gestionar sesión.* • *CU-002 Iniciar sesión.* • *CU-004 Ver sus datos personales.* • *CU-005 Ver listado clientes.* • *CU-007 Ver reservas de todos los clientes.* • *CU-008 Ver sus reservas (cliente).* • *CU-009 Borrar reservas de cualquier cliente.* • *CU-010 Borrar reservas realizadas por él (cliente).* • *CU-011 Reservar libro.* • *CU-013 Insertar cliente.* • *Restr-001 Campos Autenticación No Válidos.* • *Restr-002 Campos Cliente No Válidos.* • *Restr-004 DNI.* • *Restr-005 Cliente.* • *Restr-006 Nombre.* • *Restr-007 Primer Apellido.* • *Restr-008 Segundo Apellido.* • *Restr-009 Dirección.* • *Restr-010 Contraseña.*
Descripción	El sistema deberá almacenar la información correspondiente a un usuario. En concreto, la información que debe ser almacenada por el sistema se define a continuación:
Datos específicos	• *DNI*: Reservado para el DNI del usuario. • *Cliente*: Designado para el almacenamiento del Nick del usuario en la aplicación. • *Nombre*: Usado para almacenar el nombre del usuario. • *Primer apellido*: Usado para almacenar el primer apellido del usuario. • *Segundo apellido*: Usado para almacenar el segundo apellido del usuario. • *Dirección*: Usado para almacenar la dirección del usuario.

	• *Administrador*: Se usa en la asignación de un rol al usuario, según lo definido en el apartado *Actores*. • *Contraseña*: Reservado para almacenar la contraseña necesaria en el acceso del usuario a la aplicación.
Importancia	Alta.
Comentarios	-

Tabla 14 - ReqInf-001 Cliente

ReqInf-002	**Libro**
Versión	*1.0 (15/09/2016)*
Utilizado por	• *CU-001 Gestionar sesión.* • *CU-006 Ver listado libros.* • *CU-007 Ver reservas de todos los clientes.* • *CU-008 Ver sus reservas (cliente).* • *CU-009 Borrar reservas de cualquier cliente.* • *CU-010 Borrar reservas realizadas por él (cliente).* • *CU-011 Reservar libro.* • *CU-012 Insertar libro.* •*Restr-003 Campos Libro No Vacíos.* • *Restr-011 ISBN.* • *Restr-012 Título.* • *Restr-013 Autor.* • *Restr-014 Editorial.* • *Restr-015 Año de publicación.*
Descripción	El sistema deberá almacenar la información correspondiente a un *libro*. En concreto, la información que debe ser almacenada por el sistema se define a continuación:
Datos específicos	• *ISBN*: Reservado para el ISBN del libro. • *Título*: Designado para el almacenamiento del título del libro. • *Autor*: Usado para almacenar el nombre del autor del libro. • *Editorial*: Usado para almacenar el nombre de la editorial encargada de la publicación del libro. • *Año de publicación*: Usado para almacenar el año de publicación del libro por parte de la editorial. • *Género*: Usado para almacenar la temática del libro. • *Observaciones*: Reservado para la indicación de cualquier tipo de apreciación adicional acerca del libro en cuestión.
Importancia	Alta.
Comentarios	-

Tabla 15 - ReqInf-002 Libro

3.2.3.3 Reglas de Negocio

RN-001	Registro Previo
Versión	1.0 (15/09/2016)
Descripción	Para poder acceder a la aplicación, así como para poder hacer uso de cualquier funcionalidad de la misma, los usuarios deben estar registrados previamente al momento del acceso en la misma.
Comentarios	-

Tabla 16 - RN-001 Registro Previo

RN-002	Autenticación Previa
Versión	1.0 (15/09/2016)
Descripción	Para poder para poder hacer uso de las distintas funcionalidades de la aplicación, el usuario en cuestión tiene que haberse autenticado correctamente con anterioridad.
Comentarios	-

Tabla 17 - RN-002 Autenticación Previa

RN-003	Roles
Versión	1.0 (15/09/2016)
Descripción	Como define el apartado *Actores*, dependiendo del rol que tenga un usuario tendrá acceso a unas funcionalidades o a otras. Un usuario con rol de *Cliente* no podrá tener los privilegios de *Administrador* y viceversa.
Comentarios	-

Tabla 18 - RN-003 Roles

RN-004	Ejemplares de libros
Versión	1.0 (15/09/2016)
Descripción	En la aplicación, sólo habrá un ejemplar disponible por cada libro, siendo su estado *Reservado* cuando lo tiene reservado algún usuario y *Disponible* en caso contrario.
Comentarios	Por simplicidad, debido a que el objetivo de la aplicación está centrado en la docencia, se ha decidido definir esta Regla de Negocio, es decir, sólo establecer un ejemplar por cada libro en la aplicación.

Tabla 19 - RN-004 Ejemplares de libros

RN-005	Acceso a Datos Personales

Versión	1.0 (15/09/2016)
Descripción	Los usuarios, sea cual sea su rol, podrán visualizar sus datos, pero no modificarlos
Comentarios	-

Tabla 20 - RN-005 Acceso a Datos Personales

RN-006	**Acceso a Clientes**
Versión	*1.0 (15/09/2016)*
Descripción	Un usuario con rol *Administrador* podrá ver un listado de todos los clientes, pero no modificarlos o eliminarlos desde la aplicación. Para poder realizar esto, tendrá que hacerlo a través del programa gestor de la base de datos.
Comentarios	-

Tabla 21 - RN-006 Acceso a Clientes

RN-007	**Acceso a Libros**
Versión	*1.0 (15/09/2016)*
Descripción	Cualquier usuario podrá ver un listado de todos los libros, pero no modificarlos o eliminarlos desde la aplicación. Para poder realizar esto, tendrá que hacerlo a través del programa gestor de la base de datos.
Comentarios	-

Tabla 22 - RN-007 Acceso a Libros

3.2.4 Requisitos No funcionales

3.2.4.1 Requisitos de Seguridad

ReqSeg-001	**Comprobaciones Parte Cliente**
Versión	1.0 (15/09/2016)
Descripción	Para controlar lo enviado por los Usuarios cuando rellenen un formulario, éstos serán comprobados y validados en la parte cliente antes de ser enviados.
Comentarios	Este requisito también es demandado por el carácter docente de la aplicación debido a que en alguna de las asignaturas en las que ésta será usada se imparten conocimientos relacionados con dicho requisito.

Tabla 23 - ReqSeg-001 Comprobaciones Parte Cliente

ReqSeg-002	**Comprobaciones Parte Servidor**

Versión	1.0 (15/09/2016)
Descripción	Con el fin de dotar de un nivel de seguridad mayor a la aplicación protegiéndola de modificaciones externas no deseadas, en los formularios existentes en la aplicación se realizan comprobaciones en la parte del servidor.
Comentarios	Este requisito también es demandado por el carácter docente de la aplicación debido a que en alguna de las asignaturas en las que ésta será usada se imparten conocimientos relacionados con dicho requisito.

Tabla 24 - ReqSeg-002 Comprobaciones Parte Servidor

ReqSeg-003	**Autenticación**
Versión	1.0 (15/09/2016)
Descripción	Debido al carácter puramente académico y a que la enseñanza de las asignaturas en las que, en principio, será usada esta aplicación no se centran en los métodos de autenticación, en este proyecto el método de autenticación usado será el básico, es decir, un usuario y contraseña introducidos previamente en una base de datos.
Comentarios	-

Tabla 25 - ReqSeg-003 Autenticación

3.2.4.2 Requisitos de Portabilidad

ReqPort-001	**WAR**
Versión	1.0 (15/09/2016)
Descripción	Con el fin de permitir la ejecución de la aplicación en distintos equipos y no necesitar la ejecución del IDE y del servidor instalado en él para correr la aplicación, se genera un archivo con extensión *.war*.
Comentarios	Para lograr lo definido anteriormente, será necesario tener en ejecución un servidor.

Tabla 26 - ReqPort-001 WAR

3.2.4.3 Requisitos de Mantenibilidad

ReqMant-001	**Documentación del código**
Versión	1.0 (15/09/2016)
Descripción	Para facilitar la comprensión, análisis, resolución de problemas y futuras modificaciones, será necesaria la documentación del código de programación de la aplicación. Con el fin de facilitar y perfeccionar esta tarea, será usado JavaDoc.
Comentarios	-

Tabla 27 - ReqMant-001 Documentación de código

3.2.4.4 Requisitos motivados por el carácter docente de la aplicación

Aunque la aplicación está basada en una Biblioteca y todo lo relacionado con la gestión de libros y clientes en ésta, el objetivo fundamental está relacionado con la utilidad que la aplicación tendrá para la docencia. Por ello, podemos definir los siguientes requisitos:

ReqDoc-001	**Lenguajes puros**
Versión	1.0 (15/09/2016)
Descripción	Con la finalidad de que los alumnos aprendan a programar y la base de la programación, se decide por usar lenguajes puros, sin ningún tipo de modificación ni ayuda como, por ejemplo, frameworks (entornos de trabajo).
Comentarios	-

Tabla 28 - ReqDoc-001 Lenguajes puros

ReqDoc-002	**Conceptos de diseño y programación**
Versión	1.0 (15/09/2016)
Descripción	Tanto en la asignatura Ingeniería del Software como en Fundamentos de Aplicaciones y Servicios Telemáticos se enseña la utilidad de los patrones de diseño y programación como son Cliente-Servidor y Modelo-Vista-Controlador. Debido a esto, la aplicación debe hacer uso de dichos patrones.
Comentarios	-

Tabla 29 - ReqDoc-002 Conceptos de diseño y programación

ReqDoc-003	**Antipatrón MVC**
Versión	1.0 (15/09/2016)
Descripción	En el *ReqDoc-002* se define el buen uso en el diseño y programación de una aplicación. En el *ReqDoc-003* se busca poner un ejemplo de antipatrón, es decir, de no cumplimiento con lo anteriormente mencionado. Esto se realiza con la finalidad de que los alumnos comparen las distintas formas de programación y comprueben la utilidad de una aplicación bien diseñada y estructurada y el uso de patrones.
Comentarios	El antipatrón será programado en el campo *Género* del formulario de inserción de un libro. En éste, se viola el patrón MVC, encontrándose lógica y presentación dentro de la vista.

Tabla 30 - ReqDoc-003 Antipatrón MVC

ReqDoc-004	**Lista desplegable que haga uso de la Base de Datos**
Versión	1.0 (15/09/2016)
Descripción	Para dotar a la aplicación de una funcionalidad diferente que sirva de aprendizaje para los alumnos, se decide crear una lista desplegable que haga uso de datos almacenados en la Base de Datos. Esta lista desplegable será el selector para el género de un nuevo libro en el formulario de inserción del mismo.
Comentarios	Casualmente, se ha decidido que esta funcionalidad coincida con el antipatrón de MVC definido en el *ReqDoc-003*.

Tabla 31 - ReqDoc-004 Lista desplegable que haga uso de la Base de Datos

ReqDoc-005	**Gestión de Logs**
Versión	1.0 (15/09/2016)
Descripción	Con el fin de notificar al gestor de la aplicación sobre los distintos errores de uso de la aplicación, se decide hacer uso de un sistema de trazado o sistema de *Logs*, es decir, de trazas textuales, no visibles para el usuario, que indiquen acerca de dichos errores. Se ha decidido usar estos *Logs*en la recepción por parte del Servidor de los distintos formularios, es decir, en el Controlador.
Comentarios	Además de *Logs*para gestionar la notificación de errores al gestor de la aplicación, se ha hecho uso de *un manejador de excepciones*. Por otro lado, para notificar al usuario acerca del uso incorrecto de la aplicación o de alguna acción completada con éxito, le serán *mostrados por pantalla* una serie de mensajes indicativos.

Tabla 32 - ReqDoc-005 Gestión de Logs

3.2.5 Restricciones del Sistema

Restr-001	**Campos Autenticación No Vacíos**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: •*ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-002 Iniciar sesión.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: Los campos *Cliente* y *Contraseña*, definidos en el *ReqInf-001 Atributos Cliente,* no pueden estar vacíos. Esta restricción es de especial utilidad en el *CU-002 Iniciar sesión*.
Comentarios	-

Tabla 33 - Restr-001 Campos Autenticación No Vacíos

Restr-002	**Campos Cliente No Vacíos**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: •*CU-013 Insertar cliente.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: Los campos definidos en el *ReqInf-001 Atributos Cliente*, excepto el campo *Administrador*, no pueden estar vacíos. Esta restricción es de especial utilidad en el *CU-013 Insertar cliente*.
Comentarios	-

Tabla 34 - Restr-002 Campos Cliente No Vacíos

Restr-003	**Campos Libro No Vacíos**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-002 Atributos Libro.*
Usado por	Es requerido por los siguientes requisitos: • *CU-012 Insertar libro.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: Los campos definidos en el *ReqInf-002 Atributos Libro*, excepto el campo *Observaciones*, no pueden estar vacíos. Esta restricción es de especial utilidad en el *CU-012 Insertar libro*.
Comentarios	-

Tabla 35 - Restr-001 Campos Libro No Vacíos

Restr-004	***DNI***
Versión	*1.0 (15/09/2016)*
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-013 Insertar cliente.*

Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *DNI*, definido en *ReqInf-001 Atributos Cliente*,deberá: o Ser único. o Estar formado por 8 caracteres numéricos.
Comentarios	-

Tabla 36 - Restr-004 DNI

Restr-005	**Cliente**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-001 Gestionar sesión.* • *CU-002 Iniciar sesión.* • *CU-013 Insertar cliente.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Cliente*, definido en *ReqInf-001 Atributos Cliente*,deberá: o Estar comprendido entre 5 y 12 caracteres.
Comentarios	-

Tabla 37 - Restr-005 Cliente

Restr-006	**Nombre**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-013 Insertar cliente.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Nombre*, definido en *ReqInf-001 Atributos Cliente*,deberá: o Estar formado por menos de 20 caracteres.
Comentarios	-

Tabla 38 - Restr-006 Nombre

Restr-007	**Primer apellido**
Versión	1.0(15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-013 Insertar cliente.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Primer Apellido*, definido en *ReqInf-001 Atributos Cliente*,deberá: o Estar formado por menos de 20 caracteres.
Comentarios	-

Tabla 39 - Restr-007 Primer apellido

Restr-008	**Segundo apellido**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-013 Insertar cliente.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Segundo Apellido*, definido en *ReqInf-001 Atributos Cliente*,deberá: o Estar formado por menos de 20 caracteres.
Comentarios	-

Tabla 40 - Restr-008 Segundo apellido

Restr-009	**Dirección**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-013 Insertar cliente.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Dirección*, definido en *ReqInf-001 Atributos Cliente*,deberá:

	○ Estar formado por menos de 40 caracteres.
Comentarios	-

Tabla 41 - Restr-009 Dirección

Restr-010	**Contraseña**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-001 Atributos Cliente.*
Usado por	Es requerido por los siguientes requisitos: • *CU-001 Gestionar sesión.* • *CU-002 Iniciar sesión.* • *CU-013 Insertar cliente.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Contraseña*, definido en *ReqInf-001 Atributos Cliente*,deberá: ○ Estar comprendido entre 5 y 12 caracteres.
Comentarios	-

Tabla 42 - Restr-010 Contraseña

Restr-011	**ISBN**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-002 Atributos Libro.*
Usado por	Es requerido por los siguientes requisitos: • *CU-013 Insertar libro.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *DNI*, definido en *ReqInf-002 Atributos Libro*, deberá: ○ Ser único. ○ Estar formado por 10 caracteres numéricos.
Comentarios	-

Tabla 43 - Restr-011 ISBN

Restr-012	**Título**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-002 Atributos Libro*.
Usado por	Es requerido por los siguientes requisitos: • *CU-012 Insertar libro*.
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Título*, definido en *ReqInf-002 Atributos Libro*,deberá: o Estar formado por menos de 50 caracteres.
Comentarios	-

Tabla 44 - Restr-012 Titulo

Restr-013	**Autor**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-002 Atributos Libro*.
Usado por	Es requerido por los siguientes requisitos: • *CU-012 Insertar libro*.
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Autor*, definido en *ReqInf-002 Atributos Libro*,deberá: o Estar formado por menos de 50 caracteres.
Comentarios	-

Tabla 45 - Restr-013 Autor

Restr-014	**Editorial**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-002 Atributos Libro*.
Usado por	Es requerido por los siguientes requisitos: • *CU-012 Insertar libro*.
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Editorial*, definido en *ReqInf-002 Atributos Libro*,deberá: o Estar formado por menos de 50 caracteres.
Comentarios	-

Tabla 46 - Restr-014 Editorial

Restr-015	**Año de publicación**
Versión	1.0 (15/09/2016)
Aplicado sobre	Para su funcionamiento, esta restricción es aplicada sobre los siguientes requisitos: • *ReqInf-002 Atributos Libro.*
Usado por	Es requerido por los siguientes requisitos: • *CU-013 Insertar libro.*
Descripción	El sistema deberá respetar la siguiente restricción técnica: El campo *Año de publicación,* definido en el *ReqInf-002 Atributos Libro,* deberá: o Estar formado por 4 caracteres numéricos.
Comentarios	-

Tabla 47 - Restr-015 Año de publicación

3.3 Diseño – Desarrollo

Se ha decidido explicar el diseño de la aplicación relacionándolo con los distintos Casos de Uso. El orden seguido en la explicación será el siguiente:

- Modelo de Datos.
- Clases, con sus respectivas funciones.
- Archivos de comprobación de formularios del lado del cliente con sus respectivas funciones.
- Arquitectura y diseño de los tres componentes del patrón MVC por cada Caso de Uso.

3.3.1 Modelo de Datos

Como ya se comentó en apartados anteriores, el Modelo de Datos está implementado sobre una Base de Datos relacional usando PostreSQL.

Esta Base de Datos relacional está formada por cuatro tablas (*clientes*, *libros*, *generos* y *reservas*), las cuales están relacionadas entre sí.

La realización de este Modelo de Datos se ha basado en los *Requisitos de Información* definidos en apartados anteriores.

A continuación, se mostrará su diagrama Entidad – Relación y se explicarán de forma individual cada una de las tablas que componen el Modelo de Datos y la relación entre ellas.

3.3.1.1 Modelo Entidad – Relación

El Modelo de Datos comentado anteriormente, se describe en el modelo entidad – relación a continuación. Con la ayuda de este diagrama estructural, se pueden ver de forma clara y esquemática la estructura de la Base de Datos, así como las relaciones entre las distintas entidades definidas anteriormente.

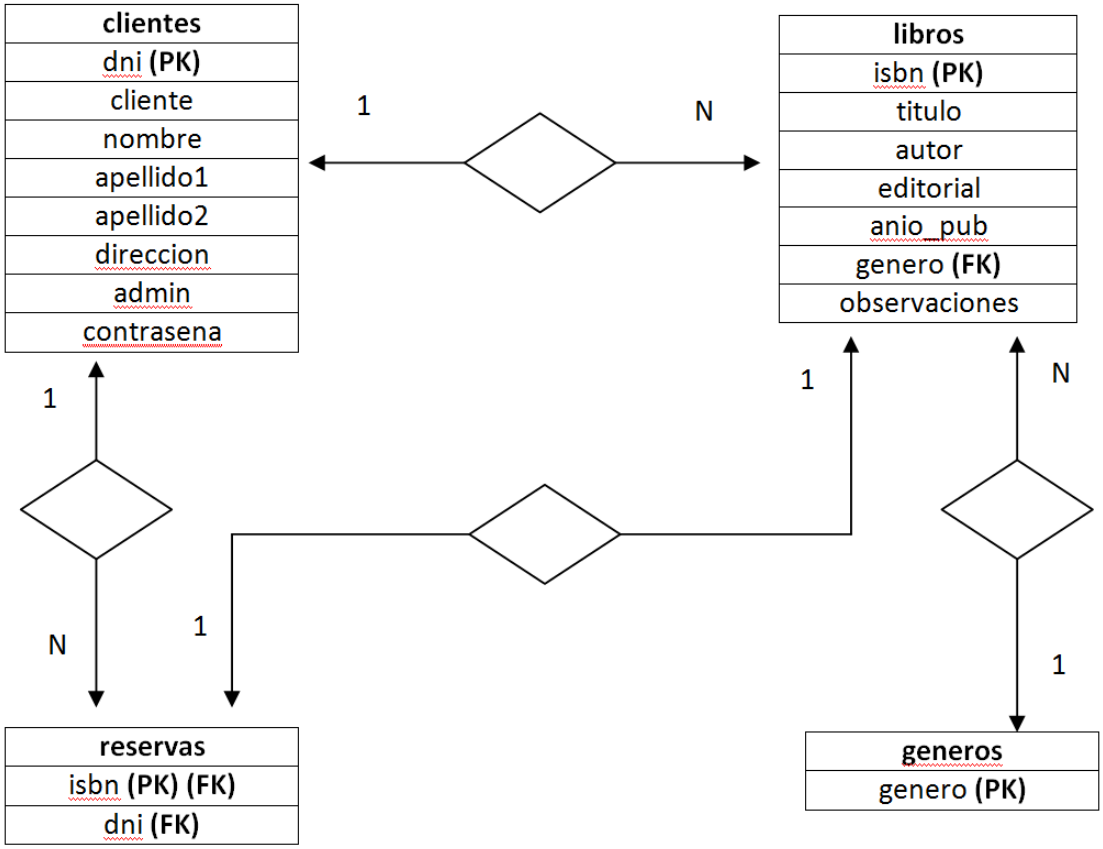

Figura 8 - Modelo Entidad - Relación de la Base de Datos de la aplicación

3.3.1.2 clientes

Esta tabla es usada en la persistencia de los datos relacionados con los usuarios de la aplicación.

Sobre ella, se realizan consultas como la visualización de los datos personales de un usuario o de todos los clientes registrados en la aplicación.

Las clases que acceden a esta tabla son: *Cliente* y *Conexion*.

Los campos de los que se compone dicha tabla son los siguientes:

- **dni:**Este campo está reservado para almacenar el DNI del usuario de la aplicación. Como se ha decidido utilizar el DNI sin letra, este campo es de tipo *integer*. Este campo es Clave Primaria (PK) de ésta tabla, así como único ya que será el identificador de cada cliente en la aplicación.
- **cliente:** Este campo está reservado para almacenar el Nick del usuario de la aplicación. Es de tipo *character* y , al igual que el campo DNI, y para que no haya dos usuarios con el mismo nombre de usuario, también será único.
- **nombre:** Destinado al almacenamiento del nombre del usuario de la aplicación. Es de tipo *character*.
- **apellido1:** Destinado al almacenamiento del primer apellido del usuario de la aplicación. Es de tipo *character*.

- **apellido2:** Destinado al almacenamiento del segundo apellido del usuario de la aplicación. Es de tipo *character*.
- **direccion:** Destinado al almacenamiento de la dirección del domicilio del usuario de la aplicación. Es de tipo *character*.
- **admin:** Debido a su carácter booleano, este campo es de tipo bit, tomando el valor 1 (verdadero) cuando el usuario tiene el rol de administrador y 0 (falso) en caso contrario.
- **contrasena:** Este campo está reservado para almacenar la contraseña del usuario de la aplicación. Es de tipo *character* ya que permite almacenar caracteres alfanuméricos. Este campo, junto a *cliente*, forman las credenciales de autenticación de los usuarios en la aplicación.

El modelo físico de datos lo podemos encontrar resumido en la siguiente tabla:

CLIENTES					
Campo	**Tipo**	**Longitud**	**PK**	**FK**	**Descripción**
dni	integer	-	X		DNI del usuario
cliente	character	100			Nick del usuario en la aplicación
nombre	character	100			Nombre del usuario
apellido1	character	100			Primer apellido del usuario
apellido2	character	100			Segundo apellido del usuario
dirección	character	100			Dirección física del usuario
admin	bit	1			Booleano que indica si el usuario es cliente o administrador
contrasena	character	20			Contraseña para el ingreso del usuario en la aplicación

Tabla 48–Resumen de la tabla clientes de la Base de Datos

3.3.1.3 libros

Esta tabla es usada en la persistencia de los datos relacionados con los libros de la aplicación.

Sobre ella, se realizan consultas como la visualización de los libros que se encuentran registrados en la aplicación o la inserción de un nuevo libro en ésta.

Las clases que acceden a esta tabla son: *Libro* y *Conexion*.

Los campos de los que se compone dicha tabla son los siguientes:

- **isbn:**Este campo está reservado para almacenar el ISBN de un libro en la aplicación. Es de tipo *bigint* y es Clave Primaria (PK) de esta tabla, así como único ya que será el identificador de cada libro en la aplicación.
- **titulo:** Este campo está reservado para almacenar el título de los libros de la aplicación. Es de tipo *character*.

- **autor:** Destinado al almacenamiento del nombre del autor del libro en cuestiónen la aplicación. Es de tipo *character*.
- **editorial:** Destinado al almacenamiento de la editorial del libro en cuestión en la aplicación. Es de tipo *char*acter.
- **anio_pub:** Destinado al almacenamiento del año de publicación del libro en cuestión en la aplicación. Es de tipo *integer*.
- **genero:** Destinado al almacenamiento del género del libro en cuestión en la aplicación. Es de tipo *character* y es Clave Externa (FK) en esta tabla, sirviendo de enlace con la tabla *generos*, en la cual es Clave Primaria (PK).
- **observaciones:** Destinado al almacenamiento de las posibles observaciones relativas al libro en cuestión en la aplicación. Es de tipo *char*acter.

El modelo físico de datos lo podemos encontrar resumido en la siguiente tabla:

LIBROS					
Campo	**Tipo**	**Longitud**	**PK**	**FK**	**Descripción**
isbn	bigint	-	X		ISBN del libro
titulo	character	100			Título del libro
autor	character	100			Nombre del autor del libro
editorial	character	100			Nombre de la editorial del libro
anio_pub	integer	-			Año de publicación de esa edición del libro
genero	character	100		X	Temática en la que está encuadrada el libro
observaciones	character	200			Observaciones adicionales sobre el libro

Tabla 49 - Resumen de la tabla libros de la Base de Datos

3.3.1.4 generos

Esta tabla es usada en la persistencia de los datos relacionados con los diferentes géneros que puede tener los libros dentro de la aplicación como pueden ser *Narrativa*, *Informática* o *Ingeniería* entre otros.

La creación de esta tabla viene motivada por el requisito no funcional docente *ReqDoc-004 Lista desplegable que haga uso de la Base de Datos*.

Las clases que acceden a esta tabla son: *Libro* y *Conexion*.

Esta tabla está compuesta por el campo genero, el cual es Clave Primaria (PK) en esta tabla y Clave Externa en la tabla *libros*, en la cual influye asignando el género a los libros que en ella están

almacenados.

El modelo físico de datos lo podemos encontrar resumido en la siguiente tabla:

GENEROS					
Campo	**Tipo**	**Longitud**	**PK**	**FK**	**Descripción**
genero	character	100	X		Temática en la que está encuadrada el libro. Es FK de la tabla *libros*.

Tabla 50 - Resumen de la tabla generos de la Base de Datos

3.3.1.5 reservas

Esta tabla es usada en la persistencia de los datos relacionados con las reservas de los libros por parte de los clientes.

Sobre ella, se realizan consultas como la visualización de las reservas de todos los clientes o de las que dispone un cliente en concreto, así como el borrado de éstas.

Las clases que acceden a esta tabla son: *Libro* y *Conexion*.

Los campos de los que se compone dicha tabla son los siguientes:

- **isbn:**Este campo está reservado para almacenar el ISBN de un libro que se encuentra reservado por un cliente. Es de tipo *bigint* y es Clave Primaria (PK) de esta tabla, así como único ya que será el identificador de cada libro en la aplicación.
- **dni:** Este campo está reservado para almacenar el DNI del cliente que tiene reservado el libro. Es de tipo *integer*.

El modelo físico de datos lo podemos encontrar resumido en la siguiente tabla:

RESERVAS					
Campo	**Tipo**	**Longitud**	**PK**	**FK**	**Descripción**
isbn	bigint		X	X	ISBN del libro reservado
dni	integer			X	DNI del cliente que tiene el libro reservado

Tabla 51 - Resumen de la tabla reservas de la Base de Datos

3.3.2 Clases

Para el desarrollo de la aplicación, se han implementado las siguientes clases:

- Constantes
- Conexion
- Cliente
- Libro

- Reserva

Estas clases, junto con sus constructores y métodos, serán explicadas a continuación.

3.3.2.1 Constantes

El uso de esta clase es el de definir las constantes, las cuales no será habitual que se modifiquen, que se vea conveniente utilizar en la aplicación.

Esta clase no instancia ningún objeto. Todas las variables que en ella se definen son estáticas.

Actualmente, en dicha clase están definidas las constantes relacionadas con JDBC y la conexión de la aplicación con la Base de Datos. JDBC es el conector de la Base de Datos con el código escrito en Java. Su función es la de habilitar la ejecución de operaciones sobre la Base de Datos desde el lenguaje de programación Java.

3.3.2.2 Conexion

Como su nombre indica, esta clase es usada en la conexión con la Base de Datos. Para ello, en su constructor, se define la carga del driver de JDBC y el posterior acceso a la Base de Datos con las credenciales de acceso definidas en la clase *Constantes*explicada con anterioridad.

Esta clase incluye los siguientes métodos:

- *public Connection getConexion():*Devuelve información sobre la conexión creada anteriormente.
- *public void cerrarConexion():*Se encarga de cerrar la conexión creada anteriormente.

Además, debido a que ésta es la clase utilizada como elemento de conexión con la Base de Datos, en ella están implementados los métodos de acceso directo a la Base de Datos. Otras clases les pasarán los datos a los métodos implementados en esta clase para que éstos los gestionen de forma directa con la Base de Datos.

Estos métodos son los siguientes:

- *public Integer insertar(String sql):*Se encarga de insertar en la Base de Datos los datos recibidos como parámetro en la variable *sql*.
- *public Integer consultar(String sql):*Se encarga de realizar una consulta a la Base de Datos. La consulta a realizar está especificada en los datos recibidos como parámetro en la variable *sql*.
- *public Integer borrar(String sql):* Se encarga de eliminar de la Base de Datos lo indicado en la variable *sql* recibida como parámetro.
- *public ArrayList<Cliente> listar_Clientes(String sql):* Se encarga de listar los clientes relacionados con la consulta contenida en la variable *sql* recibida como parámetro.

  Para ello, crea un objeto para enviar instrucciones a la Base de Datos y un ArrayList de tipo Cliente. Rellena el ArrayList con los datos correspondientes a la consulta anteriormente mencionada. Realizará esto mientras el valor de rs.next() sea *true*.

  Tras esto, devuelve el listado creado anteriormente.

- *public ArrayList<Libro> listar_Libros(String sql):*Se encarga de listar los libros relacionados con la consulta contenida en la variable *sql* recibida como parámetro.

  Para ello, crea un objeto para enviar instrucciones a la Base de Datos y un ArrayList de tipo Libro. Rellena el ArrayList con los datos correspondientes a la consulta anteriormente mencionada. Realizará esto mientras el valor de rs.next() sea *true*.

  Tras esto, devuelve el listado creado anteriormente.

- *public ArrayList<Reserva> listar_Reservas(String sql):* Se encarga de listar las reservas relacionadas con la consulta contenida en la variable *sql* recibida como parámetro.

  Para ello, crea un objeto para enviar instrucciones a la Base de Datos y un ArrayList de tipo Reserva. Rellena el ArrayList con los datos correspondientes a la consulta anteriormente mencionada. Realizará esto mientras el valor de rs.next() sea *true*.

  Tras esto, devuelve el listado creado anteriormente.

3.3.2.3 Libro

Esta clase es usada en todo lo relacionado con la gestión de los libros desde la aplicación.

Está formada por variables, constructores, setters y getters y métodos. Todos estos componentes serán descritos a continuación.

a) Variables

En esta clase están definidas las siguientes variables:

- isbn: Es de tipo *long* y almacena el isbn del libro.
- titulo: Es de tipo *String* y almacena el título del libro.
- autor: Es de tipo *String* y almacena el autor del libro.
- editorial: Es de tipo *String* y almacena la editorial que publica el libro.
- anio_pub: Es de tipo *Integer* y almacena el año de publicación del libro.
- genero: Es de tipo *String* y almacena el género del libro.
- observaciones: Es de tipo *String* y almacena las observaciones adicionales acerca del libro en cuestión.
- dni: Es de tipo *Integer* y se usa para indicar los libros que están reservados.
- obj_conn: Es de tipo *Conexion* y se usa para pasarle los datos a la clase *Conexion.java* para que ésta los gestione con la Base de Datos.

b) Constructores

Se han definido varios constructores diferentes para abarcar futuros usos aunque en la actualidad no se están usando todos ellos.

Los constructores definidos son los siguientes:

- Libro(): Constructor vacío.
- Libro(Long isbn): Constructor que instancia un objeto de la clase Libro con la clave primaria.

- Libro(Long isbn, String titulo, String autor, String editorial, Integer anio_pub, String genero,String observaciones): Constructor para instanciar un objeto de la clase Libro completando todos los campos referentes a un libro.
- Libro(Long isbn, String titulo, String autor, String editorial, Integer anio_pub, String genero,String observaciones, Integer dni): Constructor para instanciar un objeto de la clase Libro completando todos los campos referentes a un libro y el DNI del cliente que tiene reservado el libro.

c) Setters y Getters

Por cada variable existente en esta clase hay definido un método *set* utilizado para establecer datos en dicha variable de un objeto instanciado previamente y un método *get* para obtener el valor de dicha variable.

d) Métodos

En esta clase nos encontramos definidos los siguientes métodos:

- *public Integer insertar_Libro():*Colabora en la inserción de un nuevo libro en la Base de Datos. En concreto, define la consulta que se le va a realizar a la Base de Datos con los datos que recibe del formulario de inserción del nuevo libro y se la envía al método *insertar(String sql)* de la clase *Conexion*para que éste lo inserte en la Base de Datos.

  Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.
- *public ArrayList<Libro> listar_Libros():* Colabora en la inserción de un nuevo libro en la Base de Datos. En concreto, define la consulta que se le va a realizar a la Base de Datos con sobre el listado de todos los libros de la aplicación y se la envía al método *listar_Libros(String sql)* de la clase *Conexion*para que éste la gestione.

  Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.

3.3.2.4 Cliente

Esta clase es usada en todo lo relacionado con la gestión de los clientes desde la aplicación.

Está compuesta por variables, constructores, setters y getters y métodos. Todos estos componentes serán descritos a continuación.

a) Variables

En esta clase están definidas las siguientes variables:

- dni: Es de tipo *Integer* y almacena el DNI del cliente.
- cliente: Es de tipo *String* y almacena el Nick del cliente.
- nombre: Es de tipo *String* y almacena el nombre del cliente.
- apellido1: Es de tipo *String* y almacena el primer apellido del cliente.
- apellido2: Es de tipo *String* y almacena el segundo apellido del cliente.
- direccion: Es de tipo *String* y almacena la dirección del cliente.

- admin: Es de tipo *Boolean* y se usa para distinguir entre los usuarios con rol de administrador cuando esta variable vale *True* y entre los usuarios con rol de cliente cuando esta variable vale *False*.
- contrasena: Es de tipo *String* y almacena la contraseña del cliente.
- obj_conn: Es de tipo *Conexion* y se usa para pasarle los datos a la clase *Conexion* para que ésta los gestione con la Base de Datos.
- correcto: Es de tipo *boolean* y se usa para indicar si el usuario existe en la aplicación, adquiriendo esta variable el valor *True* y permitiendo el acceso a la aplicación a dicho usuario.

b) Constructores

Se han definido varios constructores diferentes para abarcar futuros usos aunque en la actualidad no se están usando todos ellos.

Los constructores definidos son los siguientes:

- Cliente(): Constructor vacío.
- Cliente(Integer dni): Constructor que instancia un objeto de la clase Cliente con la clave primaria.
- Cliente(Integer dni, String nombre, String apellido1, String apellido2, String direccion,Boolean admin): Constructor para instanciar un objeto de la clase Cliente completando todos los campos referentes a un cliente, excepto *cliente* y *contrasena*.
- Cliente(Integer dni, String cliente, String nombre, String apellido1, String apellido2, String direccion,Boolean admin, String contraseña): Constructor para instanciar un objeto de la clase Cliente completando todos los campos referentes a un cliente.

c) Setters y Getters

Por cada variable existente en esta clase hay definido un método *set* utilizado para establecer datos en dicha variable de un objeto instanciado previamente y un método *get* para obtener el valor de dicha variable.

d) Métodos

En esta clase nos encontramos definidos los siguientes métodos:

- *public Integer login(String c,String p):* Se encarga de permitir o denegar la entrada de un cliente en la aplicación. Para ello, se encarga de comprobar si las credenciales de autenticación insertadas en el formulario inicial existen en la Base de Datos. En concreto, define la consulta que se le va a realizar a la Base de Datos con los datos que recibe del formulario de autenticación y se la envía al método *consultar(String sql)* de la clase *Conexion*para que éste realice la consulta a la Base de Datos.

  Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.
- *public boolean getCorrecto():* Se encarga de obtener los datos de la Base de Datos para rellenar las distintas propiedades del Bean de la clase cliente en caso de que éstas no hayan sido rellenadas anteriormente, es decir, en caso de que la variable *correcto* sea *false*. Esto lo realiza de la siguiente forma:

- Comprueba que el valor de la variable *correcto* sea *false*.
- Si lo anteriormente mencionado ocurre (en caso contrario no hace nada), comprueba que el cliente exista en la Base de Datos.
- Tras esto, rellena las distintas propiedades del cliente en cuestión.

Por último, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.

Como se explicará posteriormente, este método es ejecutado al inicio de la mayoría de las funcionalidades de la aplicación, inicializando los datos del Bean de cliente en caso de que éstos no hayan sido inicializados.

- *public Integer insertar_Cliente():* Colabora en la inserción de un nuevo cliente en la Base de Datos. En concreto, define la consulta que se le va a realizar a la Base de Datos con los datos que recibe del formulario de inserción del nuevo cliente y se la envía al método *insertar(String sql)* de la clase *Conexion*para que éste lo inserte en la Base de Datos.

  Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.
- public ArrayList<Cliente>listar_Clientes(): Colabora en la inserción de un nuevo cliente en la Base de Datos. En concreto, define la consulta que se le va a realizar a la Base de Datos con sobre el listado de todos los clientes de la aplicación y se la envía al método *listarClientes(String sql)* de la clase *Conexion*para que éste la gestione.

  Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.

3.3.2.5 Reserva

Esta clase es usada en todo lo relacionado con la gestión de los libros desde la aplicación.

Está formada por variables, constructores, setters y getters y métodos. Todos estos componentes serán descritos a continuación.

a) Variables

En esta clase están definidas las siguientes variables:

- isbn: Es de tipo *long* y almacena el isbn del libro reservado. Usada para listar las reservas.
- titulo: Es de tipo *String* y almacena el título del libro reservado. Usada para listar las reservas.
- autor: Es de tipo *String* y almacena el autor del libro reservado. Usada para listar las reservas.
- editorial: Es de tipo *String* y almacena la editorial que publica el libro reservado. Usada para listar las reservas.
- anio_pub: Es de tipo *Integer* y almacena el año de publicación del libro reservado. Usada para listar las reservas.
- genero: Es de tipo *String* y almacena el género del libro reservado. Usada para listar las reservas.
- observaciones: Es de tipo *String* y almacena las observaciones adicionales acerca del libro en cuestión reservado. Usada para listar las reservas.
- dni: Es de tipo *Integer* y almacena el DNI del usuario que tiene reservado el libro.

- obj_conn: Es de tipo *Conexion* y se usa para pasarle los datos a la clase *Conexion.java* para que ésta los gestione con la Base de Datos.

b) Constructores

Se han definido varios constructores diferentes para abarcar futuros usos aunque en la actualidad no se están usando todos ellos.

Los constructores definidos son los siguientes:

- Reserva(): Constructor vacío.
- Reserva(Long isbn): Constructor que instancia un objeto de la clase Reserva con la clave primaria.
- Reserva(Long isbn,Integer dni): Constructor que instancia un objeto de la clase Reserva con todos los campos contenidos en la tabla *reservas*.
- Reserva (Long isbn, String titulo, String autor, String editorial, Integer anio_pub, String genero,String observaciones, Integer dni): Constructor para instanciar un objeto de la clase Reserva completando todos los campos referentes a un libro y el DNI del cliente que tiene reservado el libro.

c) Setters y Getters

Por cada variable existente en esta clase hay definido un método *set* utilizado para establecer datos en dicha variable de un objeto instanciado previamente y un método *get* para obtener el valor de dicha variable.

d) Métodos

En esta clase nos encontramos definidos los siguientes métodos:

- *public Integer reservar_Libro():*Colabora en la reserva de un libro por parte del cliente que tiene abierta la sesión. En concreto, define la consulta que se le va a realizar a la Base de Datos pasándole al método *insertar(String sql)* de la clase *Conexion*(encargado de la inserción en la Base de Datos) el *isbn* y el *dni* correspondientes al libro a reservar y al cliente que realiza la reserva respectivamente. Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.
- *public ArrayList<Reserva>listar_Reservas():*Colabora en el listado de todas las reservas en el momento de la consulta. En concreto, define la consulta que se le va a realizar a la Base de Datos con sobre el listado de todas las reservas de la aplicación y se la envía al método *listar_Reservas(String sql)* de la clase *Conexion*para que éste la gestione.

  Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.
- *public ArrayList<Reserva>listar_Reservas():*Colabora en el listado de todas las reservas realizadas por el cliente que realiza la consulta en el momento de la realización de ésta. En concreto, define la consulta que se le va a realizar a la Base de Datos con sobre el listado de todas las reservas de la aplicación y se la envía al método *listar_Reservas(String sql)* de la clase *Conexion*para que éste la gestione.

  Tras esto, cierra la conexión instanciada y abierta anteriormente en la llamada al constructor.

3.3.3 Archivos Comprobación Parte Cliente

En las asignaturas en las que la aplicación será herramienta de trabajo, se trata la seguridad en las aplicaciones web. Debido a esto, y basándonos en el *ReqSeg-001 Comprobaciones Parte Cliente*, se realizarán comprobaciones al contenido insertado por los usuarios en los formularios antes de enviarlos. Para ello, hemos usado una serie de ficheros Javascript, los cuales enumeramos a continuación, así como las funciones que en éstos se encuentran.

- global.js
- funciones.js
  - compruebaVacio
  - compruebaLongitud
  - compruebaMax
  - compruebaISBN
  - compruebaAnioPub
  - compruebaDNI
- jsinicio.js
- compr_nuevo_cliente.js
- compr_nuevo_libro.js

3.3.3.1 global.js

En este archivo podemos encontrar las variables globales usadas en los demás archivos. La motivación del uso de este archivo es la de adquirir una mayor manejabilidad, usabilidad, rapidez y simplicidad en cuanto a la modificación del valor de las distintas variables.

Este archivo está dividido en dos partes. En la primera, están las variables que afectan a la comprobación del formulario de autenticación de los usuarios, situado en la página inicial de la aplicación, y del formulario de inserción de nuevos clientes. Por otro lado, en la segunda parte, podemos encontrar las variables relacionadas con la comprobación del formulario de inserción de nuevos libros.

3.3.3.2 funciones.js

En este archivo podemos encontrar todas las funciones necesarias para la comprobación en la parte cliente de los formularios de inserción de nuevos clientes, nuevos libros y de autenticación de los usuarios.

A continuación, procederemos a explicar de forma detallada cada una de las funciones que forman parte de este archivo:

a) compruebaVacio(param,texto)

Variables:

- param: Parámetro a comprobar.
- texto: Texto a mostrar en caso de que se cumpla lo especificado en las condiciones.

Definición: Esta función se encarga de comprobar si la variable que le llega por parámetros, en el campo *param*, está vacía o no.

Ejecución: En el caso de que la variable *param* esté vacía, se mostrará un mensaje por pantalla indicando que el campo con el valor introducido en la variable *texto* no puede estar vacío y devuelve el valor *false*. En caso contrario, devuelve *true*.

Ejemplo de uso: Se realiza a la función compruebaVacio la siguiente llamada: *compruebaVacio("","DNI");* La función lanzaría un mensaje por pantalla indicando que "*El campo DNI no puede estar vacío."* y devolvería *false*.

b) compruebaLongitud(param,min_long_max_long,texto)

Variables:

- param: Parámetro a comprobar.
- min_long: Rango inferior con el que comparar la longitud del parámetro *param*.
- max_long: Rango superior con el que comparar la longitud del parámetro *param*.
- texto: Texto a mostrar en caso de que se cumpla lo especificado en las condiciones.

Definición: Esta función se encarga de comprobar si la longitud de la variable que le llega por parámetros, en el campo *param*, es mayor que *max_long* o menor que *min_long*.

Ejecución: En el caso de que la longitud de la variable *param* cumpla lo explicado anteriormente, se mostrará por pantalla un mensaje indicando el campo *texto* tiene que tener entre *min_long* y *max_long* caracteresy devolverá el valor *false*. En caso contrario, devuelve *true*.

Ejemplo de uso: Se realiza a la función compruebaLongitud la siguiente llamada: *compruebaLongitud("jose",5,12,"cliente");* La función lanzaría un mensaje por pantalla indicando que *"El campo cliente tiene que estar entre 5 y 12 caracteres."* y devolvería *false*.

c) compruebaMax(param,max_long,texto)

Variables:

- param: Parámetro a comprobar.
- max_long: Rango superior con el que comparar la longitud del parámetro *param*.
- texto: Texto a mostrar en caso de que se cumpla lo especificado en las condiciones.

Definición: Esta función se encarga de comprobar si la longitud de la variable que le llega por parámetros, en el campo *param*, es mayor que *max_long*.

Ejecución: En el caso de que la longitud de la variable *param* cumpla lo explicado anteriormente, se mostrará por pantalla un mensaje indicando que el campo *texto* no puede tener más de *max_long* caracteresy devolverá el valor *false*. En caso contrario, devuelve *true*.

Ejemplo de uso: Se realiza a la función compruebaMax la siguiente llamada: *compruebaMax("Juan Rafael",8,"nombre");* La función lanzaría un mensaje por pantalla indicando que *"El campo nombre no puede tener más de 8 caracteres."* y devolvería *false*.

d) compruebaISBN(param,tamanio,texto)

Variables:

- param: Parámetro a comprobar.
- tamanio: Tamaño a mostrar en caso de que se cumpla lo especificado en las condiciones.
- texto: Texto a mostrar en caso de que se cumpla lo especificado en las condiciones.

Definición: Esta función se encarga de comprobar si la longitud de la variable que le llega por parámetros, en el campo *param*, no tiene *tamanio* caracteres numéricos. Es decir, que el ISBN del libro a insertar no tenga 10 caracteres numéricos. Para ello, hace uso de una expresión regular con la que comparar los parámetros recibidos.

Ejecución: En el caso de que la longitud de la variable *param* cumpla lo explicado anteriormente, se mostrará por pantalla un mensaje indicando que el campo *texto* tiene que tener *tamanio* caracteres numéricos y devolverá el valor *false*. En caso contrario, devuelve *true*.

Ejemplo de uso: Se realiza a la función compruebaISBN la siguiente llamada: *compruebaISBN("123434239aa",10,"ISBN");* La función lanzaría un mensaje por pantalla indicando que *"El campo ISBN tiene que tener 10 caracteres numéricos."* y devolvería *false*.

Comentarios: Debido a la complejidad que supone pasar un parámetro/variable para que forme parte de una expresión literal, el campo *tamanio* es simplemente informativo (sólo usado en el mensaje de error). Si se quisiera modificar el número de caracteres a cumplir, habría que modificar la expresión literal (sustituyendo *d{9}* por el valor deseado).

A causa de lo anteriormente mencionado, y como no se ha considerado crítica la optimización de código, nos encontraremos con varias funciones (compruebaAnioPub y compruebaPub) cuya finalidad es la misma que la de esta función y cuya diferencia radica en el número de caracteres especificado en la expresión literal.

e) compruebaAnioPub(param,tamanio,texto)

Variables:

- param: Parámetro a comprobar.
- tamanio: Tamaño a mostrar en caso de que se cumpla lo especificado en las condiciones.
- texto: Texto a mostrar en caso de que se cumpla lo especificado en las condiciones.

Definición: Esta función se encarga de comprobar si la longitud de la variable que le llega por parámetros, en el campo *param*, no tiene *tamanio* caracteres numéricos. Es decir, que el año de publicación del libro a insertar no tenga 4 caracteres numéricos. Para ello, hace uso de una expresión regular con la que comparar los parámetros recibidos.

Ejecución: En el caso de que la longitud de la variable *param* cumpla lo explicado anteriormente, se mostrará por pantalla un mensaje indicando que el campo *texto* tiene que tener *tamanio* caracteres numéricos y devolverá el valor *false*. En caso contrario, devuelve *true*.

Ejemplo de uso: Se realiza a la función compruebaAnioPub la siguiente llamada: *compruebaISBN(12345,4,"año de publicación");* La función lanzaría un mensaje por pantalla indicando que *"El campo Año de publicación no puede tener más de 4 caracteres numéricos."* y devolvería *false*.

Comentarios: Debido a la complejidad que supone pasar un parámetro/variable para que forme parte de una expresión literal, el campo *tamanio* es simplemente informativo (sólo usado en el mensaje de error). Si se quisiera modificar el número de caracteres a cumplir, habría que modificar la expresión literal (sustituyendo *d{3}* por el valor deseado).

f) compruebaDNI(param,tamanio,texto)

Variables:

- param: Parámetro a comprobar.
- tamanio: Tamaño a mostrar en caso de que se cumpla lo especificado en las condiciones.
- texto: Texto a mostrar en caso de que se cumpla lo especificado en las condiciones.

Definición: Esta función se encarga de comprobar si la longitud de la variable que le llega por parámetros, en el campo *param*, no tiene *tamanio* caracteres numéricos. Es decir, que el DNI del cliente a insertar no tenga 8 caracteres numéricos. Para ello, hace uso de una expresión regular con la que comparar los parámetros recibidos.

Ejecución: En el caso de que la longitud de la variable *param* cumpla lo explicado anteriormente, se mostrará por pantalla un mensaje indicando que el campo *texto* tiene que tener *tamanio* caracteres numéricos y devolverá el valor *false*. En caso contrario, devuelve *true*.

Ejemplo de uso: Se realiza a la función compruebaDNI la siguiente llamada: *compruebaDNI(123456789,8,"DNI");* La función lanzaría un mensaje por pantalla indicando que *"El campo DNI tiene que tener 8 caracteres numéricos."* y devolvería *false*.

Comentarios: Debido a la complejidad que supone pasar un parámetro/variable para que forme parte de una expresión literal, el campo *tamanio* es simplemente informativo (sólo usado en el mensaje de error). Si se quisiera modificar el número de caracteres a cumplir, habría que modificar la expresión literal (sustituyendo *d{7}* por el valor deseado).

3.3.3.3 jsinicio.js

Este archivo está destinado a la comprobación del formulario de autenticación que se encuentra en la página inicial antes de enviarlo al servidor. Además de esto, en él se definen las cookies y las funciones que se encargan de gestionarlas. Esta funcionalidad se ha decidido conservar de la aplicación originaria para posibles desarrollos futuros, pero no se tiene en cuenta en la actualidad.

A continuación, se procederá a explicar las funciones relevantes para nuestro uso de la aplicación:

a) validacionLogin()

Variables:

- cliente: Almacena el valor introducido por el usuario en el campo destinado al Nick en el formulario de autenticación para su comprobación antes de ser enviado al servidor.
- contrasena: Almacena el valor introducido por el usuario en el campo destinado a la contraseña en el formulario de autenticación para su comprobación antes de ser enviado al servidor.

Definición: Esta función se encarga de comprobar que los datos introducidos en el formulario de autenticación de aplicación cumplen con las especificaciones requeridas. Se encarga, a través de llamadas al archivo *funciones.js*, de comprobar que las variables especificadas anteriormente ni están

vacías ni incumplen las restricciones indicadas en cuanto a su longitud se refiere.

Ejecución: La función *validacionLogin()* obtiene los valores introducidos por el usuario en el formulario y los almacena en las dos variables explicadas anteriormente.

Tras esto, valida que tanto el campo *cliente* como el campo *contrasena* estén vacíos o no tengan la longitud requerida. Esto lo realiza llamando a las funciones compruebaVacio y compruebaLongitud del archivo *funciones.js*. Si lo anteriormente explicado ocurre, estas funciones se encargarán de enviar por pantalla un mensaje de error y la función devolverá el valor *false*, no enviándose los datos al servidor. En caso contrario, la función devolverá *true* y el formulario será enviado al servidor para que éste se encargue de procesarlo.

3.3.3.4 compr_nuevo_cliente.js

Archivo encargado de comprobar el formulario de inserción de un nuevo cliente antes de enviar los datos al servidor.

Obtiene los datos insertados por el usuario con rol administrador en el formulario y los almacena en las variables: *dni, cliente, nombre, apellido1, apellido2, direccion, admin y contrasena*.

Tras esto, comprueba:

- Sobre la variable *dni* que no esté vacía (llamando a la función *compruebaVacio* del archivo *funciones.js*) y si esto es correcto que la variable esté formada por 8 caracteres numéricos (llamando a la función *compruebaDNI* del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *cliente* que no esté vacía (llamando a la función *compruebaVacio* del archivo *funciones.js*) y si esto es correcto que la longitud de la variable esté comprendida entre 5 y 12 caracteres (llamando a la función compruebaLongitud del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *nombre* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable sea inferior a 20 caracteres (llamando a la función compruebaMax del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *apellido1* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable sea inferior a 20 caracteres (llamando a la función compruebaMax del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *apellido2* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable sea inferior a 20 caracteres (llamando a la función compruebaMax del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *direccion* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable sea inferior a 40 caracteres (llamando a la función compruebaMax del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *contrasena* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable esté comprendida entre 5 y 12 caracteres (llamando a la función compruebaLongitud del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa, devolviendo *true* y permitiendo el envío del formulario al servidor. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

3.3.3.5 compr_nuevo_libro.js

Archivo encargado de comprobar el formulario de inserción de un nuevo libro antes de enviar los datos al servidor.

Obtiene los datos insertados por el usuario con rol administrador en el formulario y los almacena en las variables: *isbn, titulo, autor, editorial, anio_pub y genero*.

Tras esto, comprueba:

- Sobre la variable *isbn* que no esté vacía (llamando a la función *compruebaVacio* del archivo *funciones.js*) y si esto es correcto que la variable esté formada por 10 caracteres numéricos (llamando a la función *compruebaISBN* del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *titulo* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable sea inferior a 50 caracteres (llamando a la función compruebaMax del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *autor* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable sea inferior a 50 caracteres (llamando a la función compruebaMax del archivo *funciones.js*). Si esto es correcto, continúa

la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *editorial* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la longitud de la variable sea inferior a 50 caracteres (llamando a la función compruebaMax del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *anio_pub* que no esté vacía (llamando a la función compruebaVacio del archivo *funciones.js*) y si esto es correcto que la variable esté formada por 4 caracteres numéricos (llamando a la función compruebaAnioPub del archivo *funciones.js*). Si esto es correcto, continúa la ejecución normal del programa. En caso contrario, la función en la que no se cumpla lo anteriormente indicado lanzará un mensaje por pantalla indicando el error y devolverá false paralizando la ejecución normal.

- Sobre la variable *genero* que se haya seleccionado alguna opción de la lista, es decir, que no esté vacía. Si esto es correcto, continúa la ejecución normal del programa, devolviendo *true* y permitiendo el envío del formulario al servidor. En caso contrario, devolverá *false* y mostrará un mensaje por pantalla solicitando la elección de una opción de la lista de *generos*. El formulario no será enviado al servidor hasta que este error sea subsanado.

3.4 Arquitectura y Diseño por Caso de Uso

La aplicación está estructura siguiendo el patrón Cliente-Servidor y el patrón MVC.

Para explicar dicho diseño, estructuraremos la explicación por Casos de Uso, en los cuales separaremos la explicación en los tres componentes del patrón de arquitectura software MVC.

Es necesario tener en cuenta que los archivos que componen el Controlador son Servlets de Java, encargados de solicitar datos al Modelo y de comunicárselos a la Vista.

3.4.1 Iniciar sesión

La **Vista** se encarga de mostrar el formulario de autenticación del usuario en la aplicación. En éste, aparecen dos campos destinados a ser rellenados por el usuario, uno para el Nick y otro para la Contraseña. Esto está implementado en el archivo *inicio.jsp*.

Tras ser rellenado el formulario y pulsado el botón *Entrar* por parte del usuario, los datos rellenados son comprobados en el archivo *jsinicio.js* antes de ser enviados al servidor.

Si los datos cumplen con los requisitos anteriormente definidos, los datos son enviados al servidor.

El **Controlador**, en el Servlet*Login*, recibe los datos enviados desde la vista.

Éste, en primer lugar, se encarga de comprobar que los datos recibidos cumplen con los requisitos definidos.

Tras esto, y tras llamar al constructor para instanciar un objeto de tipo *Cliente*, llama al Modelo para que consulte en la Base de Datos si el Nick y la Contraseña del usuario existen en la Base de Datos con el fin de permitirle o denegarle el acceso a la aplicación. En concreto, llama al método *login()* de la clase *Cliente*.

El **Modelo** consulta a la Base de Datos y devuelve los datos resultantes de la consulta al Controlador.

Si existe el Nick y la Contraseña en la Base de Datos, el **Controlador** llama al archivo *datos.jsp* de la **Vista**, el cual se encarga de mostrar por pantalla los datos del usuario, almacenados anteriormente en el *Bean Cliente* a través del archivo *login.jsp*.

En caso contrario, si no existen Nick o Contraseña en la Base de Datos, el **Controlador**se encarga de llamar al archivo *inicio_tras_error.jsp* de la Vista, el cual muestra un mensaje indicando que existe un error en las credenciales de autenticación e invitando a introducirlas correctamente. Para poder realizar esto, facilita el de nuevo el formulario preparado para ser rellenado.

3.4.2 Ver sus datos personales

Un usuario puede ver sus datos personales de tres formas diferentes:

a) Al iniciar sesión, como hemos explicado anteriormente.

b) Seleccionando del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Datos*.

c) Pulsando con el botón izquierdo del ratón sobre el logo del Departamento de Ingeniería Telemática, situado en la cabecera de la página.

Tras elegir un usuario ver sus datos personales por cualquiera de las tres formas anteriormente explicadas, la vista pasa los datos al **Controlador** que se encarga de llamar al archivo *datos.jsp* de la **Vista**, el cual se encarga de mostrar por pantalla los datos del usuario, almacenados anteriormente en el *Bean Cliente* a través del archivo *login.jsp*.

3.4.3 Ver listado de clientes

Para ver la lista completa de clientes, el usuario con rol *Administrador* selecciona del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Listado de Clientes*.

El botón del menú seleccionado provoca la invocación del Servlet*Listar_Clientes*del **Controlador**. En él, se instancia, a través de una llamada al constructor, un objeto de tipo *Cliente* para posteriormente crear un *ArrayList*, destinado a almacenar el listado, y llamar almétodo *listar_Clientes()* de la clase *Cliente* perteneciente al **Modelo**, destinado a obtener de la Base de Datos dicho listado**.** Esto último lo realiza invocando al método *listar_Clientes(String consulta)* de la clase *Conexion.java*. Tras esto, le devuelve el listado requerido al Controlador.

Posteriormente, el **Controlador** invoca a la **Vista** para que muestre los datos obtenidos en una tabla diseñada para ello en el archivo *listar_clientes.jsp*. Dichos datos son enviados del Controlador a la Vista a través de la sesión abierta.

Por último, la **Vista**, a través del archivo *listar_clientes.jsp*, muestra en una tabla el listado de clientes que se encuentran registrados en la aplicación.

3.4.4 Ver listado de libros

Para ver la lista completa de libros, el usuario con rol *Administrador* selecciona del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Listado de Libros*.

El botón del menú seleccionado provoca la invocación del Servlet*Listar_Libros*del **Controlador**. En él, se instancia, a través de una llamada al constructor, un objeto de tipo *Cliente* para posteriormente crear un *ArrayList*, destinado a almacenar el listado, y llamar almétodo *listar_Libros()* de la clase *Libro* perteneciente al **Modelo**, destinado a obtener de la Base de Datos dicho listado**.** Esto último lo realiza invocando al método *listar_Libros(String consulta)* de la clase *Conexion.java*. Tras esto, le devuelve el listado requerido al Controlador.

Posteriormente, el **Controlador** invoca a la **Vista** para que muestre los datos obtenidos en una tabla diseñada para ello en el archivo *listar_libros.jsp*. Dichos datos son enviados del Controlador a la Vista a través de la sesión abierta. Además de los datos anteriormente mencionados, también se le envía a la Vista el número de libros incluido en dicho listado. Esto se realiza para poder recorrer el listado de libros con el fin de mostrar en la Vista si cada libro está reservado o disponible para reservar, dando la opción, si el usuario tiene rol de *Cliente* de poder reservarlo. Si el usuario tiene rol de *Administrador*, por cada libro podrá ver si está reservado o disponible, pero en ningún caso reservar un libro.

Por último, la **Vista**, a través del archivo *listar_libros.jsp*, se encarga de mostrar al usuario el listado de libros, con las distintas opciones de reserva que hemos comentado anteriormente.

3.4.5 Ver reservas de todos los clientes

Para ver la lista de todas las reservas, el usuario con rol *Administrador* selecciona del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Reservas*.

El botón del menú seleccionado provoca la invocación del Servlet*Listar_Reservas* del **Controlador**. En él, se instancia, a través de una llamada al constructor, un objeto de tipo *Reserva* para posteriormente crear un *ArrayList*, destinado a almacenar el listado de libros reservados junto con el DNI de los clientes que los tiene reservados, y llamar almétodo *listar_Reservas()* de la clase *Reserva* perteneciente al **Modelo**, destinado a obtener de la Base de Datos dicho listado**.** Esto último lo realiza invocando al método *listar_Reservas(String consulta)* de la clase *Conexion*. Tras esto, le devuelve el listado requerido al Controlador.

Posteriormente, el **Controlador** invoca a la **Vista** para que muestre los datos obtenidos en una tabla diseñada para ello en el archivo *reservas.jsp*. Dichos datos son enviados del Controlador a la Vista a través de la sesión abierta.

Por último, la **Vista**, a través del archivo *reservas.jsp*, muestra en una tabla el listado de las reservadas en curso en el momento en que se realiza la consulta.

3.4.6 Ver sus reservas (cliente)

Para que un usuario con rol *Cliente* vea sus reservas, éste selecciona del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Mis Reservas*.

El botón del menú seleccionadoprovoca la invocación del Servlet*Listar_Reservas_Cliente* del

Controlador. En él, se instancia, a través de una llamada al constructor, un objeto de tipo *Reserva* para posteriormente crear un *ArrayList*, destinado a almacenar el listado de libros reservados por el cliente en cuestión, y llamar almétodo *listar_Reservas_Cliente(Integer dni)* de la clase *Reserva* perteneciente al **Modelo**, destinado a obtener de la Base de Datos dicho listado**.** Esto último lo realiza invocando al método *listar_Reservas(String consulta)* de la clase *Conexion*. Tras esto, le devuelve el listado requerido al Controlador.

Posteriormente, el **Controlador** invoca a la **Vista** para que muestre los datos obtenidos en una tabla diseñada para ello en el archivo *reservas.jsp*. Dichos datos son enviados del Controlador a la Vista a través de la sesión abierta.

Por último, la **Vista**, a través del archivo *reservas.jsp*, muestra en una tabla el listado de los libros que se encuentran reservados por el cliente que realiza la consulta en el momento de la realización de ésta.

3.4.7 Borrar reservas de cualquier cliente

Para poder borrar la reserva de un libro por parte de cualquier cliente, el usuario con rol *Administrador* tiene que encontrarse visualizando el listado de reservas, es decir, el contenido del archivo *reservas.jsp* perteneciente a la **Vista**. En dicha pantalla, el *Administrador* puede borrar una reserva pinchando sobre la opción *Borrar*, situada tras la información de cada reserva.

Al elegir dicha opción, se provoca la invocación de Servlet *Borrar_Reserva*, el cual forma parte del Controlador,enviándole en la URL el *ISBN* del libro cuya reserva se desea eliminar.

Tras esto, el **Controlador** obtiene el *ISBN* del libro anteriormente enviado por la Vista, utilizado en la instanciación de un objeto de tipo *Reserva* y llama al método *borrar_Reserva(Long isbn)* contenido en el **Modelo**, en la clase *Reserva*. Éste, junto con el método *borrar(String sql)* de la clase *Conexion* al cual invoca, se encargan de eliminar la reserva de la Base de Datos.

En el Servlet *Borrar_Reserva*, el **Controlador** comprueba si se ha podido eliminar la reserva de la Base de Datos llamando al archivo *borrar_reserva_correcto.jsp* de la Vista, el cual mostrará un mensaje indicando *que La reserva del libro con ISBN XXXXXXXXXX ha sido realizada con éxito*. En caso contrario, el Controlador llamará al archivo *borrar_reserva_error.jsp* de la Vista, el cual mostrará un mensaje indicando que *No se ha podido eliminar la reserva del libro con ISBN XXXXXXXXXX*.

3.4.8 Borrar reservas realizadas por él (cliente)

Para poder borrar la reserva de un libro por parte de un cliente, el usuario con rol Cliente que tiene realizada la reserva a borrar tiene que encontrarse visualizando el listado de sus reservas, es decir, el contenido del archivo *reservas.jsp* perteneciente a la **Vista**. En dicha pantalla, el *Cliente* puede borrar una reserva pinchando sobre la opción borrar, situada tras la información de cada reserva.

Al elegir dicha opción, se provoca la invocación de Servlet *Borrar_Reserva*, el cual forma parte del Controlador, enviándole en la URL el *ISBN* del libro cuya reserva se desea eliminar.

Tras esto, el **Controlador** obtiene el *ISBN* del libro anteriormente enviado por la Vista, utilizado en la instanciación de un objeto de tipo *Reserva* y llama al método *borrar_Reserva(Long isbn)* contenido en el **Modelo**, en la clase *Reserva.* Éste, junto con el método *borrar(String sql)* de la clase *Conexion* al cual invoca, se encargan de eliminar la reserva de la Base de Datos.

En el Servlet *Borrar_Reserva*, el **Controlador** comprueba si se ha podido eliminar la reserva de la Base de Datos. En caso afirmativo, llama al archivo *borrar_reserva_correcto.jsp* de la **Vista**, el cual

mostrará un mensaje indicando *que La reserva del libro con ISBN XXXXXXXXXX ha sido realizada correctamente*. En caso contrario, el Controlador llamará al archivo *borrar_reserva_error.jsp* de la **Vista**, el cual mostrará un mensaje indicando que *No se ha podido eliminar la reserva del libro con ISBN XXXXXXXXXX*.

3.4.9 Reservar libro

Para poder reservar un libro, el usuario con rol Cliente tiene que encontrarse visualizando el listado de libros, es decir, el contenido del archivo *listar_libros.jsp* perteneciente a la **Vista**. En dicha pantalla, el *Cliente* puede realizar una reserva pinchando sobre la opción Reservar, habilitada si el libro no está reservado en el momento de realizar la reserva.

Al elegir dicha opción, se se provoca la invocación de Servlet *Reservar_Libro*, el cual forma parte del Controlador,enviándole en la URL el *ISBN* del libro que el *Cliente* desea reservar.

Tras esto, el **Controlador** obtiene el *ISBN* del libro anteriormente enviado por la Vista y el *DNI* del cliente del *Bean Cliente*, utilizados en la instanciación de un objeto de tipo *Reserva* y llama al método *reservar_Libro()* contenido en el **Modelo**, en la clase *Reserva*. Éste, junto con el método *insertar(String sql)* de la clase *Conexión*, al cual invoca, se encargan de insertar la reserva de la Base de Datos.

En el Servlet *Reservar_Libro*, el **Controlador** comprueba si se ha podido realizar la reserva correctamente en la Base de Datos. En caso afirmativo, llama al archivo *reserva_libro_correcto.jsp* de la **Vista**, el cual mostrará un mensaje indicando *que La reserva del libro con ISBN XXXXXXXXXX ha sido realizada correctamente*. En caso contrario, el Controlador llamará al archivo *reserva_libro_error.jsp* de la **Vista**, el cual mostrará un mensaje indicando que *No se ha podido realizar la reserva del libro con ISBN XXXXXXXXXX*.

3.4.10 Insertar libro

Para insertar un nuevo libro, el usuario con rol *Administrador* selecciona del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Insertar Libro*.

La **Vista**, a través del archivo *nuevo_libro.jsp*, muestra por pantalla al usuario el formulario de inserción de un nuevo libro.

En el archivo anteriormente mencionado, se ha implementado el antipatrón MVC comentado en apartados anteriores. Este antipatrón se basa en no tener en cuenta las distintas capas del patrón de diseño software *MVC*. Esto es realizado en la funcionalidad del género del nuevo libro. En la **Vista**, coexiste tanto presentación como lógica. El género de un nuevo libro es cargado en una lista desplegable. Para ello, se consulta a la Base de Datos el contenido de la tabla *generos*. El resultado de esta consulta se almacena en un ArrayList. Tras esto, se prepara la lista desplegable para que pueda mostrar al *Administrador* las distintas opciones de género a elegir al rellenar el formulario de inserción de un nuevo libro en la aplicación.

Cuando el *Administrador* haya acabado de rellenar el formulario y pulsado el botón *Insertar*, los datos rellenados son comprobados en el archivo *compr_nuevo_libro.js* antes de ser enviados al servidor.

Si los datos cumplen con los requisitos anteriormente definidos, éstos son enviados al servidor.

Esto se realiza invocando al Servlet *Insert_libro*, el cual recibe los datos enviados desde la vista. Este Servlet forma parte del **Controlador**.

Éste, en primer lugar, se encarga de comprobar que los datos recibidos cumplen con los requisitos definidos.

Tras esto, y tras llamar al constructor para instanciar un objeto de tipo *Libro* (pasándole como parámetros todos los datos del nuevo libro), llama al **Modelo** para que inserte el nuevo libro en la Base de Datos. En concreto, llama al método *insertar_Libro()* de la clase *Libro*. Éste llama al método *insertar(String sql)* de la clase *Conexion* para realizar la inserción del nuevo libro en la Base de Datos.

En el Servlet *Insert_libro*, el **Controlador** comprueba si se ha podido insertar el nuevo libro correctamente en la Base de Datos. En caso afirmativo, llama al archivo *ins_libro_correcto.jsp* de la **Vista**, el cual mostrará un mensaje indicando *que El libro con ISBN XXXXXXXXXX ha sido insertado correctamente*. En caso contrario, el Controlador llamará al archivo *ins_libro_error.jsp* de la **Vista**, el cual mostrará un mensaje indicando que *No se ha podido insertar el libro con ISBN XXXXXXXXXX*.

3.4.11 Insertar cliente

Para insertar un nuevo cliente, el usuario con rol *Administrador* selecciona del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Insertar Cliente*.

La **Vista**, a través del archivo *nuevo_cliente.jsp*, muestra por pantalla al usuario el formulario de inserción de un nuevo cliente.

Cuando el *Administrador* haya acabado de rellenar el formulario y pulsado el botón *Insertar*, los datos rellenados son comprobados en el archivo *compr_nuevo_cliente.js* antes de ser enviados al servidor.

Si los datos cumplen con los requisitos anteriormente definidos, éstos son enviados al servidor.

Esto se realiza invocando al Servlet *Insert_cliente*, el cual recibe los datos enviados desde la vista. Este Servlet forma parte del **Controlador**.

Éste, en primer lugar, se encarga de comprobar que los datos recibidos cumplen con los requisitos definidos.

Tras esto, y tras llamar al constructor para instanciar un objeto de tipo *Cliente* (pasándole como parámetros todos los datos del nuevo cliente), llama al **Modelo** para que inserte el nuevo cliente en la Base de Datos. En concreto, llama al método *insertar_Cliente()* de la clase *Cliente*. Éste llama al método *insertar(String sql)* de la clase *Conexion* para realizar la inserción del nuevo cliente en la Base de Datos.

En el Servlet *Insert_cliente*, el **Controlador** comprueba si se ha podido insertar el nuevo cliente correctamente en la Base de Datos. En caso afirmativo, llama al archivo *ins_cliente_correcto.jsp* de la **Vista**, el cual mostrará un mensaje indicando *que El cliente con DNI YYYYYYYY ha sido insertado correctamente*. En caso contrario, el Controlador llamará al archivo *ins_cliente_error.jsp* de la **Vista**, el cual mostrará un mensaje indicando que *No se ha podido insertar el cliente con DNI YYYYYYYY*.

3.4.12 Cerrar sesión

El usuario selecciona del menú de la **Vista**, implementado en el archivo *menu.jsp* y mostrado en la mayoría de páginas de la aplicación, la opción *Salir*.

Tras esto, se ejecuta el archivo *logout.jsp*, el cual se encarga de cerrar la sesión y redirigir a la página de inicio.

3.5 Librerías usadas en el desarrollo de la aplicación

En el desarrollo de la aplicación se hace uso de librerías, las cuales son de gran ayuda en la programación y ejecución de ésta.

3.5.1 Librerías de Java

- **java.sql:** Es usada para acceder a Bases de Datos relacionales y realizarle consultas a las mismas. Es una librería implementada por la API de JDBC.
- **java.util:** Es una de las bibliotecas más usadas en la programación en Java. Proporciona un conjunto de clases de gran potencialidad como *Calendar*, *Date* o *HashMap*, así como interfaces de gran relevancia como *Collection*, *List*, *Iterator*, *Comparator*, *Set* o *Map*.
- **java.io:** Es la librería encargada de gestionar las operaciones de entrada/salida. La entrada estándar es *System.in*, la salida estándar es *System.out* y la salida de errores es *System.err*.
- **javax.servlet:** Es la librería encargada de gestionar todo lo relacionado con los Servlets, especialmente lo relacionado con la ejecución de éstos.

3.5.2 Librería de PostgreSQL – JDBC

- **postgresql.jar:** Librería del driver JDBC. Ésta es usada en la conexión entre Java y la Base de Datos y contiene todo lo necesario para poder realizar lo anteriormente mencionado correctamente.

3.5.3 Librería de JSTL

- **jstl-1.2.jar:** Librería de etiquetas con utilidades ampliamente utilizadas en el desarrollo de páginas web dinámicas. En el caso concreto de la aplicación, su funcionalidad radica en permitir implementar lógica en la capa de la Vista.

3.6 JavaDoc

Junto a este documento se podrá consultar la documentación en formato JavaDoc de la parte del servidor de la aplicación generada a partir de los comentarios existentes en el código de la misma.

3.7 Código de creación de la Base de Datos

Este apartado será dedicado a mostrar el código de creación de la Base de Datos, así como la inserción de una serie de datos introducidos en las distintas tablas a modo de ejemplo.

El código sql de creación de la Base de Datos y de las respectivas tablas que ésta contiene es el siguiente:

```
--
--  Creación de la BBDD
--

DROP DATABASE IF EXISTS reserva_libros;
CREATE DATABASEreserva_libros;



-- Table: clientes

-- DROP TABLE clientes;   -- Usar en el caso de que la tabla a crear exista previamente.

CREATE TABLE clientes
(
dni integer NOT NULL UNIQUE, -- DNI del cliente
cliente character(100) NOT NULL UNIQUE, -- Login del cliente
nombre character(100) NOT NULL, -- Nombre del cliente
apellido1 character(100) NOT NULL, -- Primer apellido del cliente
apellido2 character(100), -- Segundo apellido del cliente
direccion character(200), -- Dirección del domicilio del cliente
admin bit(1) NOT NULL DEFAULT (0)::bit(1), -- Valor booleano. Verdadero (1) si es un administrador. Por defecto 0.
contrasena character(20) NOT NULL, -- Contraseña del cliente
 CONSTRAINT clientes_pkey PRIMARY KEY (dni)
)
WITH (
OIDS=FALSE
);



-- Table: generos

-- DROP TABLE generos;   -- Usar en el caso de que la tabla a crear exista previamente.

CREATE TABLE generos
(
genero character(100) NOT NULL UNIQUE, -- Género/temática del libro
```

```
CONSTRAINT generos_pkey PRIMARY KEY (genero)
)
WITH (
OIDS = FALSE
);


-- Table: libros

-- DROP TABLE libros;   -- Usar en el caso de que la tabla a crear exista previamente.

CREATE TABLE libros
(
isbn bigint NOT NULL UNIQUE, -- ISBN del libro
titulo character(100) NOT NULL, -- Título del libro
autor character(100), -- Autor/es del libro
 editorial character(100), -- Editorial del libro
anio_pub integer, -- Año de publicación del libro
genero character(100), -- Género/temática del libro
observaciones character(200), -- Observaciones adicionales para reservar el libro
CONSTRAINT libros_pkey PRIMARY KEY (isbn),
  CONSTRAINT libros_fkey FOREIGN KEY (genero)
    REFERENCES generos (genero) MATCH SIMPLE
    ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
OIDS=FALSE
);


-- Table: reservas

-- DROP TABLE reservas;   -- Usar en el caso de que la tabla a crear exista previamente.

CREATE TABLE reservas
(
```

```
isbn bigint NOT NULL UNIQUE, -- ISBN del libro reservado
 dni integer NOT NULL, -- DNI del cliente
  CONSTRAINT reservas_pkey PRIMARY KEY (isbn),
  CONSTRAINT reservas_isbn_fkey FOREIGN KEY (isbn)
     REFERENCES libros (isbn) MATCH SIMPLE
     ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT reservas_dni_fkey FOREIGN KEY (dni)
     REFERENCES clientes (dni) MATCH SIMPLE
     ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
OIDS=FALSE
);
```

A continuación, se mostrará el código de inserción de datos de ejemplo introducidos en las distintas tablas:

```
--
-- Inserción de datos
--

-- Introducir datos en la tabla clientes

INSERT INTO public.clientes(
 dni, cliente, nombre, apellido1, apellido2, direccion, admin,
        contrasena)
VALUES ('80169876', 'jose22', 'Jose', 'Ruíz', 'López', 'Calle Ancha 22', '0',
        'joselito22');
INSERT INTO public.clientes(
dni, cliente, nombre, apellido1, apellido2, direccion, admin,
        contrasena)
VALUES ('75202987', 'raulgonzalez', 'Raúl', 'González', 'Blanco', 'Calle Mayor 22', '0',
        'gonzalez7');
INSERT INTO public.clientes(
dni, cliente, nombre, apellido1, apellido2, direccion, admin,
        contrasena)
```

```
VALUES ('80159694', 'fabian', 'Fabián', 'Muñoz', 'Peñas', 'Calle San Bartolomé nº39', '1','fabi9');


-- Introducir datos en la tabla generos


INSERT INTO public.generos(
genero)
VALUES ('Informática');
INSERT INTO public.generos(
genero)
VALUES ('Narrativa');
INSERT INTO public.generos(
genero)
VALUES ('Ficción');


-- Introducir datos en la tabla libros

INSERT INTO public.libros(
isbn, titulo, autor, editorial, anio_pub, genero)
VALUES ('9788401328','Los pilares de la tierra', 'Ken Follet', 'PLAZA & JANES EDITORES', '2006', 'Ficción');
INSERT INTO public.libros(
isbn, titulo, autor, editorial, anio_pub, genero)
VALUES ('9788408008','La conjura de Cortés', 'Matilde Asensi', 'PLANETA', '2012', 'Narrativa');
INSERT INTO public.libros(
isbn, titulo, autor, editorial, anio_pub, genero)
VALUES ('9788448198','Programación en C(2ªEd.)', 'Luis Joyanes Aguilar', 'MCGRAW-HILL / INTERAMERICANA DE ESPAÑA', '2005', 'Informática');
INSERT INTO public.libros(
isbn, titulo, autor, editorial, anio_pub, genero)
VALUES ('9788408099','La conspiración', 'Dan Brown', 'PLANETA', '2012', 'Narrativa');


-- Introducir datos en la tabla reservas
```

```
INSERT INTO public.reservas(
isbn, dni)
VALUES ('9788401328', '80169876');
INSERT INTO public.reservas(
isbn, dni)
VALUES ('9788448198', '80159694');
```

3.8 Métodos de trabajo

3.8.1 Orden seguido en el desarrollo del Proyecto

En este apartado, se va a proceder a explicar brevemente el método de trabajo seguido en el desarrollo de la aplicación.

En primer lugar, y tras entender las distintas funcionalidades que tenía que cumplir la aplicación, se procedió a comenzar a realizar el **diseño** de la arquitectura que debía tener la aplicación.

Se comenzó **definiendo** los distintos actores y sus funciones, los requisitos tanto funcionales como no funcionales y las distintas restricciones a cumplir por la aplicación.

Tras esto, se procedió a instalar y configurar los diferentes programas útiles que formarán parte del entorno de desarrollo de la aplicación.

El paso siguiente consistió en el **diseño de la Base de Datos**, estructurando dicha Base de Datos en cuatro tablas (clientes, libros, reservas y generos) relacionadas entre sí. En este paso, fue de gran ayuda el programa *pgAdmin*.

Posteriormente, **se creó** el proyecto en Eclipse, **configurando** el Contenedor de Servlets *Apache Tomcat*, el conector con la Base de Datos *JDBC* o el contenido del fichero *web.xml* entre otros elementos necesarios para poder desarrollar la aplicación.

Tras esto, y como los conceptos relacionados con el patrón MVC no los tenía comprendidos totalmente, se procedió a **implementar** algún Caso de Uso. En esta primera versión se encontraban en el mismo fichero tanto la representación visual como la lógica. Con este paso, comencé a tener claro qué tenía que hacer y cómo lo iba a llevar a cabo.

Posteriormente, se comenzó a desarrollar la aplicación estructurándola conforme a las especificaciones requeridas, en especial a los patrones de diseño software Cliente-Servidor y MVC. Para ello, fui desarrollando de forma simultánea:

- En el lado del **Modelo**, se crearon las distintas clases, a las cuales, posteriormente, se les fueron añadiendo los respectivos métodos que se encuentran incluidos en ellas conforme fueron siendo necesitados.
- En la capa de la **Vista**, se crearon y adaptaron los distintos archivos encargados de mostrar los datos al usuario.
- En la capa del **Controlador**, se crearon las distintas llamadas a las funciones. así como la preparación para la recepción y envío de datos tanto a/de la Vista como del Modelo.

El siguiente paso consistió en desarrollar las **comprobaciones** de los datos insertados en los formularios, tanto del lado Cliente, en archivos escritos en lenguaje JavaScript, como del lado Servidor, en el Controlador.

Tras esto, se realizaron **pruebas** a la aplicación para verificar que su funcionamiento era el especificado anteriormente, solucionando los puntos que no pasaron las pruebas. Además, aunque algunos puntos superasen las pruebas, se buscó optimizar la aplicación con prácticas como la reutilización de código haciendo uso de funciones para evitar código repetido.

Seguidamente, se generó el archivo .WAR para su despliegue en cualquier servidor, sin necesidad de que tener que estar ejecutando el servidor del Entorno de Desarrollo. El proceso tanto de generación del .WAR como de manejo del servidor son explicados en apartados posteriores.

Por último, se procedió a **documentar** lo realizado, tanto en JavaDoc y con los distintos comentarios a nivel de código como en una memoria explicativa del trabajo realizado. Además, se realizó un documento explicativo, habilitado en el apartado de Anexos de esta memoria, para el arranque del servicio PostgreSQL tanto en Windows como en Linux Ubuntu, así como un tutorial de manejo de Base de Datos Relacional. En este último se realizan explicaciones acerca del mantenimiento y gestión de la base de datos.

3.8.2 Depuración de código

Una práctica muy útil para la detección de errores en la programación es la depuración de código. Podríamos diferenciar la realización de esta tarea en dos partes:

- **Depuración en la parte del *Cliente*:** Para llevar a cabo esta depuración, se ha usado el depurador de los navegadores web Mozilla Firefox y Google Chrome. La manera de realizarlo ha sido ejecutar la aplicación de forma normal y abrir el inspector de elementos del navegador en cuestión. La aplicación se para en el punto seleccionado y es posible ver, e incluso modificar, los distintos elementos, como valores de variables o resultados de funciones, en tiempo de ejecución. Este depurador también permite medir tiempos y rendimientos.

- **Depuración en la parte del *Servidor*:** Para llevar a cabo esta depuración, se ha arrancado el servidor instalado en Eclipse en modo depuración y, con la ayuda de puntos de ruptura, sistema de trazas(logs) e impresiones de texto por pantalla y por consola, se ha podido detectar y subsanar los distintos errores o mejoras que han ido surgiendo.
  En la gestión de errores a través de un sistema de trazas o logs, definida en el *ReqDoc-005 Gestión de Logs*, se hace uso de las librerías *java.util.logging.level* y *java.util.logging.Logger*. Esto es usado en la comprobación en el Servidor de los datos insertados en los distintos formularios. Además de esto, se hace uso de un manejador de excepciones, también definido en el requisito mencionado anteriormente.

  El depurador aporta una gran serie de funcionalidades. Las más utilizadas al depurar la aplicación han sido la de seguir paso a paso el hilo del programa y la de ver el valor de una variable en un punto concreto del programa o el valor devuelto por una función tras su ejecución recibiendo una serie de valores concretos.

3.9 Problemas encontrados y soluciones

Este apartado ha sido creado con el fin de recopilar de forma breve los problemas surgidos en el desarrollo del proyecto. A fin de simplificar, sólo serán recogidos y explicados los problemas

generales más significativos, sin entrar en profundidad en problemas concretos de programación. Junto a éstos, será incluida una explicación de los procedimientos llevados a cabo para solventarlos.

A continuación, se procederá a describir lo anteriormente mencionado:

- Siguiendo un orden cronológico, el primer problema encontrado en el desarrollo de la aplicación es el de realizar un análisis concreto y claro de lo que realmente necesita cumplir la aplicación. Es decir, la definición de actores, requisitos tanto funcionales como no funciones, así como la relación entre estos.

  El desarrollo de esta aplicación ha sido una toma de contacto con un proyecto real completo, ya que en las prácticas de las distintas asignaturas se suelen entregar a los alumnos las tareas que tienen que hacer con un análisis previo realizado y con los problemas a resolver detallados de forma relativamente clara y concisa.

  Este problema fue resuelto estudiando en profundidad los apuntes de la asignatura Ingeniería del Software, la página Marco de Desarrollo de Software de la Junta de Andalucía y otros tutoriales relacionados con este tema.

- El mayor problema y más difícil de solventar ha sido conceptual. Fue relativamente sencillo realizar una aplicación que más o menos realizase lo solicitado en los requisitos anteriormente mencionados, pero fue más complicado estructurar la aplicación conforme a los patrones de diseño y desarrollo software (Cliente – Servidor y Modelo – Vista – Controlador) requerían. Éste era uno de los principales requisitos que tenía que cumplir la aplicación debido a ser estos patrones parte importante del material docente de las distintas asignaturas que harán uso de la aplicación.

  Para resolver este problema, la clave principal ha sido llegar a comprender de forma conceptual los distintos patrones de diseño software. Tras esto, gracias al estudio de información proveniente de muy diversas fuentes y siguiendo el método de *prueba y error*, se ha podido implementar la aplicación conforme a lo requerido.

- Otro problema conceptual ha sido el de usar Java Bean para gestionar el traspaso de información entre Servlet y JSP, así como para obtener los datos del usuario al autenticarse.

  Tras estudiar dicho concepto, fue considerada ésta como la mejor opción a implementar en estos casos, pero fue complicado llegar a dicha conclusión debido al desconocimiento de dicho concepto y a lo poco intuitivo que lo considero.

- En cuanto a los problemas de instalación, puede ser destacado el conectar la aplicación con la Base de Datos. Dicha conexión es realizada a través de un driver llamado JDBC.

  Tras instalar y configurar dicho driver siguiendo el tutorial diseñado para ello, no era posible acceder a la Base de Datos desde la aplicación debido a que la aplicación no encontraba el driver JDBC.

  Tras horas de investigación y pruebas, la solución fue encontrada. Dicha solución pasaba por colocar el archivo .jar de dicho driver en *WEB-INF/lib*. En esta carpeta es donde la aplicación buscar el driver para ejecutarlo cuando es necesario.

- Debido a que no puede haber lógica en la capa de la Vista de la aplicación, no puede ser usado Java en dicho componente del MVC. Este fue un problema que reportó grandes quebraderos de cabeza.

  La forma de solventarlo ha sido haciendo uso, en la Vista, del lenguaje EL (Expression Languaje), el cual es capaz de incrustar expresiones en páginas web, haciendo uso en la Vista de lógica, pero respetando el patrón MVC, es decir, sin ejecutar Java en dicho componente de MVC.

4 LÍNEAS FUTURAS

Este apartado está reservado para indicar todas aquellas evoluciones pensadas para ser desarrolladas en el futuro.

Estas ideas no tienen por qué ser desarrolladas obligatoriamente, pero pueden servir de ayuda a la hora de decidir mejorar la aplicación.

A continuación, se procederá a explicar las ideas mencionadas anteriormente:

- Debido al **uso de repositorios** para almacenar el código (en distintas versiones ordenadas por fecha) en las distintas asignaturas en las que se utilizará la aplicación, ésta también tendría que ser almacenada de dicha forma. Esto facilitaría el acceso de los alumnos al código y de los profesores a la gestión del mismo, así como el trabajo de los usuarios de forma simultánea.

  A lo largo del desarrollo de la aplicación se investigó al respecto y se estudió cómo debería de estructurarse la aplicación dentro del repositorio, así como la orientación a la docencia de dicha tarea, pero finalmente se optó por dejar ésta para realizarla más adelante.

  Como, en la actualidad, se hace uso de *Git*, lo más conveniente es seguir usando este repositorio en lo concerniente a la aplicación.

- **Batería de pruebas** para comprobar que el sistema actúa como ha sido especificado con anterioridad. Un programa recomendable para la gestión de dichas pruebas es *Selenium*.

- En cuanto a la funcionalidad de la aplicación, se recogen las siguientes ideas:
  - **Disponibilidad de varios ejemplares por cada libro** para que pudieran ser reservados por los usuarios.

    Esto podría realizarse añadiendo para el libro un campo contador que se fuera actualizando (creciendo cuando se devolviera un libro y decreciendo cuando se retirase. En el caso de que éste fuera igual a cero, estarían todos los ejemplares de dicho libro reservados).

  - **Tiempo de reserva**, es decir, una vez que un usuario reserve un libroel tiempo que éste puede disponer de dicho libro antes de devolverlo.

    Esto podría realizarse añadiendo para las reservas un campo de tipo fecha para la fecha de devolución del libro reservado. En él se registraría la fecha de devolución del libro en cuestión, resultado de la fecha en la que se realizó la reserva más el tiempo que el usuario puede disponer del libro.

  - **Posibilidad de renovación de la reserva de un libro antes de su devolución.**

    Esta idea podría ser implementada con un botón *Renovar reserva* por libro reservado. Cuando el usuario seleccionase dicho botón, podría renovar la reserva del libro en

cuestión.

Asociado a esto, llevaría un contador que no permitiese renovar la reserva del libro más de una cantidad fijada de veces, por ejemplo una vez.

Además, debería de añadirse la opción de que un usuario pudiese ponerse en **lista de espera** de un libro reservado. En este caso, si hay algún usuario en la lista de espera de un libro, cuando el usuario que tiene reservado el libro quiera renovarlo no podrá hacerlo.

Esto último se podría realizar añadiendo un campo boleano que se actualizase tanto cuando un usuario se apunta en la lista de reserva del libro en cuestión como cuando se borra de ésta y, en función del resultado de dicho campo, permitir o no la renovación del libro en cuestión.

- **Historial de reservas de los clientes**, el cual muestre todas las reservas que ha realizado un cliente desde que se registró en la aplicación hasta el momento en el que se realiza la consulta.

  Esto podría ser realizado creando una nueva tabla en la Base de Datos. A esta nueva tabla, llamada *Reservas expiradas*, serían trasferidas todas las reservas cuyo libro haya sido devuelto.

5 BIBLIOGRAFÍA / RECURSOS

Para la realización de este apartado, se ha seleccionado el material, tanto en formato físico como en formato digital, de mayor interés entre todas las fuentes consultadas, debido a que éstas eran muy numerosas.

Además, se ha decidido dividir dicho material en función de su utilización (patrón de diseño software, lenguaje de programación, programa utilizado… con el que el material esté relacionado). En ocasiones, esta tarea ha resultado algo compleja debido a que a la mayoría de enlaces es complicado "encasillarlos" en un único apartado.

En este material, se encuentran tanto referencias relacionadas con conceptos como referencias relacionadas con resolución de los distintos problemas que han ido surgiendo en el desarrollo del proyecto.

Tras esta aclaración, se procederá a detallar el material seleccionado:

5.1 Uso general

- Apuntes de la asignatura Ingeniería del Software.
- Apuntes de la asignatura Fundamentos de Aplicaciones y Servicios Telemáticos.
- Apuntes de la asignatura Diseño de Bases de Datos.
- Apuntes de la asignatura Fundamentos de Programación II.

5.2 Análisis de la aplicación - Requisitos

- http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf
- http://www.pmoinformatica.com/2017/02/requerimientos-funcionales-ejemplos.html
- ftp://ftp.inf.utfsm.cl/pub/linux/Docs/LuCaS/Tutoriales/doc-dise%F1o-software/doc-dise%F1o-software-parte-1.pdf
- http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/407
- http://www.lsi.us.es/~amador/publicaciones/metodologia_elicitacion_2_3.pdf.zip

5.3 Patrones de Diseño Software

5.3.1 Cliente – Servidor

- https://es.wikipedia.org/wiki/Cliente-servidor
- http://nereida.deioc.ull.es/~cleon/doctorado/tic02/cs.html
- http://ocw.pucv.cl/cursos-1/arquitectura-de-sistemas-de-software/materiales-de-clases/web-cliente-servidor
- https://www.programacion.com.py/varios/arquitectura-cliente-servidor

- http://ingsoftluisf.blogspot.com.es/2012/11/cliente-servidor.html
- https://robiniclienteservidor.weebly.com/ventajas---desventajas.html
- http://www.inf-cr.uclm.es/www/mpolo/asig/0708/tutorJavaWeb.pdf

5.3.2 Modelo – Vista – Controlador

- https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador
- https://book.cakephp.org/2.0/es/cakephp-overview/understanding-model-view-controller.html
- https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html
- https://desarrolloweb.com/articulos/que-es-mvc.html
- https://codigofacilito.com/articulos/mvc-model-view-controller-explicado
- http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122#Ventajas_y_desventajas_del_uso_del_patron
- https://prezi.com/zcsejbxmrjgm/modelo-vista-controlador/
- http://rodrigogr.com/blog/modelo-vista-controlador/
- https://www.arquitecturajava.com/patron-mvc-arquitectura-cliente-vs-servidor/
- https://es.stackoverflow.com/questions/32431/c%C3%B3mo-enviar-datos-de-una-pagina-jsp-a-un-servlet-sin-un-form
- https://es.stackoverflow.com/questions/33250/c%C3%B3mo-puedo-guardar-un-objeto-en-la-sesi%C3%B3n-y-c%C3%B3mo-utilizarlo-en-un-jsp-servle

5.4 Lenguajes de Programación

5.4.1 HTML

- http://definicion.de/html/
- http://www.acercadehtml.com/manual-html/que-es-html.html
- https://desarrolloweb.com/articulos/que-es-html.html
- https://es.wikipedia.org/wiki/HTML
- http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=435:ique-es-y-para-que-sirve-html-el-lenguaje-mas-importante-para-crear-paginas-webs-html-tags-cu00704b&catid=69&Itemid=192
- http://www.hazunaweb.com/curso-de-html/estructura-basica-una-pagina-web/
- http://fpsalmon.usc.es/genp/doc/cursos/html/estructura.html
- http://platea.pntic.mec.es/~abercian/guiahtml/estructura.htm
- https://stackoverflow.com/questions/17892094/how-to-add-html-editor-to-eclipse
- https://www.w3schools.com/html/default.asp
- http://www.maestrosdelweb.com/dom/

5.4.2 CSS

- http://librosweb.es/libro/css/capitulo_1.html
- http://es.html.net/tutorials/css/lesson1.php
- https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada
- http://www.arumeinformatica.es/dudas/css/
- http://www.masadelante.com/faqs/css
- http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=546:que-es-y-para-que-sirve-el-lenguaje-css-cascading-style-sheets-hojas-de-estilo&catid=46:lenguajes-y-entornos&Itemid=163
- http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo
- https://www.w3schools.com/css/default.asp
- http://www.htmlhelp.com/es/reference/css/properties.html

5.4.3 JavaScript

- http://librosweb.es/libro/javascript/capitulo_1.html
- https://es.wikipedia.org/wiki/JavaScript
- https://desarrolloweb.com/javascript/
- http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=777:ique-es-javascript-principales-usos-servidor-y-cliente-html-css-y-programacion-efectos-cu01103e&catid=78:tutorial-basico-programador-web-javascript-desde-&Itemid=206
- https://devcode.la/blog/que-es-javascript/
- https://norfipc.com/inf/que-es-lenguaje-javascript-introduccion-usos-practicos.html
- http://www.maestrosdelweb.com/que-es-javascript/
- https://www.uv.es/jac/guia/jscript/javascr01.htm
- http://nereida.deioc.ull.es/~pcgull/hli04/js/node2.html
- http://www.4rsoluciones.com/blog/para-que-se-utiliza-javascript-2/
- http://culturacion.com/para-que-se-utiliza-javascript/
- http://manualesdelaweb.com/a/que-es-javascript-para-que-sirve-y-que-puedo-hacer-con-el-20110117
- https://www.campusmvp.es/recursos/post/5-razones-por-las-que-todo-programador-deberia-aprender-JavaScript.aspx
- https://www.w3schools.com/js/
- https://www.aprenderaprogramar.com/foros/index.php?topic=4466.0
- http://librosweb.es/libro/javascript/capitulo_7/validacion.html
- http://regexpre.orgfree.com/
- https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/RegExp/test
- http://lineadecodigo.com/java/digitos-de-un-numero/
- https://www.bonaval.com/kb/wp-content/uploads/2011/06/TutorialExpresionesRegularesJavaScript.pdf

5.4.4 JSP

- https://users.dcc.uchile.cl/~jbarrios/servlets/jsp.html
- http://www.suarezdefigueroa.es/manuel/IAW/Java/teoriajsp.html
- http://geneura.ugr.es/~jmerelo/JSP/
- https://es.wikipedia.org/wiki/JavaServer_Pages
- http://html.rincondelvago.com/jsp.html
- https://desarrolloweb.com/articulos/831.php
- http://www.ciberaula.com/articulo/jsp
- https://www.tutorialspoint.com/jsp/index.htm

5.4.5 EL y JSTL

- https://es.wikipedia.org/wiki/Expression_Language
- https://www.tutorialspoint.com/jsp/jsp_expression_language.htm
- http://www.programacionj2ee.com/el-expression-language/
- http://docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html
- https://www.guru99.com/jsp-expression-language.html
- https://www.tutorialspoint.com/jsp/jsp_expression_language.htm
- https://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html
- https://www.javatpoint.com/jstl
- https://www.journaldev.com/2090/jstl-tutorial-jstl-tags-example
- http://java.ciberaula.com/articulo/introduccion_jstl
- http://www.suarezdefigueroa.es/manuel/IAW/Java/teoriajstl.html
- https://www.adictosaltrabajo.com/tutoriales/jspel/
- http://www.pinchaiconos.com/n/3/libreria-functions-de-jstl
- https://stackoverflow.com/questions/24330155/set-request-attribute-using-jstl
- https://stackoverflow.com/questions/21480040/is-request-setattribute-necessary-with-el
- http://desarrollo-java.readthedocs.io/es/latest/tutorial6.md.html
- http://www.java2s.com/Code/Java/JSTL/JSTLIfElse.htm
- https://stackoverflow.com/questions/2811626/evaluate-empty-or-null-jstl-c-tags

5.4.6 Java

- https://es.wikipedia.org/wiki/Java_(lenguaje_de_programación)
- https://www.ibm.com/developerworks/ssa/java/tutorials/j-introtojava1/
- https://docs.oracle.com/javase/tutorial/java/index.html
- https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Caracter%C3%ADsticas_del_lenguaje
- http://dis.um.es/~bmoros/Tutorial/parte2/cap2-5.html
- http://definicion.de/java/
- http://profesores.fi-b.unam.mx/carlos/java/java_basico1_1.html

- http://www.lenguajes-de-programacion.com/programacion-java.shtml
- http://personales.upv.es/rmartin/cursoJava/Java/Introduccion/PrincipalesCaracteristicas.htm
- https://prezi.com/ze6-3u0y745u/por-que-java-es-seguro/
- http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=368:ique-es-java-concepto-de-programacion-orientada-a-objetos-vs-programacion-estructurada-cu00603b&catid=68&Itemid=188
- https://espanol.answers.yahoo.com/question/index?qid=20090728113501AA3VQ6j
- https://www.digitalocean.com/community/tutorials/instalar-java-en-ubuntu-con-apt-get-es
- https://www.youtube.com/watch?v=PvzRQoqCQME
- http://chuwiki.chuidiang.org/index.php?title=Consulta_de_Base_de_Datos_desde_JSP
- http://chuwiki.chuidiang.org/index.php?title=Consulta_de_Base_de_Datos_con_JSP_usando_una_Clase_Java_separada
- http://www.convertdatatypes.com/Convert-boolean-to-byte-in-Java.html
- http://www.forosdelweb.com/f45/variables-sesion-con-java-672347/
- http://labojava.blogspot.com.es/2012/05/jdbc-conexion-con-base-de-datos.html
- https://institucional.us.es/foros/read.php?60,87394,87419,quote=1
- http://java-white-box.blogspot.com.es/2012/05/eclipse-como-generar-javadoc-con-el.html
- Pérez Menor, J.M.; Carretero Pérez, J.; García Carballeira, F. y Pérez Lobato, J.M. (2003). *Problemas Resueltos de Programación en lenguaje Java*. Madrid: Thomson.
- Arnold, K.; Gosling, J. y Holmes, D. (2001). *El lenguaje de programación Java*. Pearson Educación, S.A.
- García de Jalón, J.; Rodríguez, J.I.; Mingo, I.; Imaz, A.; Brazález, A.; Larzabal, A.; Calleja, J. y García, J (Enero 2000). *Aprenda Java como si estuviera en primero*. San Sebastián: Campus Tecnológico de la Universidad de Navarra.

5.4.7 Servlet

- http://www.edu4java.com/es/servlet/servlet.html
- http://flanagan.ugr.es/docencia/2005-2006/2/servlets/
- https://www.tutorialspoint.com/servlets/index.htm
- http://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?codigo=137&punto=&inicio=60
- https://www.solvetic.com/tutoriales/article/544-paginas-jsp-como-enviar-parametros-metodo-get-y-post-en-servlets/
- http://www.galisteocantero.com/sesiones-en-servlets-y-jsp-ejemplo-login-logout/
- http://www.codejava.net/java-ee/servlet/handling-html-form-data-with-java-servlet
- http://www.edu4java.com/es/servlet/servlet3.html
- Falkner, J. y Jones, K. (2004). *Servlets a JavaServer Pages*. Boston: Pearson Education, Inc.

5.4.8 SQL

- https://es.wikipedia.org/wiki/SQL
- http://geotalleres.readthedocs.io/es/latest/conceptos-sql/conceptos_sql.html

- https://prezi.com/hfjcupyrqmus/uso-de-lenguaje-de-consulta-estructurado-sql/
- http://www.ite.educacion.es/formacion/materiales/93/cd/m6/index.html
- http://definicion.de/sql/
- http://searchdatacenter.techtarget.com/es/definicion/SQL-o-lenguaje-de-consultas-estructuradas
- https://www.w3schools.com/sql/default.asp
- https://es.wikipedia.org/wiki/Lenguaje_de_definici%C3%B3n_de_datos
- https://es.wikipedia.org/wiki/Lenguaje_de_manipulaci%C3%B3n_de_datos
- https://es.wikipedia.org/wiki/Lenguaje_de_control_de_datos
- http://enciclopedia.us.es/index.php/Lenguaje_declarativo
- http://chancrovsky.blogspot.com.es/2014/12/constraints-o-restricciones.html
- https://desdeceronetsql2.wordpress.com/2011/11/04/insertar-campo-bit-en-sql/
- https://www.campusmvp.es/recursos/post/Fundamentos-de-SQL-Consultas-SELECT-multi-tabla-Tipos-de-JOIN.aspx
- Oppel, A. y Sheldon, R. (2009). *Fundamentos de SQL*. México, D.F: The McGraw-Hill Companies.

5.5 Eclipse

- http://losteatinos.com/servlets/instalar-eclipse-e-incluir-tomcat-en-eclipse.html
- https://es.wikipedia.org/wiki/Eclipse_(software)
- https://www.genbetadev.com/herramientas/eclipse-ide
- http://www.uv.es/~jgutierr/MySQL_Java/TutorialEclipse.pdf
- https://es.slideshare.net/Benedeti/ide-eclipse-breve-gua-201399
- https://www.ecured.cu/Eclipse,_entorno_de_desarrollo_integrado
- https://www.ibm.com/developerworks/ssa/library/os-ecov/index.html
- https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado
- https://www.ecured.cu/IDE_de_Programaci%C3%B3n
- http://www.mauriciomatamala.net/ED/eclipse.php
- http://evergarzadam.blogspot.com.es/2016/06/caracteristicas-ventajas-y-desventajas.html
- http://soyprogramador.liz.mx/importancia-de-un-buen-ide/
- https://gabriel24rp.wordpress.com/2013/01/31/entorno-de-desarrollo-integrado-2/
- https://es.slideshare.net/MagaLasic/presentacion-eclipse-grupo-6
- https://prezi.com/8_lkuuyre_nm/conceptos-caracteristicas-ventajas-y-desventajas-de-los-i/?webgl=0
- https://www.if-not-true-then-false.com/2010/linux-install-eclipse-on-fedora-centos-red-hat-rhel/
- http://javarevisited.blogspot.com.es/2015/06/org.postgresql.Driver-javalangclassnotfoundexception.html

5.6 MagicDraw UML

- https://www.ecured.cu/MagicDraw
- https://es.wikipedia.org/wiki/MagicDraw_UML
- http://descargar.cnet.com/MagicDraw-UML/3000-18496_4-14701.html

- http://www.intercambiosvirtuales.org/software/magicdraw-uml-enterprise-v16-8-modelado-visual-uml-para-realizar-diagramas-de-procesos-de-negocio
- https://es.wikipedia.org/wiki/Herramienta_CASE
- https://www.youtube.com/watch?v=49whtRzXFzg
- https://www.youtube.com/watch?v=9tPfgYQLNd0

5.7 PostgreSQL – pgAdmin y psql

- https://platzi.com/blog/que-es-postgresql/
- https://es.wikipedia.org/wiki/PostgreSQL
- http://postgresql-dbms.blogspot.com.es/p/limitaciones-puntos-de-recuperacion.html
- http://www.guia-ubuntu.com/index.php?title=PgAdmin_III
- https://www.youtube.com/watch?v=B6Ck6oCjszA
- https://www.postgresql.org/docs/9.1/static/index.html
- https://www.digitalocean.com/community/tutorials/como-instalar-y-utilizar-postgresql-en-ubuntu-16-04-es

5.8 Apache Tomcat

- https://apachefoundation.wikispaces.com/Apache+Tomcat
- https://www.fdi.ucm.es/profesor/jpavon/web/42-tomcat.pdf
- http://www.ajpdsoft.com/modules.php?name=Encyclopedia&op=content&tid=769
- http://www.edu4java.com/es/servlet/servlet1.html
- https://es.wikipedia.org/wiki/Tomcat
- http://profesores.elo.utfsm.cl/~agv/elo330/2s03/projects/Tomcat/index.html
- https://www.youtube.com/watch?v=jd3T7Vaawzo
- http://www.ubuntufacil.com/2013/12/instalar-servidor-web-tomcat-en-ubuntu/

5.9 Otros

- https://www.linuxtotal.com.mx/index.php?cont=info_admon_003
- http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/aceptacion.htm
- https://geekytheory.com/como-crear-y-modificar-ficheros-con-java/
- https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/
- http://javacafesv.blogspot.com.es/2009/03/herencia-multiple-en-java-conveniente-o.html
- http://lab.dit.upm.es/~lprg/material/apuntes/doc/doc.htm
- https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=646:documentar-proyectos-java-con-comentarios-simbolos-tags-deprecated-param-etc-cu00680&catid=68&ltemid=188

- https://ociotec.com/como-documentar-el-codigo/
- http://mpison.webs.upv.es/investigacion_aplicada/textos/como_citar_upv.pdf

6 ANEXOS

6.1 Generación del archivo .WAR y despliegue en el Servidor

Para poder desplegar la aplicación en cualquier contenedor de aplicaciones se genera un archivo .WAR con la misma.

6.1.1 Generación del archivo con extensión .WAR

Para generar dicho fichero, desde Eclipse, tenemos que:

- Pinchar con el botón derecho en el directorio raíz del proyecto, en nuestro caso llamado *reserva_libros*.
- Seleccionar la opción *Export* y dentro de ésta elegir *WAR File*.

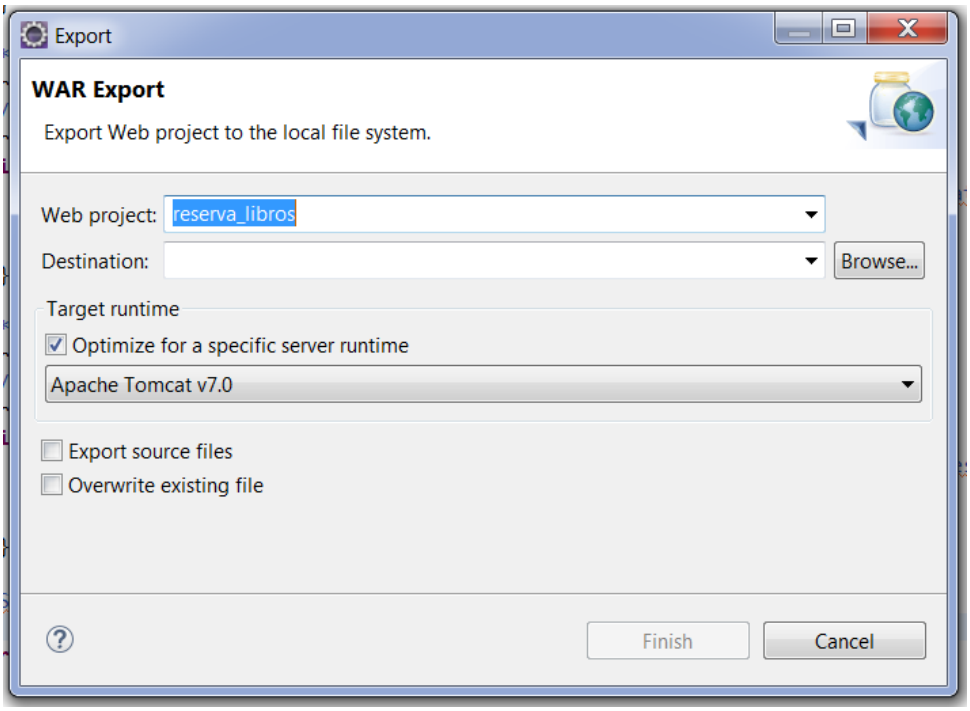

Figura 9 - Generación del fichero con extensión .WAR

- Tras esto, elegimos cómo se llamará el fichero .war, la carpeta en la que será guardo dicho fichero y por último *Finish*.
- Ya se encuentra generado dicho fichero en la carpeta destino seleccionada.

6.1.2 Despliegue del archivo con extensión .WAR

Para desplegar el fichero con extensión .war tenemos que:

- En primer lugar, necesitamos tener el servidor Apache Tomcat instalado.
- Procedemos a arrancar dicho servidor.
- Una vez arrancado el servidor, abrimos un navegador web y entramos en la configuración del servidor (en la dirección http://localhost:8080).

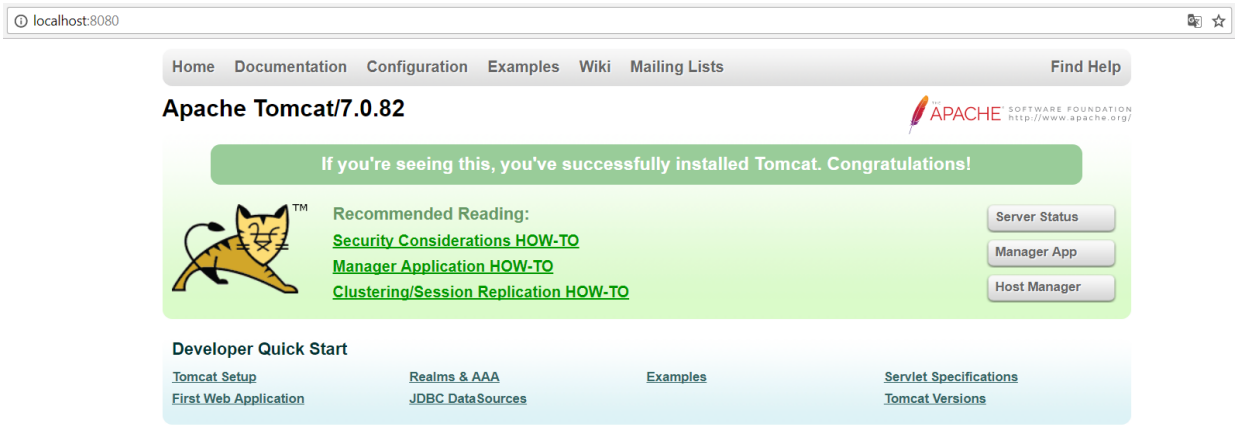

Figura 10 - Página dela configuración del servidor Apache Tomcat

- En la página web que nos aparece, elegimos la opción *Manage App*.
- Introducimos Usuario y Contraseña. En nuestro caso, estas credenciales son *admin* y *admin* respectivamente.
- En *Archivo .war a desplegar*, seleccionamos la ruta en la que se encuentra el archivo .war generado en el apartado anterior y pulsamos en el botón *Desplegar*.

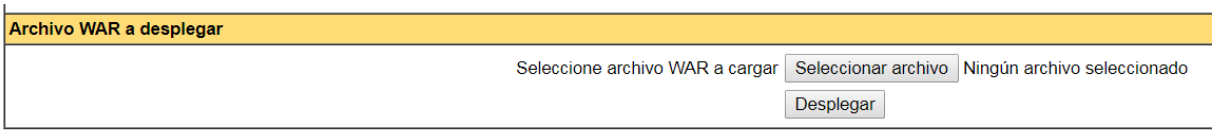

Figura 11 - Selección del archivo .WAR a descargar en el servidor

- En el apartado *Aplicaciones*, se puede visualizar que ya se encuentra añadida la aplicación, la ruta en la que ésta se encuentra y el estado (si no aparece como arrancada, necesitaríamos arrancarla).

Aplicaciones

Trayectoria	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado	Welcome to Tomcat	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/docs	Ninguno especificado	Tomcat Documentation	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/manager	Ninguno especificado	Tomcat Manager Application	true	1	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/reserva_libros	Ninguno especificado	reserva_libros	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos

Figura 12 - Aplicaciones cargadas en el servidor Apache Tomcat

- Tras esto, introducimos en la barra del navegador web la dirección en la que se encuentra nuestra aplicación. Ésta se encontrará lista para ejecutarla.

Figura 13 - Ejecución de la aplicación tras la configuración de ésta en el servidor

6.2 Arranque de servicio PostgreSQL

En este documento, se va a realizar una explicación de cómo poder manejar el servicio de la base de datos de nuestra aplicación. Se va a dividir la explicación para dos sistemas operativos: Windows 7 y Linux Ubuntu.

6.2.1 Windows

En Windows, se va a arrancar el servicio de nuestra base de datos PostgreSQL de dos formas. Por un lado, éste se arrancará a través de la interfaz gráfica de Windows y, por otro lado, a través de la línea de comandos.

A través de la interfaz gráfica:

Se accede al lugar habilitado para la gestión de los servicios a través de ***Equipo >Panel de Control>Sistema y Seguridad>Herramientas administrativas>Servicios***.

Aparecerá una pantalla desde la que se podrán controlar todos los servicios disponibles en Windows.

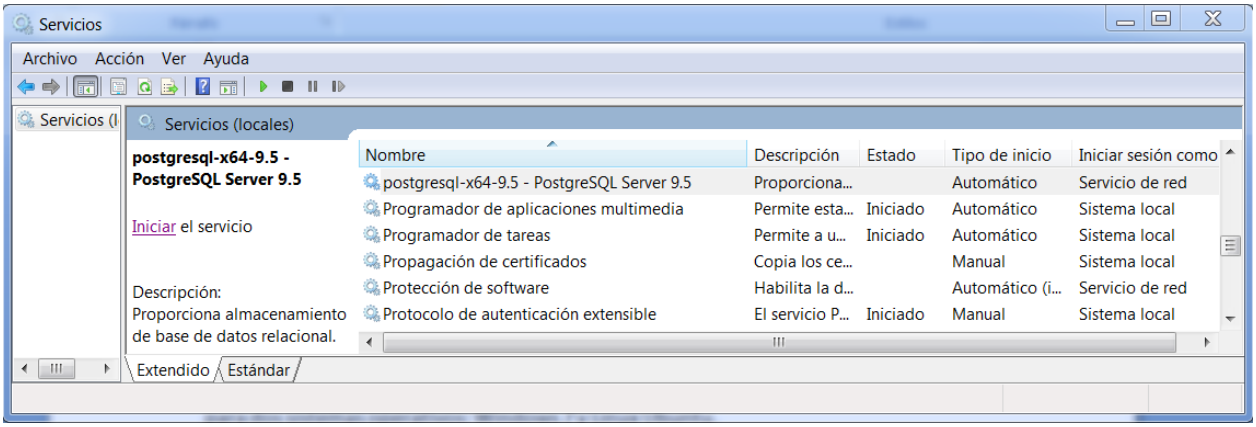

Figura 14– Gestor de los servicios de Windows

Se busca el servicio correspondiente, en este caso concreto se llama ***postresql-x64-9.5 – PostgreSQL Server 9.5*** y se inicia el servicio.

Tras esto, el servicio está iniciado y aparecerá la misma pantalla con las opciones de ***detener, pausar y reiniciar*** el servicio.

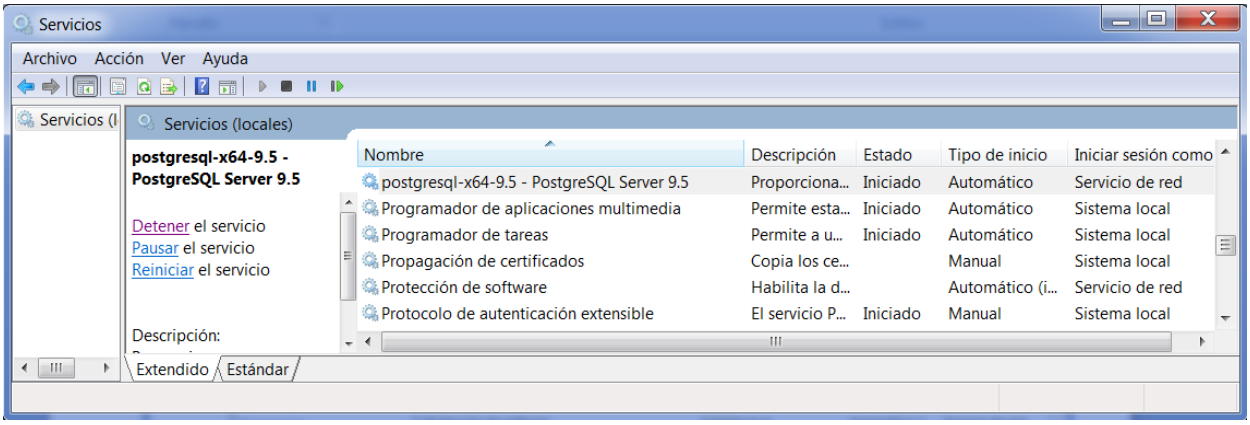

Figura 15 - Comprobación de que el servicio postrgresql-x64-9.5 - PostgreSQL Server 9.5 se encuentra en iniciado en Windows a través de la interfaz gráfica

A través de la línea de comandos:

Se abre el terminal a través de ***cmd***como ***Administrador***.

Se ejecuta el comando:

```
sc query | more
```

La ejecución de este comando se realiza para poder listar todos los servicios.

Tras esto, se busca el servicio que se desea arrancar (en este caso concreto el servicio se llama ***postgresql-x64-9.5***).

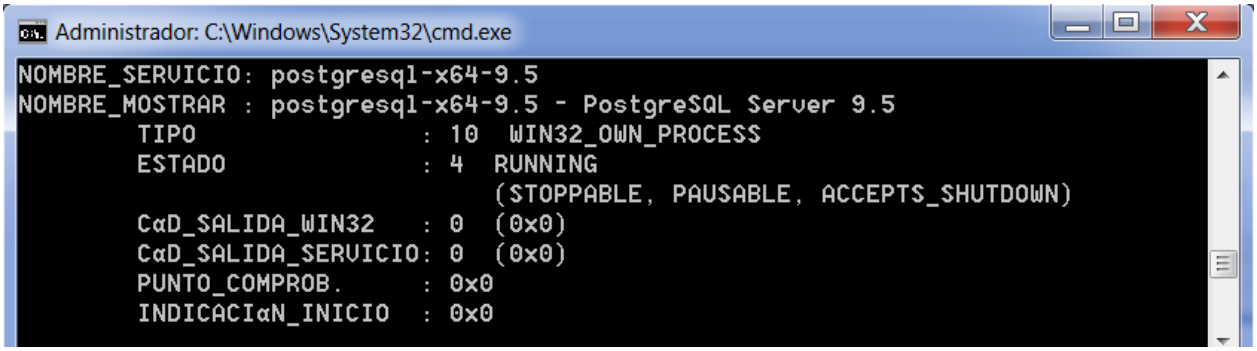

Figura 16 - Comprobación de que el servicio postrgresql-x64-9.5 - PostgreSQL Server 9.5 se encuentra en iniciado en Windows a través del terminal

Para arrancar el servicio, se ejecuta el comando:

```
sc start postgresql-x64-9.5
```

También se tienen las opciones para parar el servicio (**sc stop postgresql-x64-9.5**) y reiniciarlo (**sc restart postgresql-x64-9.5**). Además, hay otras opciones como **description** y **qdescription** para modificar la descripción del servicio y poder ver dicha descripción respectivamente.

6.2.2 Linux Ubuntu

En Linux Ubuntu, se procede a arrancar el servicio de la base de datos PostgreSQL a través de la línea de comandos. Además, antes de hacer estas indicaciones se explicará cómo instalar PostgreSQL a través de los repositorios de Ubuntu.

Para la instalación de PostgreSQL, se ejecuta:

```
sudo apt-get update
sudo apt-get install postgresql
```

La primera instrucción es recomendable ejecutarla. Sirve para actualizar la lista de paquetes disponibles en los repositorios locales.

Con la segunda instrucción, se instala cliente y servidor de la base de datos y algunas extensiones útiles (instaladas con el paquete ***postgresql-contrib***).

Para comprobar que la instalación se ha realizado de forma correcta se puede ver la versión de PostgreSQL instalada en nuestro equipo. Esto se puede hacer ejecutando:

```
psql --version
```

Como resultado se obtiene:

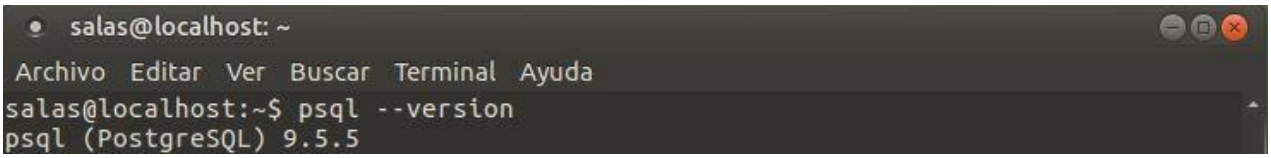

Figura 17 - Comprobación de que PostgreSQL se encuentra instalado en el sistema y versión de éste

El servicio de PostgreSQL puede ser arrancado de dos formas (con ejecutar una de las dos se arranca el servicio):

```
/etc/init.d/postgresql start
service postgresql start
```

Si se ejecuta la primera instrucción, se arranca el servicio desde la ruta en la que se encuentra ubicado el script de arranque de dicho servicio. Dentro de la carpeta ubicada en ***/etc***(en Ubuntu) están ubicados, entre otros,los scripts para iniciarla mayor parte de los servicios instalados en el sistema.

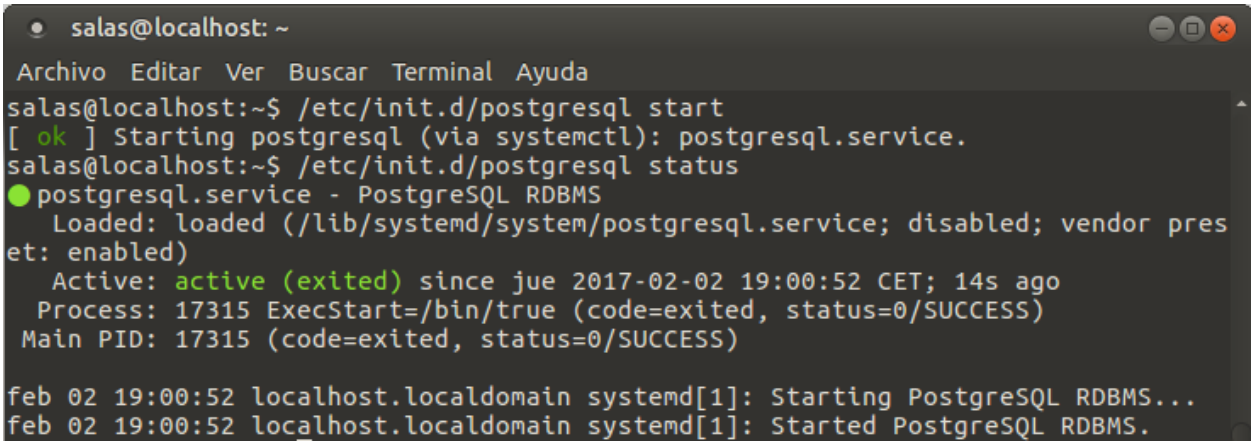

Figura 18–Arranque del servicio de PostgreSQL en Linux Ubuntu (Primera forma)

Ha sido usado el comando ***/etc/init.d/postgresql status***para verificar que el servicio está arrancado.

Si se decide ejecutar la segunda instrucción, se arranca el servicio a través del comando ***service***, el cual tiene entre sus funciones la de iniciar servicios.

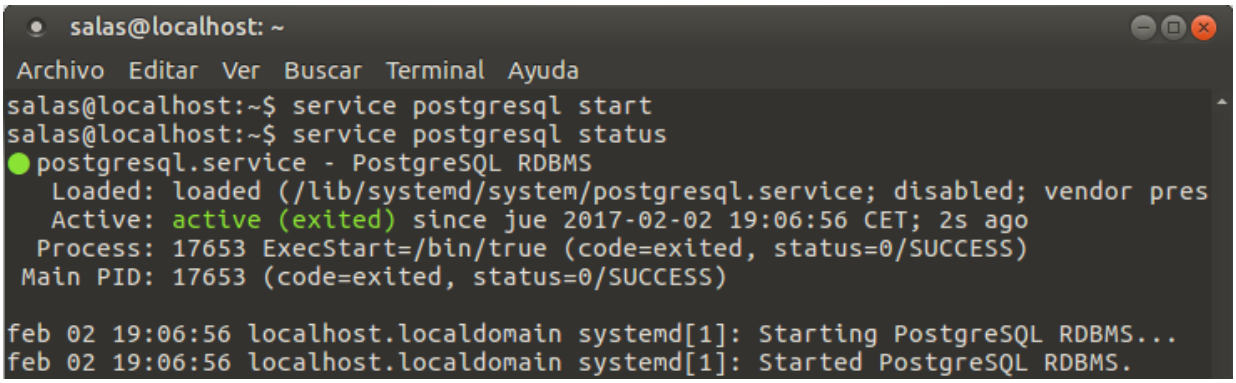

Figura 19 - Arranque del servicio de PostgreSQL en Linux Ubuntu (Segunda forma)

Además de arrancar el servicio, cualquiera de las dos formas de manejo de los servicios permite ***detener***, ***reiniciar*** y ***ver el estado de los servicios*** mediante ***stop***, ***restart*** y ***status*** respectivamente.

6.3 Tutorial básico de manejo de PostgreSQL

En este apartado se verán una serie de comandos e instrucciones (a través de la consola de Ubuntu y de la consola de PostgreSQL) para el mantenimiento y gestión de nuestra base de datos. PostgreSQL tiene muchas más funcionalidades (se invita al que quiera a investigar y avanzar más sobre este tema), pero se ha decidido seleccionar en este tutorial las que se consideran más adecuadas para tener una primera toma de contacto con esta aplicación.

Abrir consola de PostgreSQL:

En su instalación, PostgreSQL crea un usuario (sin contraseña) por defecto. Para abrir la consola de nuestra base de datos con ese usuario por defecto se ejecuta:

```
sudo –u postgres psql
```

En la siguiente captura de la consola de Ubuntu se puede ver que ya estamos dentro de nuestra consola de PostgreSQL a través de la cuenta del usuario ***postgres***(usuario creado por defecto).

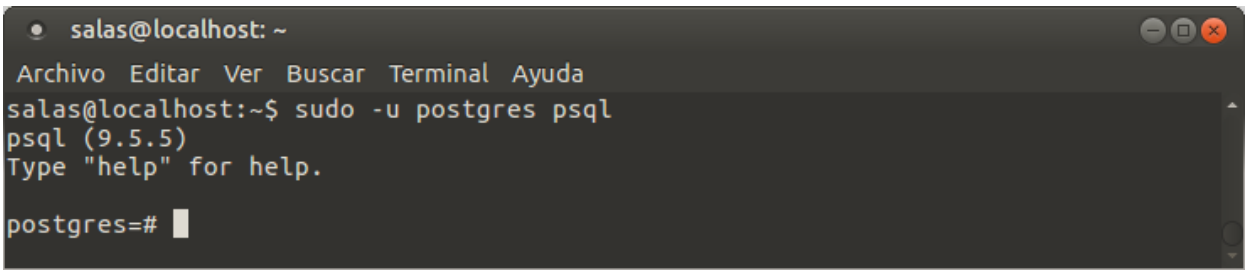

Figura 20 - Entrada en la consola de PostgreSQL con el usuario por defecto

Salir de la consola de PostgreSQL:

Para salir de la consola de PostgreSQL se ejecuta la instrucción:

```
\q
```

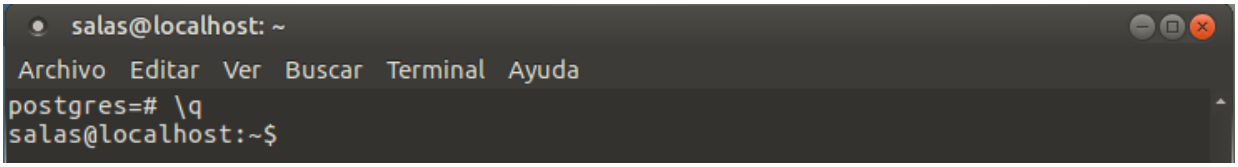

Figura 21 - Salir de la consola de PostgreSQL

Creación de roles/usuarios:

Como se ha comentado anteriormente, se encuentra disponible el rol/usuario de ***postgres*** creado por defecto en la instalación de la aplicación. Además, en la instalación de la aplicación también se ha creado por defecto el usuario ***dit*** que es un usuario del sistema. Este usuario dispone del privilegio de poder crear bases de datos.

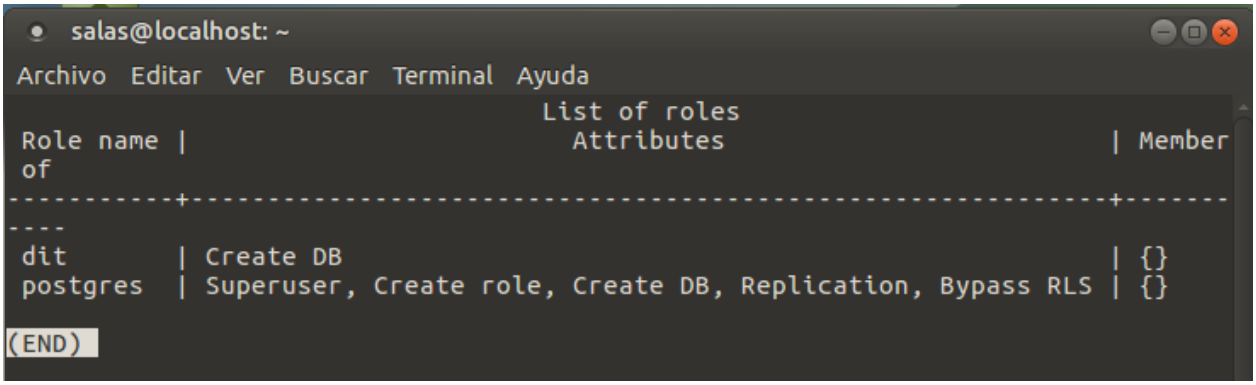

Figura 22 - Lista de usuarios de la Base de Datos I

Se pueden crear nuevos roles/usuarios. En este tutorial, se explicará cómo crearlos de tres formas diferentes (dos desde el terminal de Ubuntu y una desde la consola de administración de nuestra base de datos).

La **primera forma**de crear nuevos usuarios/roles que será explicada se ejecuta desde el terminal de Ubuntu y se realiza ejecutando el siguiente comando:

```
sudo –u postgres createuser --interactive
```

donde ***postgres*** es un rol con el privilegio de poder crear nuevos roles/usuarios.

Al ejecutar dicho comando, el script solicita la introducción del nombre del nuevo rol/usuario. Además, van apareciendo distintas opciones para dotar a dicho usuario de los privilegios que se consideren convenientes. Se pueden aceptar cada una de estas opciones mediante el carácter ***y(yes)*** o rechazarla mediante el carácter ***n (no)***.

La primera opción sobre la que el script nos da a elegir consiste en decidir si el nuevo usuario será ***superusuario*** dentro de nuestra base de datos. Como el usuario que es superusuario posee todos los privilegios dentro de la aplicación, si se decide que sí no se nos plantearán más opciones.

Si se decide que el nuevo usuario no sea superusuario, la siguiente opción que da el script es que el usuario tenga permitido ***crear bases de datos***.

La última opción que da el script para configurar el nuevo rol es la de permitirle crear ***nuevos***

usuarios/roles.

Como ejemplo de esta forma de creación de nuevos usuarios/roles, ha sido creado un nuevo usuario/rol llamado ***ejemplo1*** al que se le ha dado el privilegio de poder crear bases de datos.

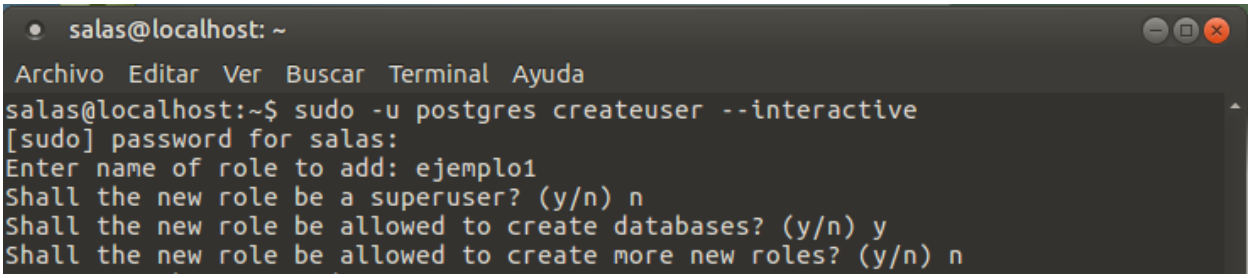

Figura 23 - Creacion de un usuario de PosgreSQL (Forma 1)

Para verificar que se ha creado bien, se entra en la consola de PostgreSQL con cualquier usuario (en este ejemplo, el usuario usado será ***postgres***, el cual teníamos creado anteriormente) y, dentro de ésta, se listan todos los usuarios. En dicha lista, aparecen también los privilegios que tiene cada usuario dentro de la aplicación. Esto se realiza ejecutando:

sudo –u postgres psql

\du (este comando se ejecuta dentro de la consola de PostgreSQL)

En la siguiente captura de pantalla se ve, como se comentó anteriormente, que está creado el usuario ***ejemplo1*** con el privilegio de poder crear bases de datos:

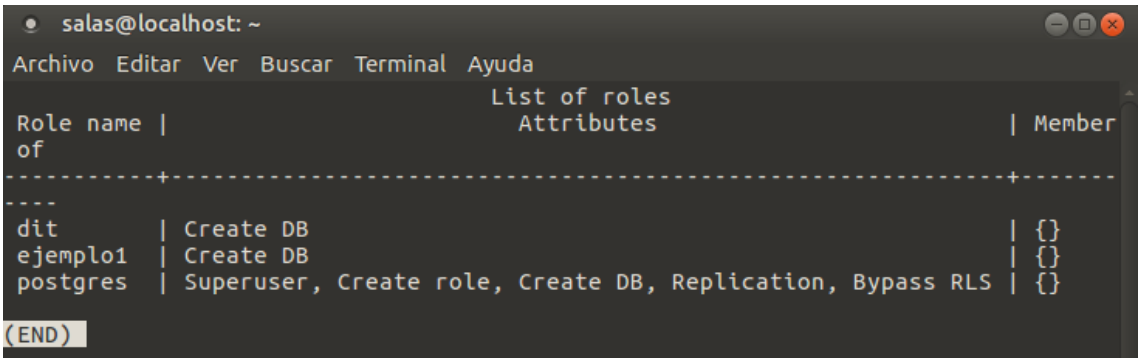

Figura 24 - Lista de usuarios de la Base de Datos II

La **segunda forma** de crear nuevos usuarios/roles será desde el terminal de Ubuntu y se realiza ejecutando el siguiente comando:

sudo –u postgres createuser nuevo_rol

donde ***postgres*** es un rol con el privilegio de poder crear nuevos roles/usuarios y ***nuevo_rol*** el usuario/rol a crear

Tras esto, ha sido creado un nuevo usuario/rol sin ningún tipo de privilegio. Posteriormente, se explicará cómo dar/quitar privilegios a un usuario ya creado.

Se verifica que ***nuevo_rol*** se ha creado de forma correcta (y que no tiene ningún tipo de privilegio)

de la misma forma que lo realizamos anteriormente:

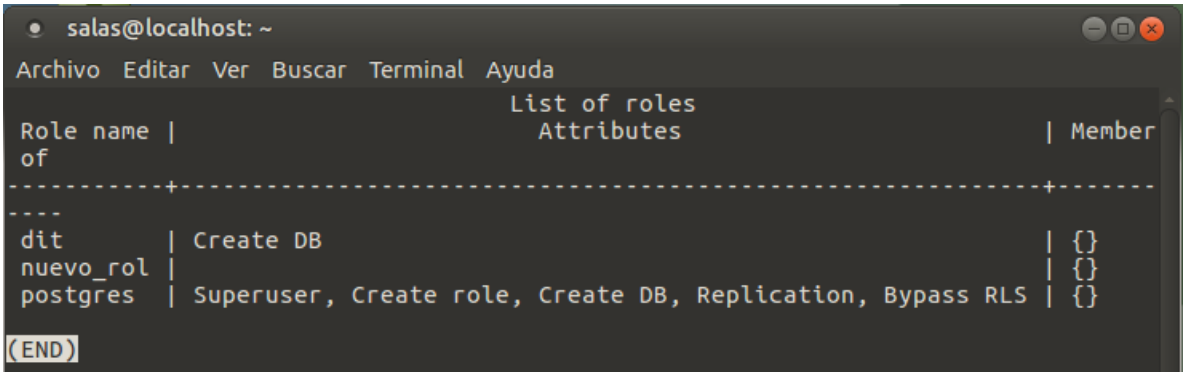

Figura 25 - Lista de usuarios de la Base de Datos III

La **tercera forma** de crear nuevos usuarios/roles será desde la consola de PostgreSQL y se realiza ejecutando el siguiente comando:

```
CREATE USER otro_rol;
```

donde ***otro_rol*** es el nuevo rol/usuario

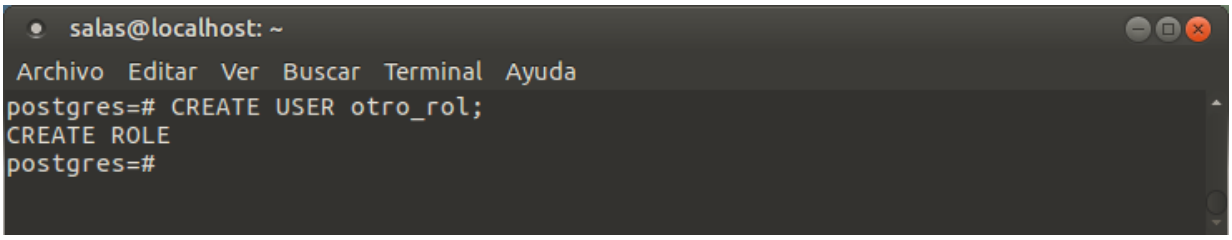

Figura 26 - Creacion de un usuario de PosgreSQL (Forma 3)

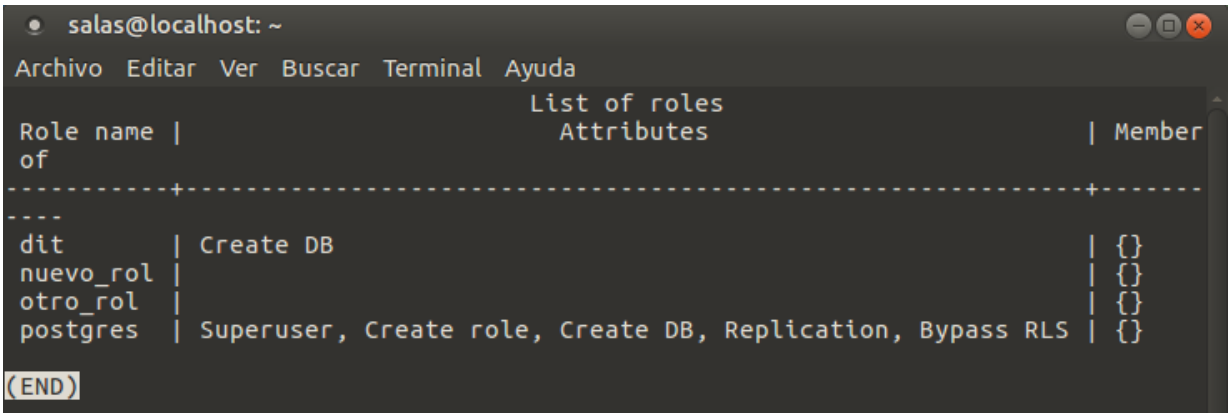

Figura 27 - Lista de usuarios de la Base de Datos IV

Como se puede ver, creando el usuario/rol de esta forma, tampoco se le asigna ningún privilegio.

Si se desea crear un nuevo rol/usuario asignándole una contraseña sólo se tendría que modificar la anterior instrucción añadiéndole ***PASSWORD 'nueva_pass'***al final.

Eliminar roles/usuarios:

En este tutorial se van a ver dos formas de ***eliminar usuarios/roles*** (una desde el terminal de Ubuntu y otra desde la consola de administración de nuestra base de datos).

La **primera forma** de eliminar usuarios/roles que se explicará desde el terminal de Ubuntu y se realiza ejecutando el siguiente comando:

```
sudo –u postgres dropuser otro_rol
```

donde ***postgres*** es un rol con el privilegio de poder eliminar roles/usuarios y ***otro_rol*** un usuario/rol existente en el sistema (a modo de recordatorio, fue creado a modo de ejemplo en este mismo tutorial)

En la siguiente captura de pantalla se puede ver cómo el usuario ***otro_rol*** ha sido eliminado ya que no aparece en el listado de usuarios/roles:

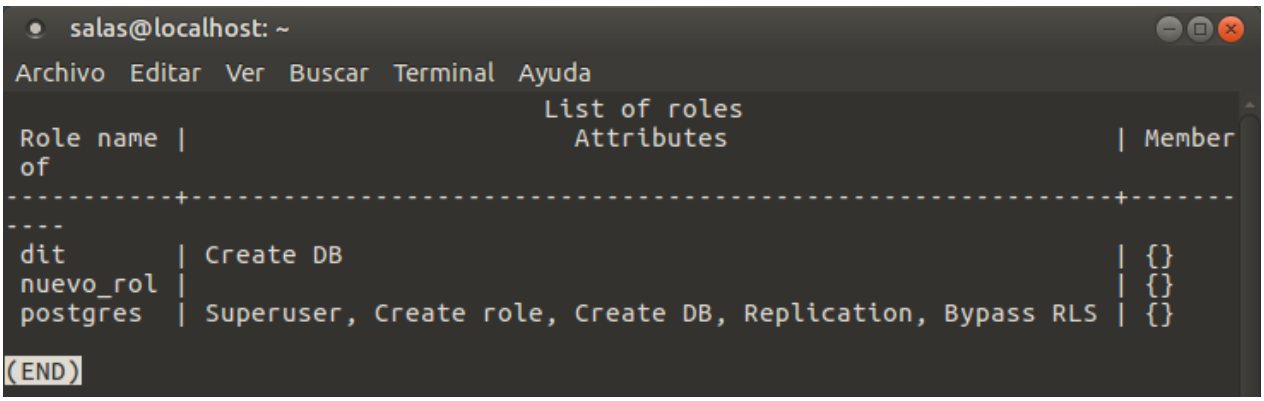

Figura 28 - Lista de usuarios de la Base de Datos V

La **segunda forma** de eliminar nuevos usuarios/roles será desde la consola de PostgreSQL y se realiza ejecutando el siguiente comando:

```
DROP USER nuevo_rol;
```

donde ***nuevo_rol*** es el usuario/rol a eliminar

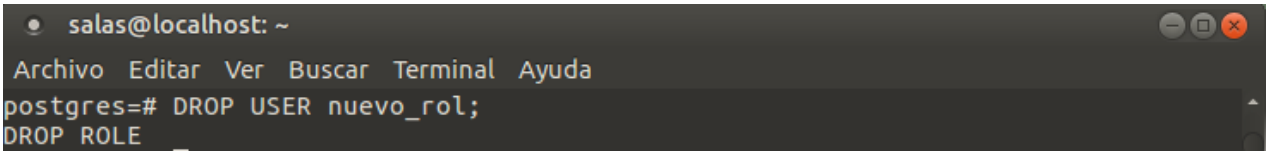

Figura 29 - Eliminación de un usuario de PostgreSQL

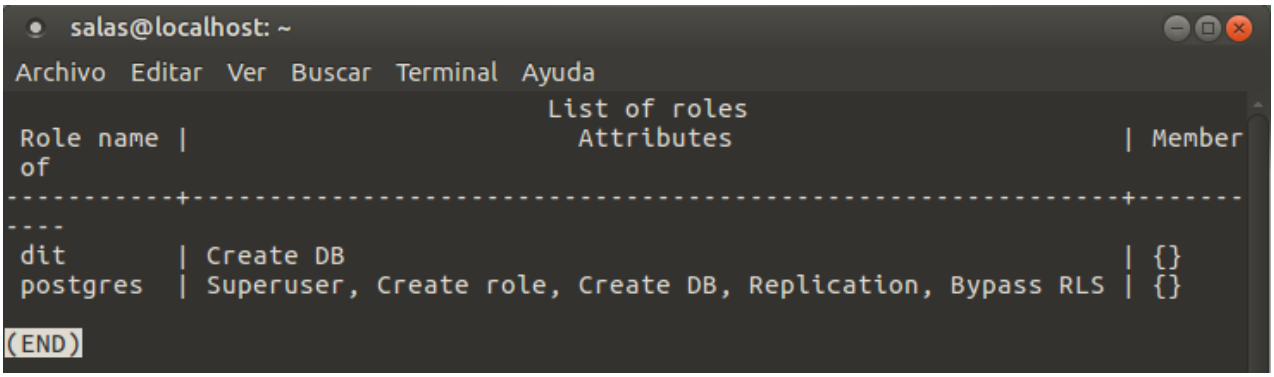

Figura 30 - Lista de usuarios de la Base de Datos VI

Se puede ver que el usuario ***nuevo_rol*** se ha eliminado de forma correcta.

Dotar a un rol/usuario con el privilegio de superusuario:

Se crea un usuario sin privilegios (se ha visto anteriormente cómo crear roles/usuarios). Por ejemplo, se va a llamar ***usuario1***.

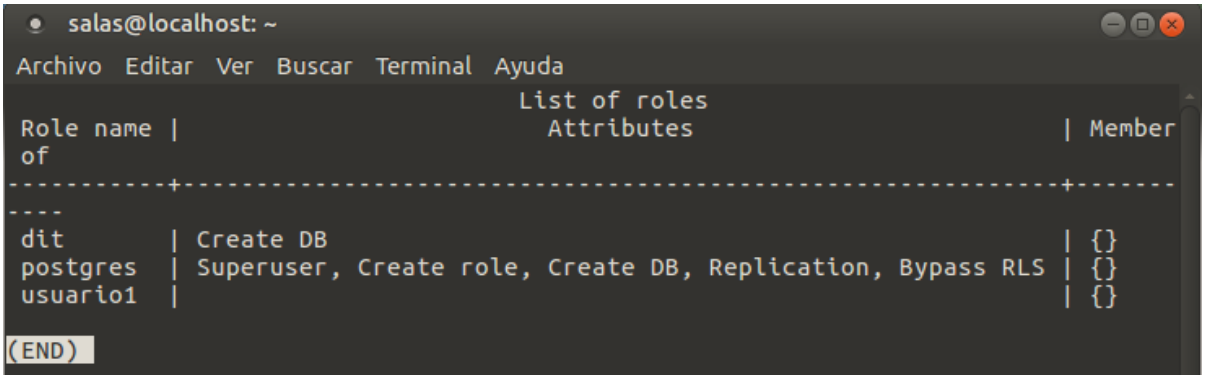

Figura 31 - Lista de usuarios de la Base de Datos VII

Ejecutando el siguiente comando se va a dotar al usuario ***usuario1*** del privilegio de ser ***superusuario***, es decir, tras esto, el usuario ***podrá crear bases de de datos*** y ***nuevos usuarios.***

```
ALTER ROLE usuario1 WITH SUPERUSER;
```

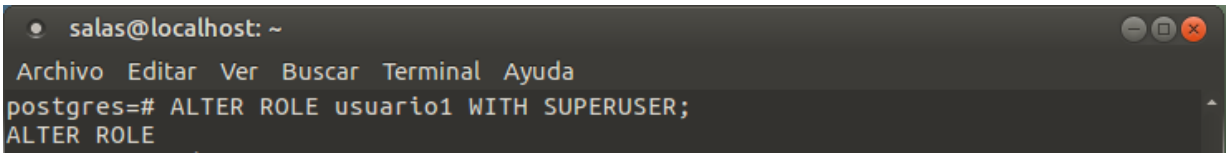

Figura 32 - Dotar a un usuario con el privilegido de superusuario

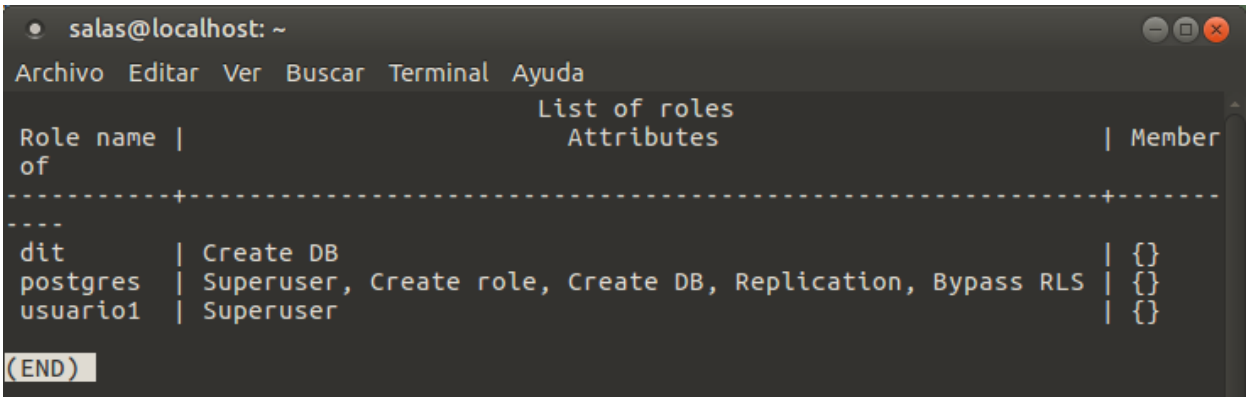

Figura 33 - Lista de usuarios de la Base de Datos VIII

Se puede realizar la operación inversa (hacer que un usuario que sea ***superusuario deje de serlo***) ejecutando el siguiente comando:

```
ALTER ROLE usuario1 WITH NOSUPERUSER;
```

Dotar a un rol/usuario con el privilegio de poder crear bases de datos:

Se crea un usuario sin privilegios (se ha visto anteriormente cómo crear roles/usuarios). Por ejemplo, se va a llamar ***usuario2***.

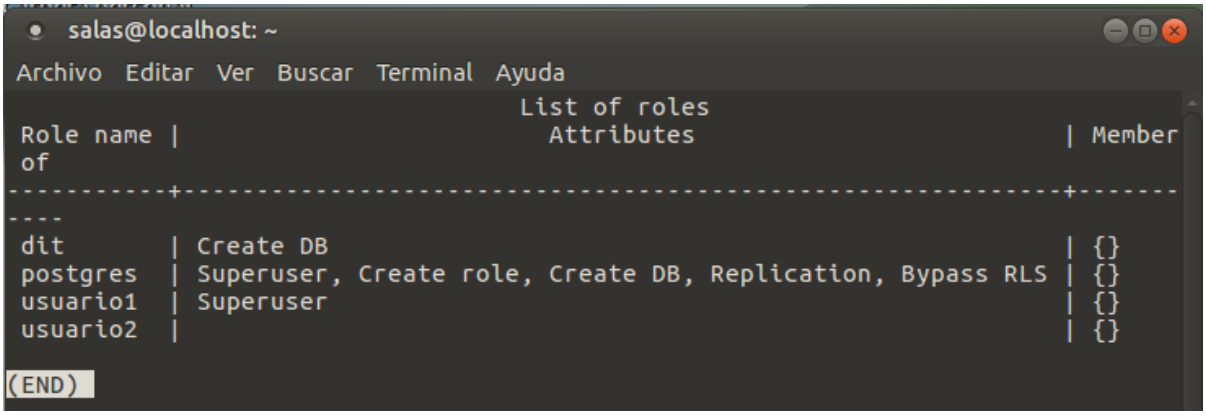

Figura 34 - Lista de usuarios de la Base de Datos IX

Ejecutando el siguiente comando se va a dotar al usuario ***usuario2*** del privilegio de ***poder crear bases de datos***.

```
ALTER ROLE usuario2 WITH CREATEDB;
```

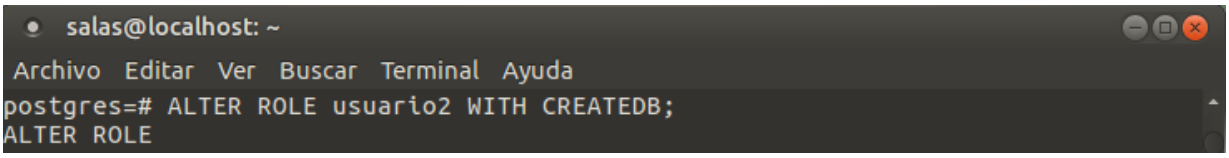

Figura 35 - Dotar a un usuario con el privilegido de poder crear Bases de Datos

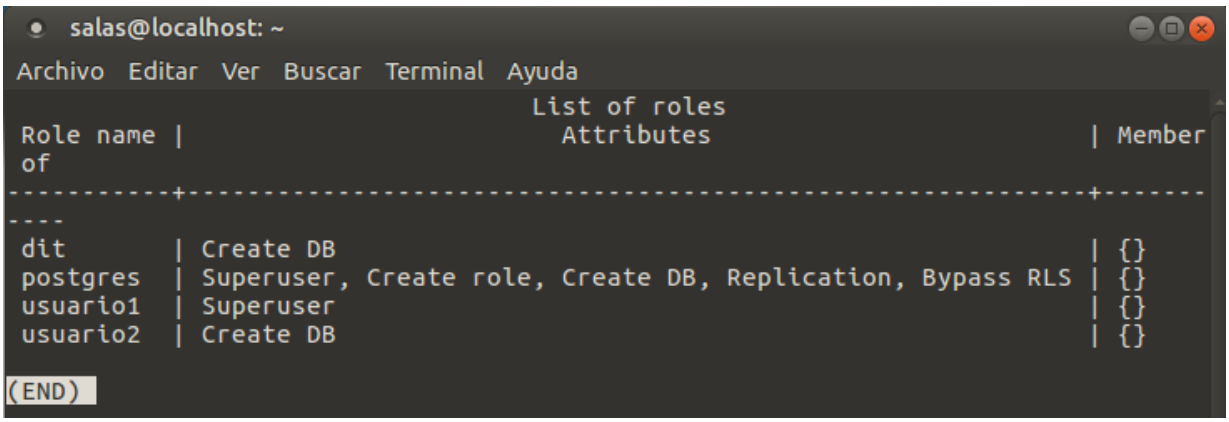

Figura 36 - Lista de usuarios de la Base de Datos X

Se puede realizar la operación inversa (hacer que el usuario ***no pueda crear bases de datos***) ejecutando el siguiente comando:

```
ALTER ROLE usuario2 WITH NOCREATEDB;
```

Dotar a un rol/usuario con el privilegio de poder crear nuevos usuarios/roles:

Se crea un usuario sin privilegios (se ha visto anteriormente cómo crear roles/usuarios). Por ejemplo, se va a llamar ***usuario3***.

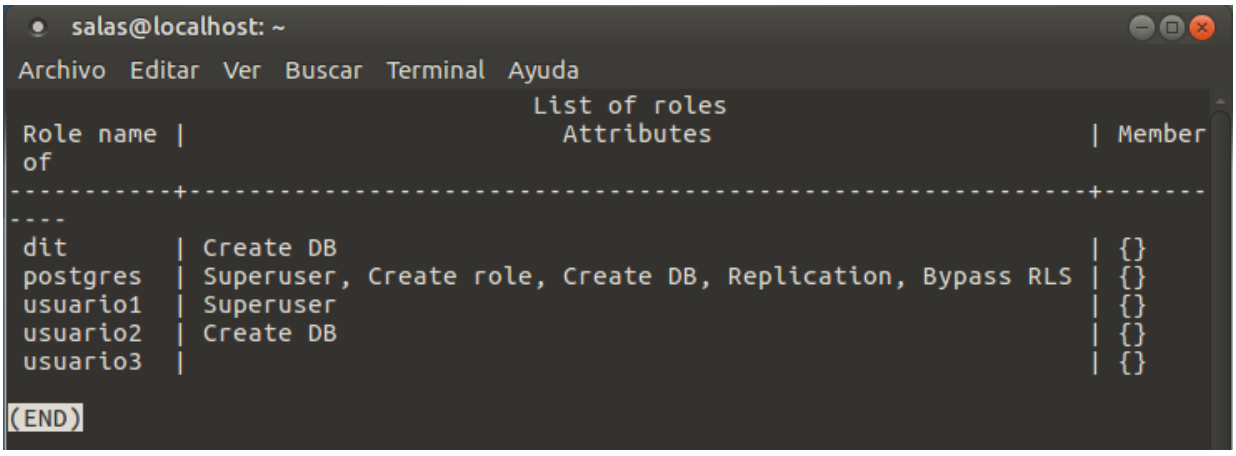

Figura 37 - Lista de usuarios de la Base de Datos XI

Ejecutando el siguiente comando se va a dotar al usuario ***usuario3*** del privilegio de poder crear nuevos roles/usuarios.

```
ALTER ROLE usuario3 WITH CREATEROLE;
```

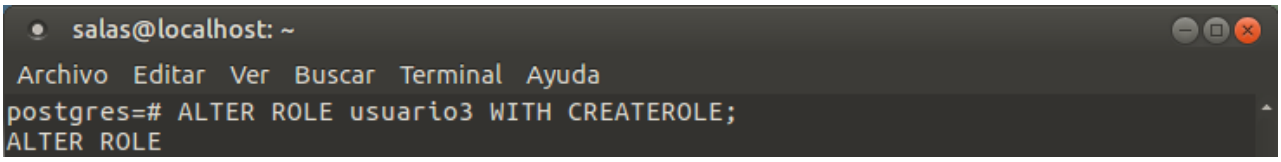

Figura 38 - Dotar a un usuario con el privilegido de poder crear nuevos usuarios/roles

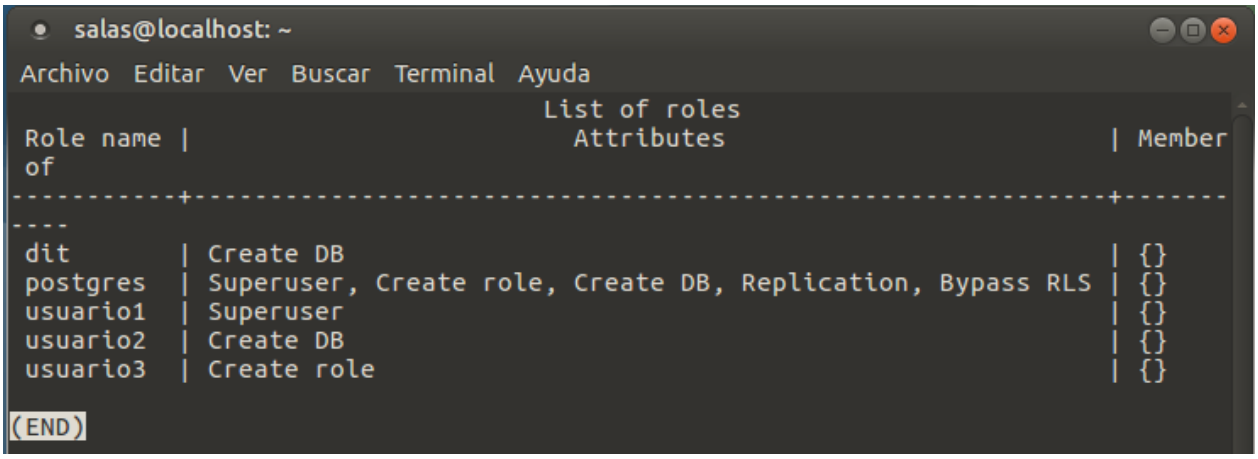

Figura 39 - Lista de usuarios de la Base de Datos XII

Se puede realizar la operación inversa (hacer que el ***usuario no pueda crear nuevos usuarios/roles***) ejecutando el siguiente comando:

```
ALTER ROLE usuario3 WITH NOCREATEROLE;
```

Cambiar la contraseña de un rol/usuario:

Hasta ahora se han creado los roles/usuarios sin contraseña, pero para más seguridad sería conveniente añadirle una. El comando a ejecutar para ***cambiar la contraseña***, o ***añadirla*** en el caso de que ésta no exista es:

```
ALTER ROLE nombre_usuario PASSWORD 'nueva_contrasena';
```

donde ***nombre_usuario*** es el nombre del rol/usuario en cuestión y ***nueva_contrasena*** la nueva contraseña.

Eliminar la contraseña de un rol/usuario:

Para ***eliminar la contraseña*** de un rol/usuario se ejecuta:

```
ALTER ROLE nombre_usuario PASSWORD NULL;
```

Crear una base de datos:

La ***creación de la base de datos*** la puede crear cualquier ***usuario que tenga permiso para crearla*** (los superusuarios también disponen de dicho privilegio).

En este tutorial, van a ser explicadas dos formas de crear bases de datos (una a través de la consola de Ubuntu y la otra a través de la consola de PostgreSQL).

En nuestro ejemplo, las bases de datos serán creadas con el usuario ***postgres***.

La **primera forma** a explicar será la creación de la base de datos desde el teminal de Ubuntu. Para ello, se ejecuta el siguiente comando:

```
sudo –u postgres createdb ejemplobd
```

donde ***postgres*** es el usuario encargado de crear la base de datos y ***ejemplobd*** el nombre de la base de datos a crear.

Se comprueba que ha sido creada la base de datos ***ejemplobd*** correctamente **listando todas las bases de datos existentes** mediante el comando:

```
\l (desde la consola de PostgreSQL)
```

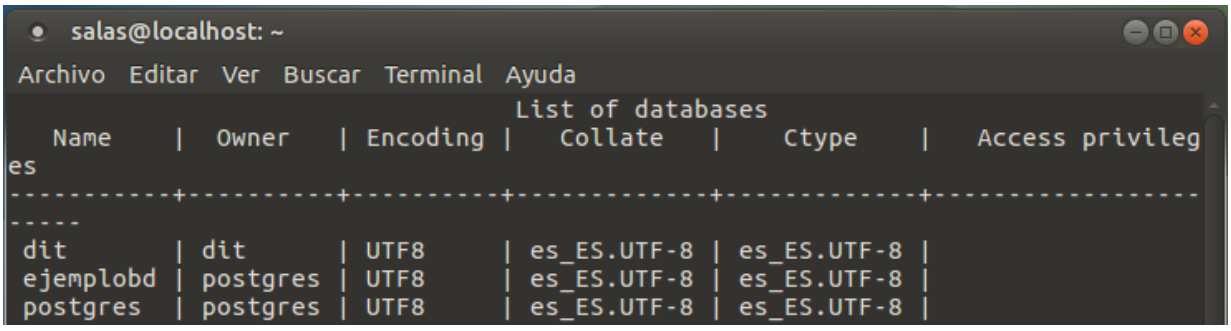

Figura 40 - Lista de todas las Bases de Datos existentes I

La **segunda forma** a explicar será la creación de la base de datos desde la consola de PostgreSQL. Para ello, se accede con el usuario ***postgres*** (como se ha explicado anteriormente) y se ejecuta el siguiente comando:

```
CREATE DATABASE otradb;
```

donde ***otradb*** es la base de datos a crear.

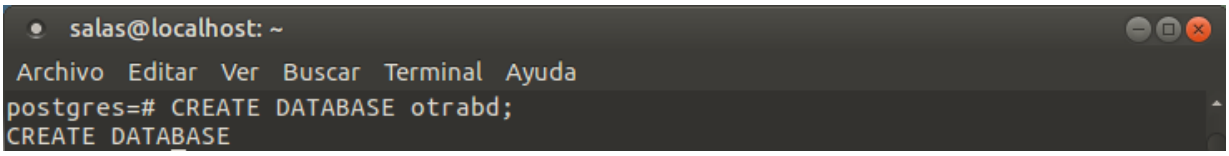

Figura 41 - Creación de una Base de Datos desde la consola de PostgreSQL

Posteriormente, se verifica que la base de datos ***otrabd*** se ha creado correctamente:

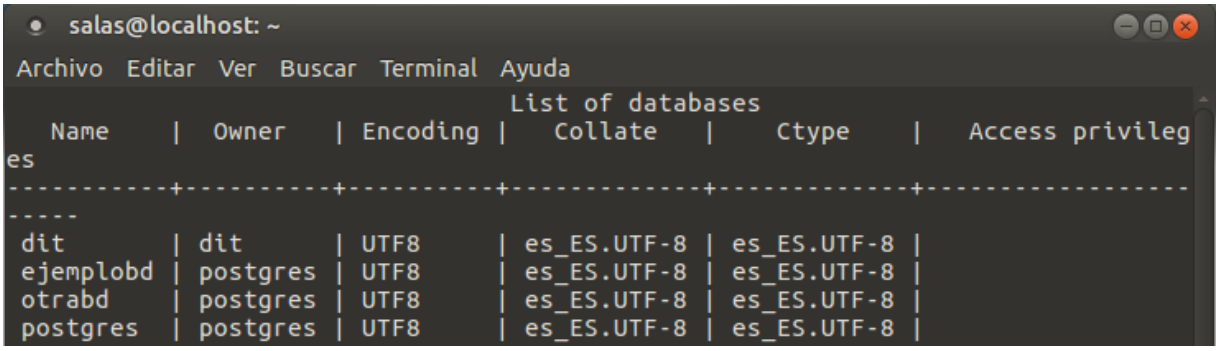

Figura 42 - Lista de todas las Bases de Datos existentes II

Las bases de datos que se han creado con un usuario (en nuestro ejemplo el usuario ***postgres***) tienen como propietario a dicho usuario. Se puede crear una base de datos especificándole un ***usuario propietario***. Esto se puede hacer ejecutando el siguiente comando:

```
CREATE DATABASE nuevabd WITH OWNER usuario2;
```

donde ***nuevabd*** es labase de datos a crear y ***usuario2*** el usuario propietario de dicha base de datos.

Borrar una base de datos:

Al igual que para crear una base de datos, van a ser explicadas dos formas para poder borrarla (una a través de la consola de Ubuntu y la otra a través de la consola de PostgreSQL). Para poder ***eliminar una base de datos***, ésta tiene que estar ***vacía***, es decir, que no tenga ninguna tabla creada en su interior.

Como ser realizó anteriormente, se van a eliminar las bases de datos de nuestro ejemplo con el usuario ***postgres***.

La **primera forma** que se va a explicar será la eliminación de la base de datos desde el teminal de Ubuntu. Para ello, se ejecuta el siguiente comando:

```
sudo –u postgres dropdb ejemplodb
```

donde ***postgres*** es el usuario encargado de borrar la base de datos y ***ejemplobd*** el nombre de la base de datos a borrar.

Se comprueba que la base de datos ***ejemplobd*** ha sido eliminada correctamente:

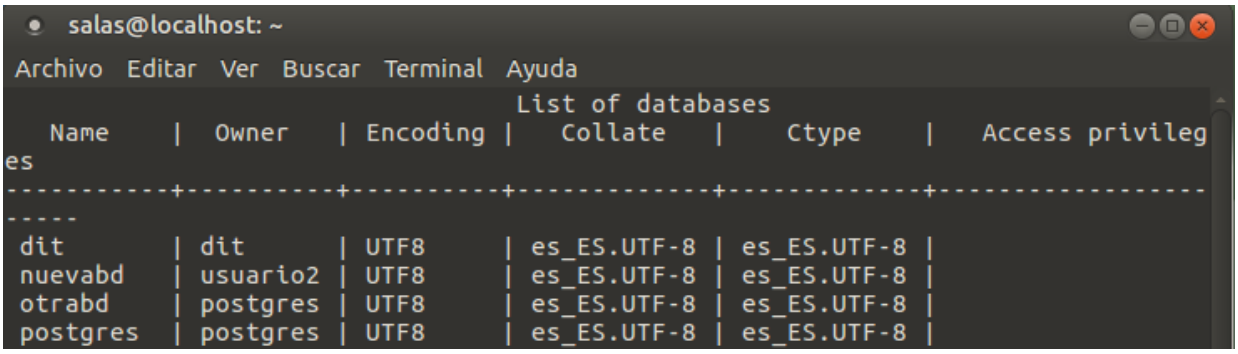

Figura 43 - Lista de todas las Bases de Datos existentes III

La **segunda forma** de eliminar bases de datos se ejecutará desde la consola de PostgreSQL. Para ello, se accede con el usuario ***postgres*** (como se ha explicado anteriormente) y se ejecuta el siguiente comando:

```
DROP DATABASE otrabd;
```

donde ***otrabd*** es la base de datos a eliminar.

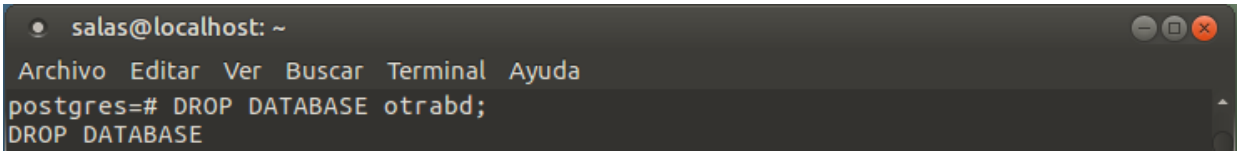

Figura 44 - Borrar una Base de Datos a través de la consola de PostgreSQL

Posteriormente, se verifica que la base de datos ***otrabd*** se ha eliminado correctamente:

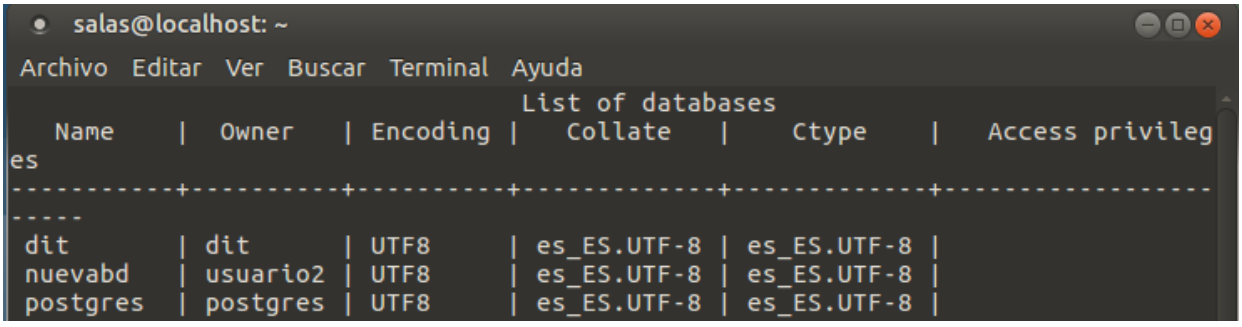

Figura 45 - Lista de todas las Bases de Datos existentes IV

Recordatorio: Para poder borrar una base de datos, ésta tiene que estar vacía, es decir, no puede contener ninguna tabla en su interior.

Comprobar información de conexión del usuario que tiene abierta la consola de PostgreSQL:

Para comprobar la información de conexión actual del usuario que tiene abierta la consola de PostgreSQL será usado el siguiente comando:

```
\conninfo (desde la consola de PostgreSQL)
```

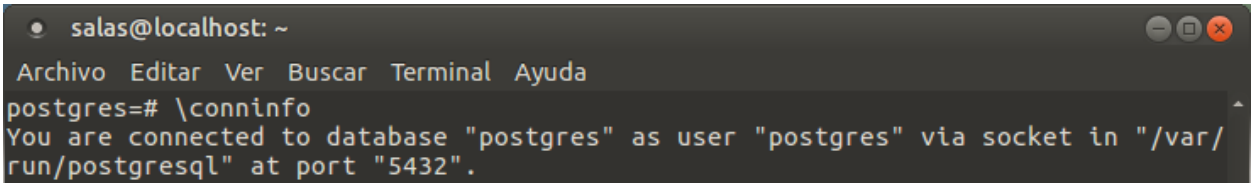

Figura 46 - Información de conexión del usuario que tiene abierta la consola de PostgreSQL

Esto indica el usuario y la base de datos a la que el usuario se encuentra conectado. En el ejemplo, el usuario activo es el usuario ***postgres*** y se está conectando a la base de datos que se crea por defecto (llamada ***postgres***).

Crear una tabla:

Ya se ha visto, entre otras cosas, cómo gestionar usuarios, bases de datos y cómo conectarse a PostgreSQL. A partir de ahora, la explicación se va a centrar en el manejo de las tablas, el "interior" de las bases de datos.

Para realizar esto, se trabajará sólo desde la consola de PostgreSQL. En el ejemplo, se hará con el usuario ***postgres*** conectado a la base de datos ***postgres***.

El objetivo de crear una tabla es el de poder almacenar datos en ella. Para crearla se usa la siguiente sintaxis:

```
CREATE TABLE nombre_tabla (
   nombre_columna_1 tipo_columna (tamaño_campo) restricciones_columna,
   nombre_columna_2 tipo_columna (tamaño_campo),
   nombre_columna_3 tipo_columna (tamaño_campo),
nombre_columna_4 tipo_columna (tamaño_campo)
);
```

En el ejemplo, se va a crear una tabla de 5 columnas. Ésta simulará a una tabla que se use en un concesionario para guardar todos los coches que tengan en catálogo debido a considerarse un ejemplo muy atractivo, claro e intuitivo. La primera, la tercera y la quinta columnas serán de tipo entero y las otras dos columnas de tipo carácter. La primera columna será clave primaria (PRIMARY KEY). Esto significa que todos los valores que haya en esta columna serán únicos y no nulos.

Como se comentó anteriormente, se ha usado esta temática en el ejemplo debido a su fácil comprensión y a que todo aquel que lo lea tiene conocimiento de dicho tema. Otro de los motivos influyentes en la elección de la temática ha sido la pasión del autor de este tutorial por el mundo del motor.

La creación de la tabla ***coches*** es la siguiente:

```
CREATE TABLE coches (
   id_coche integer PRIMARY KEY,
modelovarchar (40),
   potencia integer,
   colorvarchar (25),
   precio integer
);
```

Se comprueba que se ha creado la tabla con el siguiente comando:

```
\dt
```

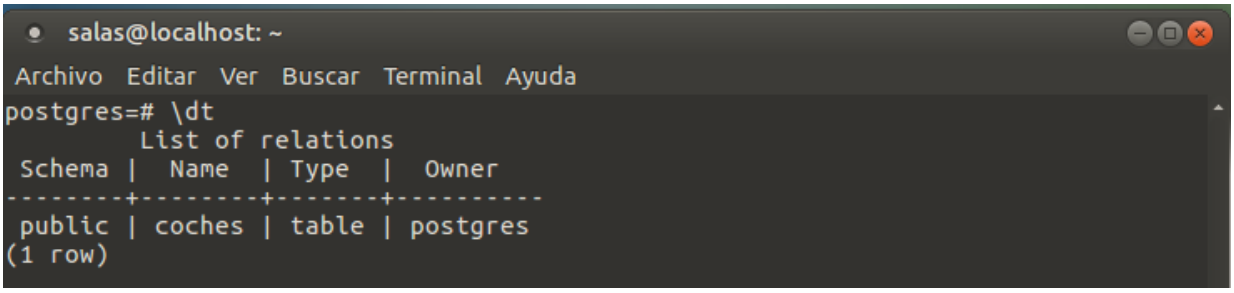

Figura 47 - Lista de las tablas que componen una Base de Datos I

Borrar una tabla:

Para eliminar una tabla se ejecuta el siguiente comando:

```
DROP TABLE nombre_tabla;
```

donde ***nombre_tabla*** es el nombre de la tabla a eliminar.

En el ejemplo, se va a eliminar la tabla que ha sido creada anteriormente:

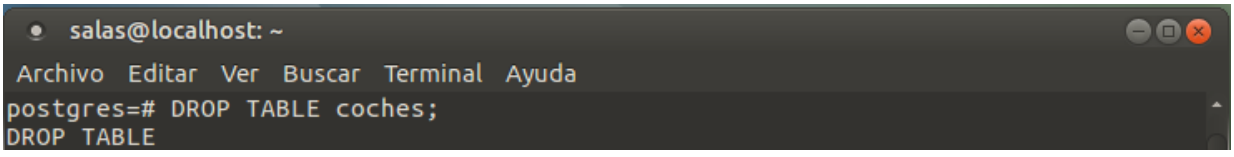

Figura 48 - Eliminación de una tabla desde la consola de PostgreSQL

Se verifica que dicha tabla ha sido borrada correctamente:

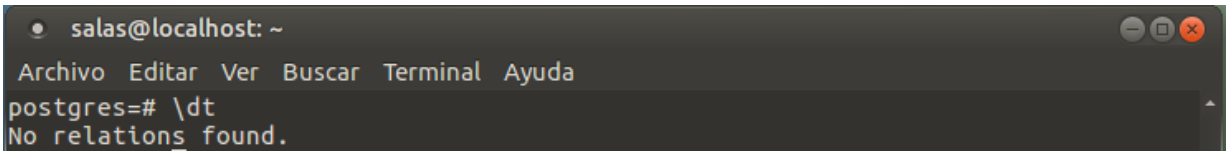

Figura 49 - Lista de las tablas que componen una Base de Datos II

Nota: Hay que estar seguros cuando se desee borrar una tabla porque, al contrario que pasaba al intentar eliminar una base de datos, al borrar la tabla se elimina con ella todo lo que esta contenga.

Insertar datos en una tabla:

Se vuelve a crear la tabla eliminada anteriormente para poder seguir "trabajando" sobre ella.

A modo de ejemplo, se van a insertar varios datos en la tabla creada anteriormente. Esto se puede realizar mediante la siguiente sintaxis:

```
INSERT INTO nombre_tabla (nombre_col_1, nombre_col_2, nombre_col_3) VALUES ('valor1', 'valor2', 'valor3');
```

Se van a añadir dos tuplas en el ejemplo:

```
INSERT INTO coches (id_coche, modelo, potencia, color, precio) VALUES (1000, 'mondeo', 140, 'azul marino', 35000);

INSERT INTO coches (id_coche, modelo, potencia, color, precio) VALUES (1001, 'fiesta', 70, 'rojo fuego', 14500);
```

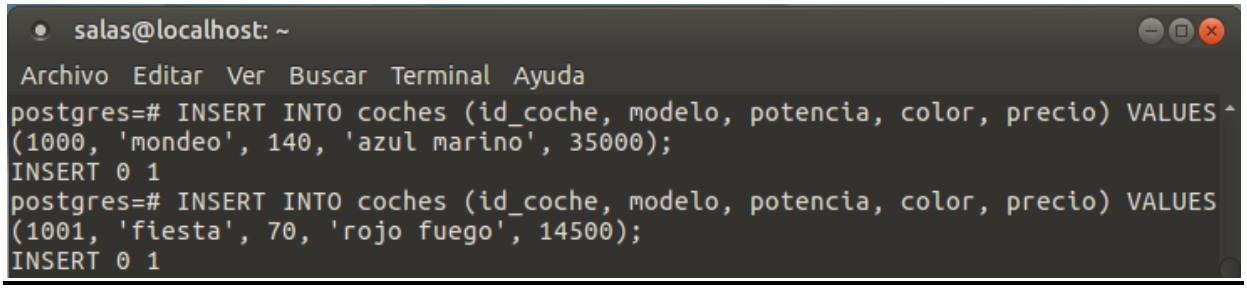

Figura 50 - Inserción de datos en una tabla de la Base de Datos

Consultar datos de una tabla:

Para ***consultar todos los datos*** que contiene una tabla se puede realizar mediante el comando:

```
SELECT * FROM nombre_tabla;
```

En el ejemplo se ejecuta:

```
SELECT * FROM coches;
```

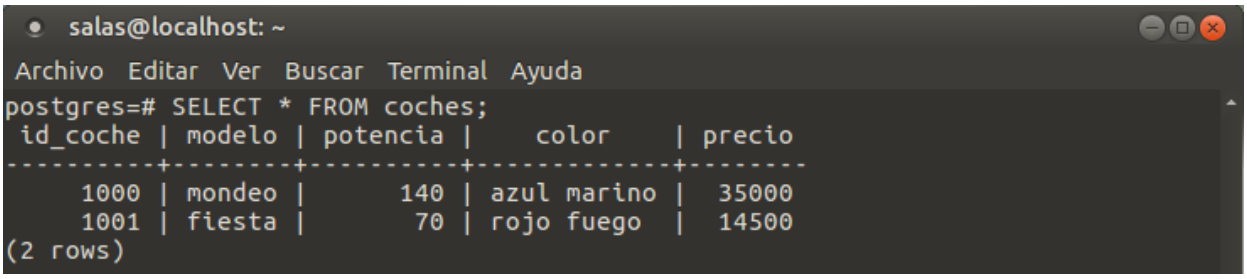

Figura 51 - Obtener todos los datos que contiene una tabla de la Base de Datos I

Se puede comprobar que aparecen todas las columnas incluidas en la creación de la tabla y las filas que han sido insertadas anteriormente.

En la consulta, se puede hacer un filtrado especificando los valores de campo que se deben cumplir, es decir, se especifican los valores del campo que se quieren mostrar en la consulta y ésta mostrará todas las filas que coincidan con dichos valores. Esto se puede lograr mediante la cláusula ***WHERE***. Por ejemplo, si se desea que la consulta sólo muestre los ***coches*** cuyo ***color*** sea ***"azul marino"*** tendrá que ser ejecutada la siguiente instrucción:

```
SELECT * FROM coches WHERE color='azul marino';
```

donde ***coches*** es el nombre de la tabla sobre la que realizamos la consulta y ***color='azul marino'*** es la columna y el valor que queremos que coincida en nuestra consulta. Todas las filas cuyo color sea ***"azul marino"*** serán mostradas en la consulta.

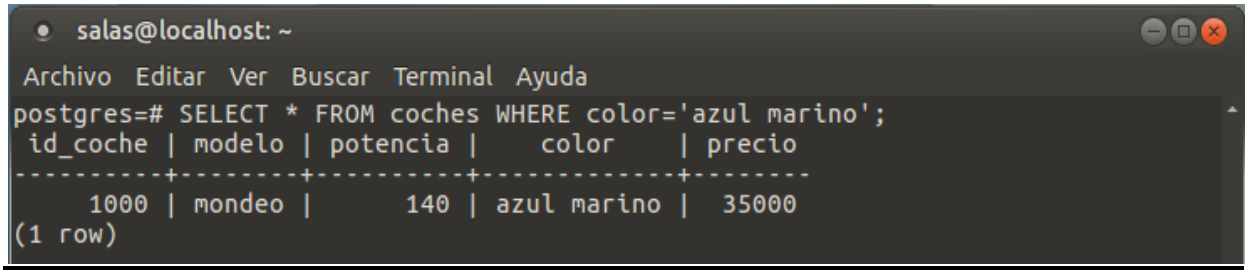

Figura 52 - Obtener los datos de una tabla que cumplan una determinada condición I (color ='azul marino')

Eliminar datos de una tabla:

Para ***eliminar todos los datos*** que contiene una tabla se puede realizar mediante el comando:

```
DELETE FROM nombre_tabla;
```

En el ejemplo se ejecuta:

```
DELETE FROM coches;
```

Se puede comprobar que la tabla se ha quedado sin ningún dato:

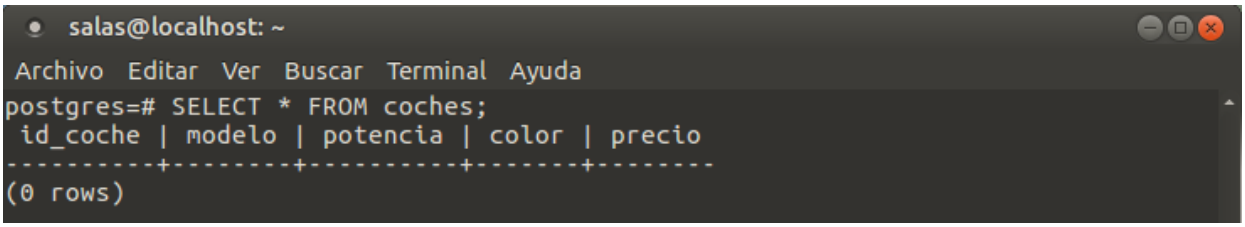

Figura 53 - Obtener todos los datos que contiene una tabla de la Base de Datos II

También puede ser usada la cláusula ***WHERE***para eliminar las filas que cumplan una determinada condición. Si, por ejemplo, se desea borrar las filas cuyo modelo sea ***"fiesta",*** se tendrá que ejecutar la siguiente condición:

```
DELETE FROM coches WHERE modelo='fiesta';
```

Como anteriormente se han borrado todas las filas, se vuelven a insertar y se ejecuta el comando anterior para borrar sólo las filas en las que el ***modelo*** sea ***"fiesta"***. El resultado es el siguiente:

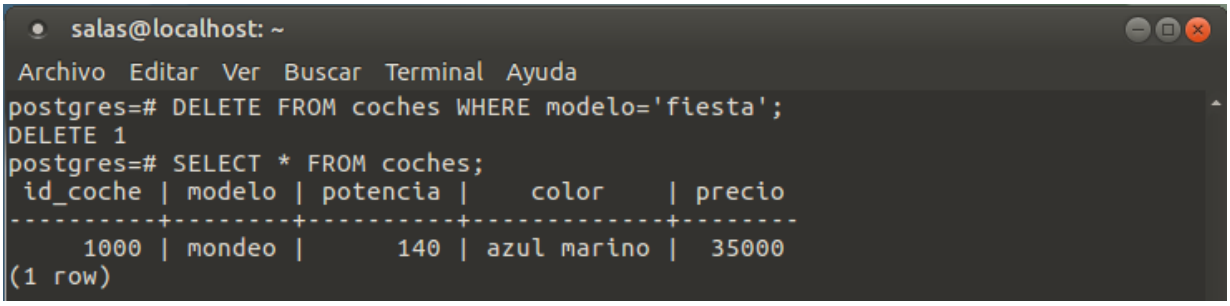

Figura 54–Eliminar los datos de una tabla que cumplan una condición específica (modelo='fiesta')

Añadir una columna a una tabla:

Si se desea***añadir una columna*** a una tabla después de su creación, se puede realizar utilizando la siguiente sintaxis:

```
ALTER TABLE nombre_tabla ADD nombre_columna tipo_columna;
```

En el ejemplo, se va a añadir una columna llamada ***cambio***(usada para especificar si el cambio del coche será manual o automático) que será de tipo carácter:

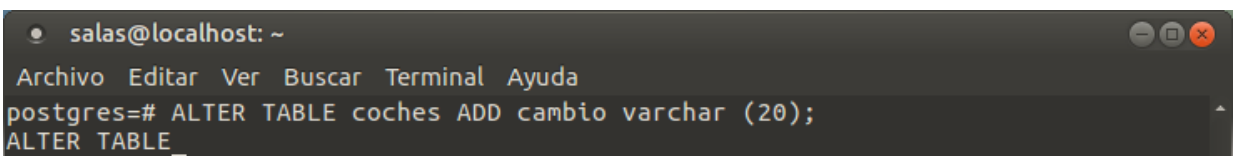

Figura 55 - Añadir una columna a una tabla de la Base de Datos después de su creación

Se comprueba que dicha columna ha sido añadida correctamente. Obviamente, en las filas que haya introducidas previamente a la inserción de esta nueva columna no habrá ningún valor en dicha

columna.

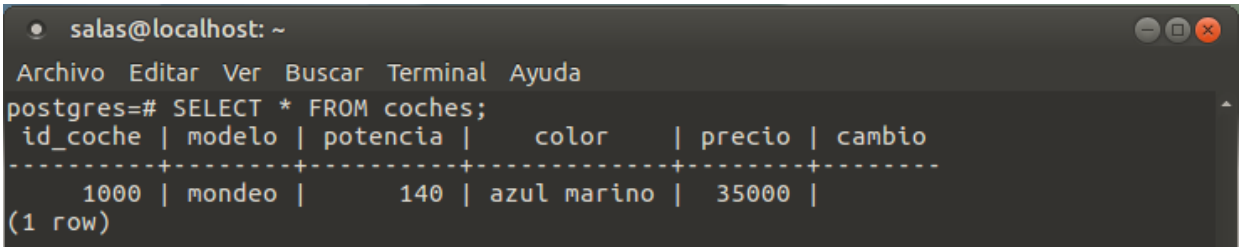

Figura 56 - Obtener todos los datos que contiene una tabla de la Base de Datos III

Eliminar una columna de una tabla:

Si se desea***eliminar una columna*** de una tabla,se puede realizar utilizando la siguiente sintaxis:

```
ALTER TABLE nombre_tabla DROP nombre_columna;
```

En el ejemplo, se va a eliminar la columna llamada ***cambio***que fue creada anteriormente:

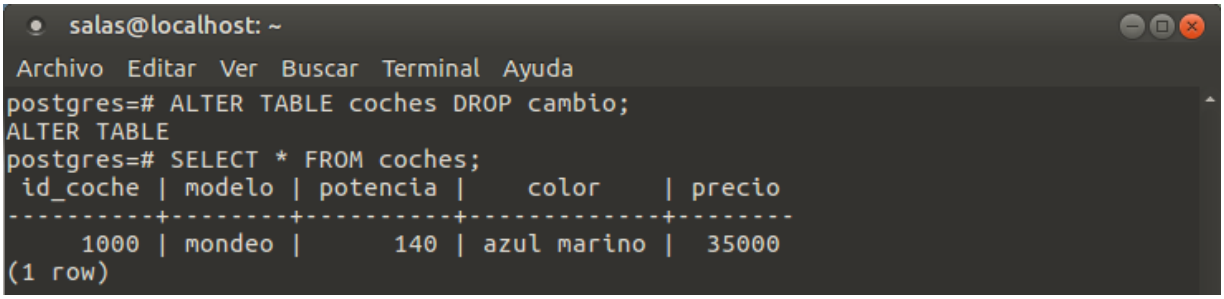

Figura 57 - Eliminar una columna de una tabla de la Base de Datos

Se verifica que la columna ***cambio*** ha sido eliminada.

Actualizar datos en una tabla:

Si se desea ***modificar los valores de una entrada existente***, se puede realizar consultando la fila/filas que se quieran modificar y estableciendo la columna en el valor que desea utilizar. Por ejemplo, si se desea modificar el ***color*** a ***"verde coral"*** de todos los coches cuyo ***modelo*** sea "***mondeo"*** (en el ejemplo parece absurdo ya que sólo tenemos una tupla, pero imaginemos que tenemos miles de tuplas). Sería realizado mediante la siguiente instrucción:

```
UPDATE coches SET color = 'verde coral' WHERE modelo = 'mondeo';
```

donde ***coches*** es la tabla de la cuál vamos a actualizar datos, ***color='verde coral'*** es el nuevo color que vamos a establecer a todos los coches que cumplan la condición de que su ***modelo*** sea ***"mondeo"***.

Se comprueba que el ***color*** del ***coche*** cuyo ***modelo*** es ***mondeo*** ahora es ***verde coral***:

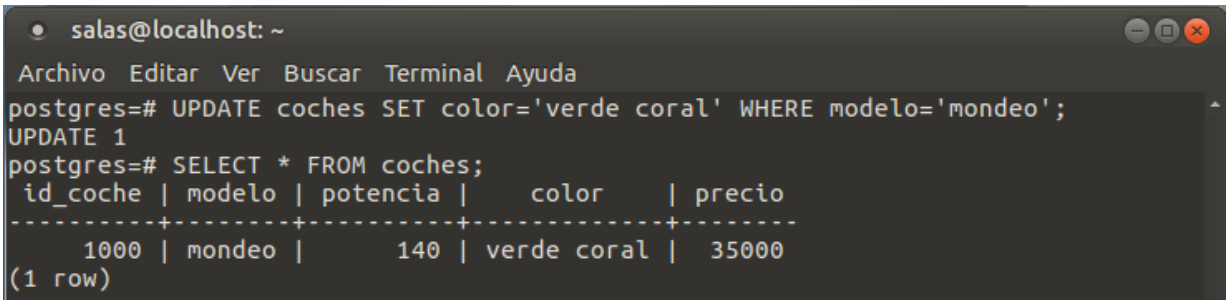

Figura 58 - Actualizar datos en una tabla de la Base de Datos

Otros criterios usados en la cláusula WHERE:

Se pueden usar muchos más criterios para limitar resultados dentro de la cláusula ***WHERE***. En este apartado, se verán algunos de los más útiles (pero hay muchos más).

Se pueden usar los ***operadores "<" y ">"***. Estos operadores se aplican sólo a campos numéricos y sirven para limitar los resultados de la consulta a los valores de la columna a evaluar que sean menores que el valor con el que se comparen en el primer caso y mayores en el segundo caso. Por ejemplo, supongamos que se desea seleccionar todos los ***coches*** cuyo ***precio*** sea ***menor que 35000 euros***. Para ello, se ejecuta la siguiente instrucción (para que haya alguna fila más en el ejemplo se volverá a añadir la fila que fue eliminada anteriormente):

```
SELECT * FROM coches WHERE precio < 35000;
```

Se comprueba que sólo aparece una fila que cumple el criterio especificado (ya que la en la otra fila el precio es de 35000 euros.):

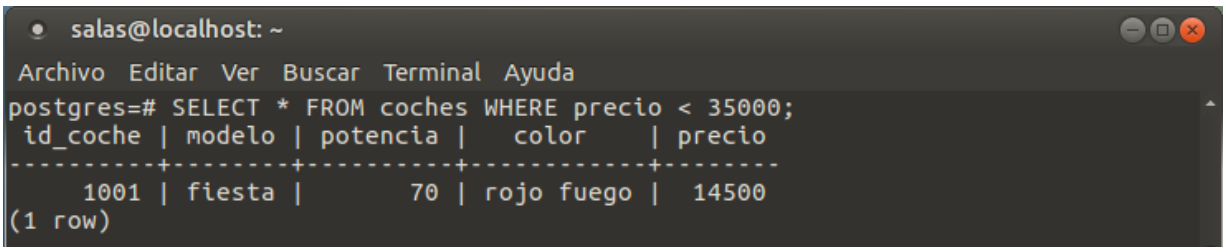

Figura 59 - Obtener los datos de una tabla que cumplan una determinada condición II (precio < 35000)

Otro criterio útil es el uso de ***Is Null***.Este criterio se aplica para mostrar los registros donde el valor del campo es nulo y se puede aplicar a cualquier tipo de campo. Por ejemplo, se va a comprobar si algún coche no tiene asignado un precio. Esto se realiza ejecutando:

```
SELECT * FROM coches WHERE precio Is Null;
```

Se comprueba que la consulta realizada no arroja ningún resultado, ya que las dos filas que se encuentran en la tabla coches tienen asignado un precio:

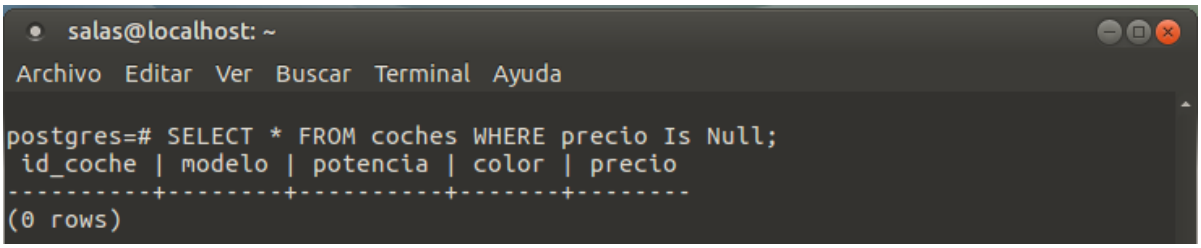

Figura 60 - Obtener los datos de una tabla que cumplan una determinada condición III (precio Is Null)

Se puede usar la cláusula ***AND*** dentro de la cláusula ***WHERE***. Usando esta cláusula, los resultados arrojados en la consulta tendrán que cumplir todas las condiciones unidas por dicha cláusula. Por ejemplo, se desea seleccionar todos los ***coches*** que cumplan que su ***modelo*** sea ***"fiesta"*** y su ***potencia*** sea de ***70 cv***. Esto se realiza ejecutando la siguiente instrucción:

```
SELECT * FROM coches WHERE modelo='fiesta' AND potencia=70;
```

Se comprueba que el resultado de la consulta es una fila que cumple esas dos condiciones:

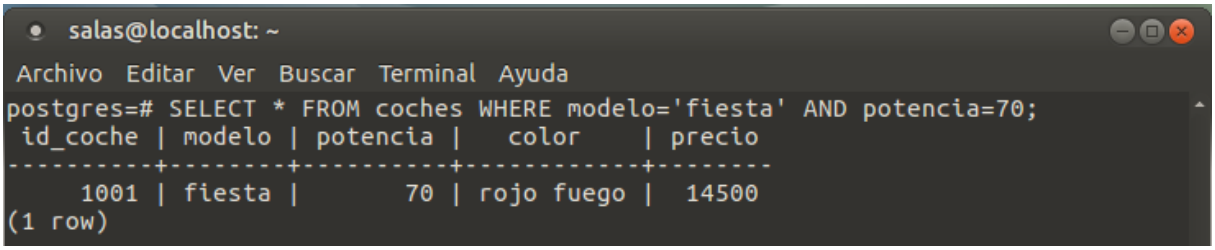

Figura 61 - Obtener los datos de una tabla que cumplan dos condiciones (modelo='fiesta' AND potencia=70)

También se puede hacer uso de la cláusula ***OR*** dentro de la cláusula ***WHERE***. Usando esta cláusula, los resultados arrojados en la consulta tendrán que cumplir cualquiera de las condiciones unidas por dicha cláusula. Por ejemplo, se desea seleccionar todos los ***coches*** que cumplan que su ***modelo*** sea ***"fiesta"*** o su ***potencia*** sea de ***140 cv***. Esto se realiza ejecutando la siguiente instrucción:

```
SELECT * FROM coches WHERE modelo='fiesta' OR potencia=140;
```

Se comprueba que el resultado de la consulta realizada son dos filas, una porque cumple que su ***modelo*** es ***"fiesta"*** y otra porque cumple que su ***potencia*** es de ***140 cv***:

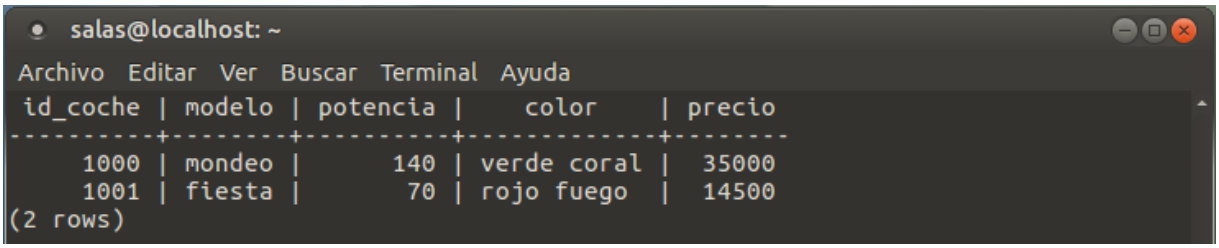

Figura 62 - Obtener los datos de una tabla que cumplan una determinada condición (modelo=’fiesta’) u otra (potencia=140)