Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Infraestructura de control del alquiler de bicicletas mediante dispositivos móviles

Proyecto Final de Carrera para
Ingeniero Técnico en Informática de Sistemas (ITIS)

Valencia 21 de Septiembre del 2012

Autor: Antonio Albiñana Martínez
Directores: Joaquín Gracia Morán
Juan Carlos Ruiz Garcia

pagina en blanco
intencionada

Índice de contenidos

Prólogo....5

Resumen....7

Introducción....9

Arquitectura de la infraestructura....11
- Capa Presentación....11
  - Software utilizado....11
  - Seguridad....12
- Capa Lógica de aplicación....13
  - Máquina Virtual....14
  - Servlets....15
  - Software Utilizado....16
    - Apache Tomcat....16
    - Eclipse....18
  - Seguridad....19
    - Acceso y Autentificación....20
    - Control de Decisiones....21
- Capa Persistencia de datos....22
    - Software....22
    - Seguridad....23

Bases de datos utilizadas....25
- Base de datos "usuarios"....25
- Base de datos "Histórico"....25
- Base de datos "puestos"....26
- Base de datos "bicicletas_puesto"....26
- Base de datos "incidencias"....26

Funcionamiento de la infraestructura....29
- Detalle del procedimiento....29
  - Transacciones....33
    - Alquiler....33
    - Devolución....34
- Integridad del e-ticket....36
  - Datos en el cliente....37
- Estructura de los servlets....38
  - Dependencias externas....39
- Sesiones....39
- Clases Java especiales....40
  - La clase Cadenas....40
  - La clase LCookies....42
  - La clase ConexionBD....43

La clase GestionLogs.......... 46
Constantes.......... 46
Método escribirLogs.......... 47
La clase Ticket.......... 48
Filtros.......... 49
Configuración del FiltroAcceso.java.......... 51
Referencias.......... 53
Palabras clave.......... 55

PRÓLOGO

En los últimos tiempos, el desarrollo tecnológico ha propiciado nuevas formas de comunicación. Lo que hace una década parecía ciencia-ficción hoy ya está anticuado.

Por encima de todas ellas se encuentran los dispositivos móviles, que han pasado en diez años de ser un simple teléfono a ser un "mini-ordenador", con una interconexión y potencia de cálculo impensable al inicio del siglo XXI. Sirva como ejemplo que la cámara en los teléfonos móviles empieza a aparecer alrededor del año 2002, y actualmente forma parte del dispositivo, y con unas prestaciones equiparables a cámaras fotográficas compactas.

La aparición de sistemas operativos específicos, como Android, diseñado para este tipo de dispositivos, ha posibilitado que la generación de aplicaciones tanto para el ocio como de negocio, haya crecido considerablemente en los últimos años, siendo las expectativas de crecimiento aún mayor.

La naturalidad y facilidad de manejo que proporcionan dichos dispositivos en la actualidad ayuda a que la incorporación de estas nuevas tecnologías a los hábitos de uso de los usuarios finales se realice de manera sencilla y con normalidad, ya que ésta se incluye en un dispositivo que el usuario conoce y está acostumbrado a usar.

En este contexto, el uso de dichos dispositivos móviles inteligentes (también denominados *smartphones*) se está generalizando de tal manera, que permite ampliar la funcionalidad de los mismos, complementando, sustituyendo o realizando diferentes tareas. Este proyecto está enfocado en esa dirección.

pagina en blanco
intencionada

RESUMEN

La idea general del proyecto consiste en la creación de una infraestructura para el control del alquiler de bicicletas. Esta infraestructura permitirá, en un futuro, utilizar dispositivos móviles para acceder a ella y alquilar una bicicleta. De esta forma, y tal y como se ha desarrollado el presente PFC, no será necesario la instalación de ninguna aplicación extra, ya que se va a utilizar el navegador que tienen los dispositivos móviles actuales incorporados por defecto.

El funcionamiento de la infraestructura es el siguiente. El primer paso consiste en el alquiler de la bicicleta, alquiler que se realiza previa validación del usuario en el sistema. Una vez cumplimentada la acción del alquiler, la aplicación devolverá un *e-ticket* con la información necesaria para realizar las operaciones de control necesarias, registrándose además esta operación en el sistema.

Al devolver la bicicleta, el *e-ticket* se borra del teléfono, registrándose también esta operación en el sistema. En este caso, la devolución de la bicicleta pone en marcha el proceso de facturación.

Para la realización del presente PFC se utilizarán herramientas de código abierto, tanto para el servidor, como para la lógica de aplicación y para la gestión de las bases de datos. Estos programas, así como la infraestructura y la operatividad entre ellos deberán instalarse y configurarse, mientras que el navegador, tal y como se ha comentado, ya forma parte del dispositivo del usuario (smartphone).

Todas las herramientas y tecnologías utilizadas serán explicadas en detalle más adelante, dentro de su contexto de uso. Brevemente, podemos decir que vamos a utilizar:

Herramientas :

- ✔ Apache Tomcat
- ✔ Eclipse
- ✔ PostgresSQL

Tecnologías :

- ✔ HTTPS
- ✔ Servlets (Java)

pagina en blanco intencionada

INTRODUCCIÓN

El presente proyecto pretende crear, tal y como se ha mencionado, la infraestructura necesaria para el control del alquiler de bicicletas mediante dispositivos móviles inteligentes (smartphones). Dichos dispositivos suponen en la actualidad el 45% de mercado y se espera un crecimiento exponencial de los mismos. Un factor importante en este crecimiento es la fuerte apuesta que ha realizado "Google" en el desarrollo del sistema operativo "Android", siendo ésta la parte fundamental de este impulso. Este esfuerzo le ha llevado a conseguir una cuota de mercado superior al 80% en los smartphones vendidos entre mayo y agosto del 2012.

Una de las principales características de los nuevos smartphones es la inclusión de nuevas tecnologías. Un ejemplo es el NFC. Esta nueva tecnología está basada en tecnologías RFID (Identificación por Radio Frecuencia), por lo que es necesario una etiqueta y un lector. El lector puede estar contenido en cualquier dispositivo como, por ejemplo, un teléfono móvil.

La tecnología NFC presenta dos modos de funcionamiento :

- Pasivo, en las que solo un dispositivo genera el campo electromagnético, y el otro se aprovecha de la modulación para poder trasferir los datos. En este caso, el dispositivo que inicia la comunicación es quien genera dicho campo.
- Activo, en el que ambos dispositivos generan campos electromagnéticos. En este caso, ambos dispositivos necesitan energía.

Cuando el lector se aproxima a una etiqueta RFID o a otro lector emite una señal de radio de corto alcance que excita el microchip de la etiqueta, con lo que podremos acceder a leer la pequeña cantidad de datos que se encuentran almacenados en ésta. En el caso de la comunicación con etiquetas o *tags,* es el lector el encargado de establecer la comunicación.

En el protocolo NFC siempre hay un dispositivo que inicia la comunicación, siendo este dispositivo el encargado de monitorizarla. NFC permite tres modos de comunicación, que son:

- Punto a punto. Utilizado para el intercambio de datos o establecimiento

de las comunicaciones entre dispositivos NFC (utilizando el propio protocolo para unos pocos Kb).

- Lectura-escritura. Tiene la capacidad de leer o escribir etiquetas. Suele utilizarse para los denominados pósters inteligentes, ya que al leer la etiqueta incluida en el póster, ésta trasmite al teléfono la dirección de una página web, abriendo automáticamente el navegador.
- Emulación de tarjeta. En este modo, el dispositivo NFC se comporta como una tarjeta inteligente, apareciendo ante el lector como una tarjeta sin contacto. Con esta configuración se pueden utilizar las características del elemento de seguridad incorporado como medio de pago, así como para el almacenamiento y gestión de todo tipo de entradas y recibos.

Con esta nueva tecnología se pueden por tanto realizar pagos, publicidad mediante smart posters y hasta llevar un control de acceso. Gran parte de esta operabilidad y versatilidad viene determinada por su corto alcance, lo que la hace inherentemente segura, así como por la rapidez para gestionar una conexión (0'1 seg. frente a los 6 seg. del Bluetooth), lo que la hace ideal para la transmisión de pequeñas cantidades de datos.

El presente proyecto va a implementar una infraestructura en la que un smartphone con tecnología NFC podría funcionar perfectamente con unos cambios mínimos en dicha infraestructura.

Arquitectura de la infraestructura

Para la creación de la infraestructura mencionada anteriormente se ha usado una organización de "3 capas", la cual permite agrupar las acciones según su cometido y separarlas por su funcionalidad. La siguiente figura (Figura 1) representa esta disposición lógica.

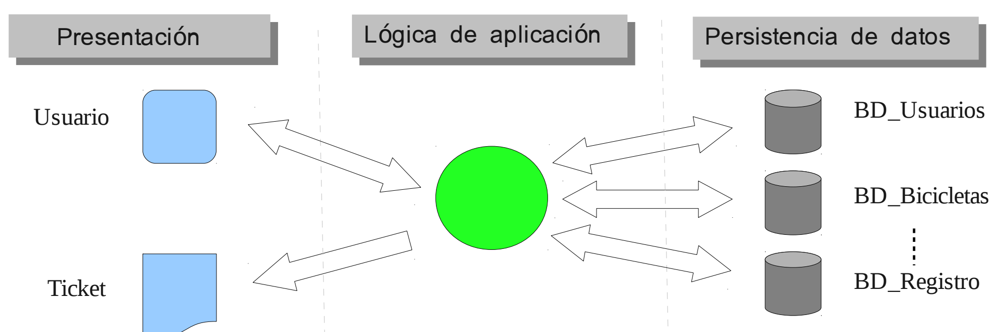

Figura 1. *Arquitectura de 3 capas.*

A continuación se explican detalladamente las tres capas expuestas en la Figura 1.

Capa Presentación

Esta capa se refiere principalmente a lo que el cliente observa a través de la aplicación que se ejecuta en su equipo. En este caso, al tratarse de un navegador, la aplicación va a permitir al usuario interactuar con páginas HTML.

Dicha presentación es proporcionada por la capa denominada "Lógica de aplicación", que es la encargada de gestionar, formatear y entregar la información necesaria para que el navegador la visualice correctamente.

Software utilizado

Dado que el navegador ya está incluido en el dispositivo móvil del cliente, no hace falta instalar ningún software adicional. De la misma manera sería posible

sustituir dicho navegador por un programa que interactuase con la capa "Lógica de aplicación", añadiendo de esta manera nuevas características, como podrían ser la gestión y el mantenimiento de un histórico de alquileres, su geolocalización, la duración del alquiler, etc., incorporando dicha información a una base de datos. Por ejemplo, "Android" incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite.

SEGURIDAD

La seguridad en esta capa viene determinada por la comunicación entre la capa "Presentación" y la capa "Lógica de aplicación". El funcionamiento normal de esta comunicación se realiza mediante el protocolo HTTP. Como es sabido, este protocolo manda sus mensajes en claro, por lo que la información intercambiada es fácilmente accesible.

Para garantizar una comunicación segura, en este punto de la infraestructura se ha utilizado el protocolo HTTPS. Este protocolo proporciona una comunicación segura sobre HTTP. Fue creado por Netscape en 1994 utilizando inicialmente SSL (*Secure Socket Layer*), aunque en la actualidad utiliza TSL (Transport Layer Security). Por lo tanto, es indispensable que tanto el navegador del usuario, como el servidor web sean capaces de manejar dicho protocolo.

Al utilizar HTTPS, la comunicación entre el usuario y la capa "Lógica de aplicación" se cifra. En concreto, la comunicación se realiza mediante cifrado simétrico, que es mucho menos costoso computacionalmente. Sin embargo, la negociación y comunicación de la clave simétrica se realiza mediante clave asimétrica, mucho más costosa de realizar pero más segura. Su funcionamiento es el siguiente:

- ✔ Cuando un cliente se conecta con un servidor seguro, se produce una autentificación para obtener el certificado del servidor.
- ✔ Con ese certificado el cliente genera y negocia una clave privada con el servidor, cifrando la comunicación con la clave pública del servidor. De esta manera, solo el servidor puede descifrarla mediante su clave privada.
- ✔ Una vez negociada la clave simétrica, toda la comunicación entre el cliente y el servidor se realiza cifrándola con dicha clave simétrica.
- ✔ Durante la comunicación posterior, cualquiera de las partes puede solicitar la negociación de una nueva clave simétrica.

Capa Lógica de aplicación

Esta parte se ejecuta en el servidor web. Su funcionalidad básica es responder a las peticiones de los clientes y acceder a los datos de la aplicación, ya estén accesibles internamente o externamente, accediendo a la capa "Persistencia de datos".

Es en esta capa donde se concentra la mayor parte (por no decir todas) las tomas de decisiones importantes. Como se puede ver en la Figura 1, se comunica tanto con la capa "Presentación" como con la capa "Persistencia de datos".

Este esquema permite separar claramente el funcionamiento de la aplicación. Por un lado, se tiene el interfaz con el usuario (navegador web, programa, etc.) y por otro lado toda la funcionalidad del almacenamiento de los datos (bases de datos, ficheros, etc.). De esta manera, en el caso de variar la forma de presentación o de acceso a los datos, solo habrá que modificar la parte del código que se encarga de devolver la información ya procesada y formateada al cliente para su correcta visualización y/o la parte encargada de obtener la información externa, pero no el resto del código.

Tanto el software como la tecnología utilizada en la capa "Lógica de aplicación" se analizarán en detalle más adelante, aunque a continuación se van a presentar brevemente a modo de introducción.

Para la lógica de control de la aplicación se han utilizado servlets. De esta manera, al usar "Java" y una estructura normalizada se consigue independencia del software del servidor web utilizado. Así pues, podemos utilizar un entorno de desarrollo (en nuestro caso Eclipse) que además de tener habilitados los mecanismos necesarios para poder generar el código que utilizará el servidor web cuando el proyecto se exporte, permite la depuración de los errores sintácticos del código en tiempo de desarrollo.

Como se ha comentado anteriormente, la capa "Lógica de aplicación" interactúa tanto con la capa "Presentación" como con la capa "Persistencia de datos". Esta interacción es gestionada por un contenedor (también denominado motor de servlets), el cual posee la capacidad de dar soporte a los servlets. En nuestro caso, como servidor web se ha elegido "Apache Tomcat" por ser gratuito, por poder trabajar con servlets y por disponer de una amplia y detallada documentación así como una gran comunidad de usuarios, lo que proporciona una cantidad muy elevada de información sobre su funcionamiento, configuración y optimización.

Todo este software se ha instalado sobre una máquina virtual. A continuación se detallan las particularidades de todo el software utilizado en esta capa.

MÁQUINA VIRTUAL

Todo el proyecto se ha implementado dentro de una máquina virtual. De esta manera se consigue tener una fácil portabilidad del mismo, así como se puede independizar el desarrollo del proyecto respecto del sistema operativo instalado en la máquina física utilizada.

Una máquina virtual es, en resumen, un sistema operativo (invitado) funcionando sobre otro sistema operativo (host). Se ha optado por esta manera de trabajar porque el sistema operativo invitado permite la independencia respecto del sistema operativo host instalado en la máquina física, proporcionando además una mayor versatilidad, y sin penalizar el rendimiento de sistema operativo host, ya que en la actualidad la virtualización apenas influye en este aspecto.

La máquina virtual utilizada es `VMwvare.` Esta herramienta es gratuita y permite la creación y gestión de máquinas virtuales. Para poder utilizar máquinas virtuales, lo primero que hay que instalar es el programa gestor de máquinas virtuales, denominado en este caso `vmware player`. Este software se puede descargar de la siguiente dirección:

`http://www.vmware.com/download/player/download.html.`

Una vez en funcionamiento dicho programa, procederemos a crear una nueva máquina virtual sobre la que deberemos instalar el sistema operativo invitado sobre el que van a funcionar el resto de componentes. La instalación del sistema operativo invitado es un procedimiento guiado, pudiendo utilizarse las opciones por defecto. Este proceso de instalación se compone de los siguientes pasos:

- Seleccionar la imagen ISO previamente descargada.
- Proporcionar el nombre de usuario del operativo a instalar y su password.
- Decidir el nombre que recibirá la máquina virtual y su localización.
- Seleccionar el tamaño del disco duro virtual que se utilizara.
- Instalar el sistema operativo invitado utilizando la configuración por

defecto que nos propone.

En la Figura 2 se puede ver un resumen de la creación de la máquina virtual.

Figura 2. Resumen de la creación de la maquina virtual.

Como sistema operativo invitado se ha utilizado Ubuntu 12.04 LTS[1]. La imagen iso del mismo se puede descargar desde:

http://www.ubuntu.com/download/desktop

Servlets

Un servlet es un componente web que está realizado en JAVA, lo que permite utilizar entornos de desarrollo avanzados, facilitando de esta forma la programación y corrección de errores sintácticos.

Dentro de la implementación del servlet pueden utilizarse clases para añadir funcionalidad y propiedades que son imposibles o complicadas de realizar mediante métodos tradicionales como los CGI's. Una de las características de los servlets es su simplicidad. Un servlet se invoca con solo dos parámetros: uno con la petición y otro con la respuesta generada dinámicamente. Además, también proporciona independencia sobre el servidor. Por otro lado, la interfaz de programación proporciona abstracciones, como las sesiones. Una sesión es

1 *LTS es la abreviatura de "Long Term Support" (soporte a largo plazo) e indica que se tendrá soporte de esta versión de al menos tres años para la versión desktop, y cinco años para el servidor.*

la manera en la que el contenedor de aplicaciones (el servidor web) relaciona la petición actual con otras peticiones previas. Con servlets existen tres maneras de implementar una sesión:

- Mediante SSL y por tanto HTTPS[2].
- Mediante Cookies.
- Mediante reescritura de la URL, en el que el id de la sesión se codifica como parámetro en la cadena URL. Por ejemplo:

Http://www...../tienda/index.thml;jssesionid=1234

Un servlet se llama a través del nombre de su contenedor dentro del sistema web, de manera que podemos decir que el contenedor es el nombre de la aplicación y el servlet el nombre de la acción a realizar. A continuación podemos ver un ejemplo de invocación de un servlet:

Http://www..../NombreContenedor/servletInvocado

¿Por qué no se ha seleccionado JSP? Básicamente por dos razones:

- Un JSP se debe convertir en un servlet de manera interna antes de ejecutarse.
- Hay más código de aplicación que código HTML, lo que produciría un código ilegible tanto a nivel HTML como JAVA .

Software Utilizado

Apache Tomcat

Como se han utilizado `servlets` para la gestión de la lógica de control y funcionamiento, se necesita un servidor capaz de poder trabajar con dicha tecnología. Se ha elegido `apache tomcat` por ser gratuito y disponer de una amplia y detallada documentación así como de una gran comunidad de usuarios.

La versión utilizada es la 6, que se puede obtener de la fundación apache a partir del siguiente enlace:

`http://tomcat.apache.org/download-60.cgi`

2 *Este ha sido el modelo elegido para este proyecto ya que HTTPS incorpora mecanismos para distinguir los accesos que forman parte de una sesión.*

Una vez descargados los binarios, se descomprimen en un directorio y ya está disponible el servidor para su puesta en marcha y funcionamiento básicos, aunque en este punto todavía no atiende peticiones HTTPS.

Para iniciar y detener el servidor utilizaremos los siguientes scripts[3]

```
[DIR_INSTALACION]/bin/startup.sh
[DIR_INSTALACION]/bin/shutdown.sh
```

Para poder habilitar `HTTPS` en el servidor, necesitamos un certificado. Existen dos maneras de conseguirlo. La más sencilla es crear un certificado[4] autofirmado. Esta es la solución que se ha tomado en este caso. El otro método de obtención de un certificado consiste, además, en crear y añadir una autoridad certificadora.

Así pues, para crear el certificado autofirmado se puede utilizar la herramienta `keytool.` Esta utilidad se encarga de la gestión de claves y certificados. Permite a los usuarios administrar sus propios pares de claves públicas/privadas y los certificados asociados para su uso en la auto-autenticación (donde el usuario se autentica mismo). Se invoca mediante el siguiente comando:

```
# $JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
```

La ejecución de este comando genera una serie de preguntas, que tras haberlas contestado, provoca la creación del fichero `.keystrobe` en el `$home` del usuario. El siguiente paso consiste en la modificación del fichero de configuración denominado `server.xml` para habilitar SSL, indicar donde se encuentra el fichero `.keystore` e indicar el *password*[5] con el que se creó.

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               keystoreFile="${user.home}/.keystore"
                    keystorePass="proyecto"
               clientAuth="false" sslProtocol="TLS" />
```

3 *Un script es una secuencia de órdenes que puede ser ejecutado directamente por el intérprete de comandos.*

4 *Un certificado es una declaración firmada digitalmente de una entidad (persona, empresa, etc). Cuando los datos están firmados digitalmente, se puede verificar que los mismos no han sido modificados (integridad) y que dichos datos vienen de quien dice haber creado y firmado (autenticidad).*

5 *Apache Tomcat utiliza el password por defecto "changeit". En nuestro caso se ha puesto "proyecto"*

Un aspecto a tener en cuenta es la disposición de los directorios de trabajo. En nuestro caso, esta disposición es distinta a la del servidor apache normal. En este caso, las páginas web 'normales' están en el directorio

```
[DIR_INSTALACION]/ROOT
```

mientras que los generadores de contenidos dinámicos se encuentran en el directorio

```
[DIR_INSTALACION]/Webapps
```

El servidor, que en este caso se comporta también como un contenedor de aplicaciones (motor de `servlets`) se encarga automáticamente de comprobar si se han añadido nuevas aplicaciones para darlas de alta en el propio servidor y habilitar su funcionamiento.

El formato estándar de fichero utilizado para esta acción es '.war', el cual contiene toda la información necesaria.

ECLIPSE

Como se ha comentado previamente, para la lógica de control de la infraestructura se han utilizado servlets. Para realizar la programación de los mismos se ha utilizado una versión de esta herramienta de programación denominada Indigo, la cual ya tiene habilitados los mecanismos necesarios para poder generar el código que utilizará "Apache Tomcat" cuando el proyecto se exporte, facilitando de esta manera la generación de código.

Eclipse es un proyecto open source de la fundación Eclipse para el desarrollo de un entorno de desarrollo integrado (IDE). Esta herramienta es de código abierto y está reconocida por la Free Softaware Foundation, aunque su licencia no es GNU sino EPL (Eclipse Public License). La dirección de descarga es :

```
http://www.eclipse.org/downloads/packages/release/indigo/sr2
```

La utilización de un entorno de trabajo de programación facilita la generación de código, ya que permite detectar errores en una etapa temprana del desarrollo del proyecto. Este tipo de errores son difíciles de encontrar en tiempo de ejecución, ya que incluyen desde errores sintácticos, como pueden

ser palabras mal escritas, hasta clases mal construidas y que son muy fáciles de detectar en la etapa de desarrollo.

Para generar una aplicación web desde eclipse deben seguirse los siguiente pasos.

- Se genera un nuevo proyecto de web dinámico.
- Se crean los `servlets` y clases auxiliares necesarias.
- Se exporta el proyecto como “`war`”.

La Figura 3 muestra un ejemplo de exportación de un proyecto.

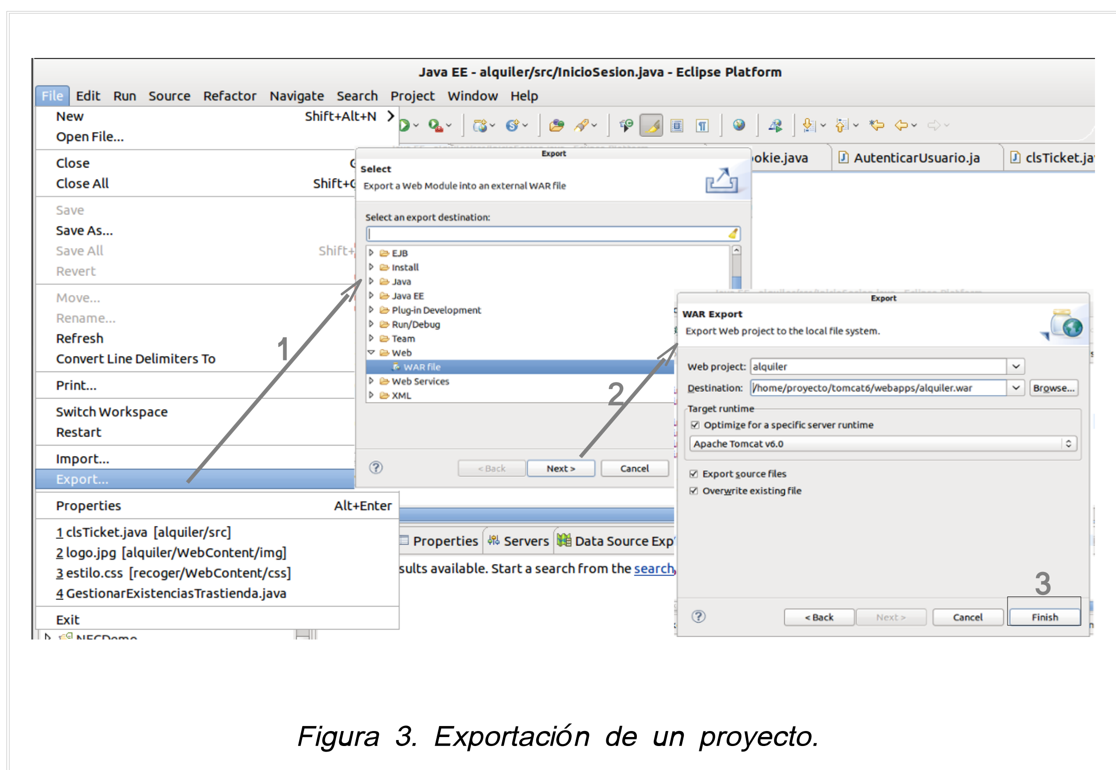

Figura 3. Exportación de un proyecto.

Seguridad

La seguridad en este capa viene caracterizada por dos aspectos complementarios entre sí. El primero de ellos es el acceso y el autenticado de los usuarios. En este caso, el sistema debe permitir solo el acceso a aquellos usuarios válidos en el sistema. Para ello, se ha utilizado un mecanismo proporcionado por “Apache Tomcat” denominado filtros, los cuales se ponen en funcionamiento antes de acceder al recurso solicitado.

El segundo aspecto tiene que ver, en primer lugar, con la manera de realizar la comprobaciones internas del control de flujo y en segundo lugar, con la forma de acotar la toma de decisiones a casos concretos, evitando la generalización.

Acceso y Autentificación

Los filtros son los principales encargados de implementar esta parte de la seguridad. Un filtro es un mecanismo que permite transformar el contenido de las peticiones, respuestas y cabeceras de los mensajes HTTP. Pueden trabajar tanto sobre contenido dinámico como estático. En el presente proyecto solo se utiliza el contenido estático, ya que el propósito es evitar que se pueda acceder al alquiler de una bicicleta sin estar autenticado. La Figura 4 muestra el esquema de funcionamiento del filtro:

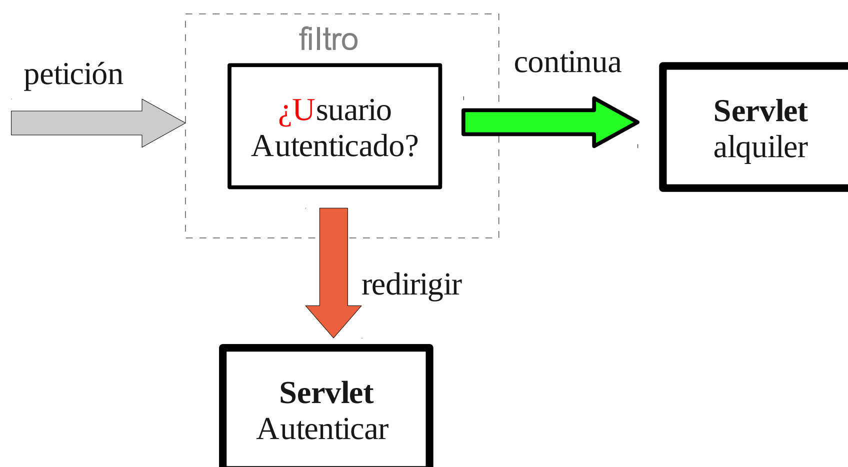

Figura 4. Esquema de funcionamiento del filtro.

De esta manera, cuando un usuario intenta alquilar una bicicleta, y antes de acceder a ningún recurso, se comprueba si el usuario está autenticado mediante la comprobación de la existencia de una sesión y si ya se ha autenticado previamente.

El aspecto más importante de este tipo de control es, como ya se ha comentado, que se realiza antes de acceder al servlet que se encarga de gestionar el alquiler. Así se evitan errores durante la gestión y tratamiento en el alquiler referentes al usuario, los cuales pueden producir algún error que permitiera el alquiler a un usuario no válido en el sistema.

Respecto al acceso posterior a los servlets encargados del alquiler o la devolución, se han inhabilitado los accesos críticos con parámetros a través de la URL (método GET), limitando su uso solo a aquellos procesos que es inevitable su uso, principalmente redirecciones por falta de parámetros. Así pues, se ha utilizado el método POST para el envío de los datos necesarios para el alquiler. Este método pasa los parámetros a través de la entrada estándar, además de no ser guardados en el navegador o en los logs de los servidores, evitando de esta manera que un usuario malintencionado pudiera alquilar una bicicleta poniendo como dirección del navegador "https://...../alquilar?puesto=xx?en=xx." o similares.

En este sentido, también los parámetros entre servlets relacionados con el alquiler/devolución (número de bicicleta, número de puesto) se pasan a través de la sesión, evitando así que un usuario no autenticado pueda acceder a esos datos durante el proceso de alquiler/devolución.

Control de Decisiones

En este apartado la seguridad está encaminada a impedir, en la medida de lo posible, los errores derivados de la información que debe ser validada antes de ser enviada al servidor a través de la aplicación web.

Están agrupadas aquí también las decisiones de control de flujo del programa, es decir los `if`, `while` y demás estructuras que utilizan comparaciones, inclusiones, etc. en la toma de decisiones.

Por ejemplo, se ha controlado la introducción de datos numéricos (del puesto) mediante un desplegable para evitar que se introduzcan por error datos no válidos. De la misma manera, se comprueba que el número de puesto sea realmente un número antes de proceder a realizar ninguna acción.

También se han limitado los caracteres válidos de usuario y password, de manera que son inválidos caracteres como "%" o "!=", puesto que pueden ser utilizados para realizar ataques de inyección SQL.

Para las sentencias de control se ha utilizado la técnica denominada "especificación positiva de valores admisibles". Con dicha técnica determinamos si sabemos las condiciones válidas para que se cumpla una determinada condición. Si esas condiciones se cumplen, entonces esas son las que permiten el acceso. De esta manera solo las opciones válidas continúan, mientras que el resto (las no válidas + todas las demás) no pueden realizar ninguna acción. A continuación podemos ver un ejemplo:

If (hay error)
 	
→ Error
 	
else
 	
→ Ok + no contempladas
	If (está bien)
	
	→ Ok
	
	else
	
	→ Error + no contempladas

Especificación negativa	*Especificación positiva*

Capa Persistencia de datos

Esta capa representa el sistema mediante el cual se guarda la información. Normalmente consta de un conjunto de bases de datos con la lógica asociada. Como en los casos anteriores, se ha optado por un gestor de base de datos de código abierto, que en nuestro caso es el denominado PostgreSQL.

Software

PostgresSQL es el motor de base de datos de código abierto más potente del mercado. Es distribuido bajo licencia BSD[6] y con su código fuente disponible libremente. Este software se encuentra disponible en los repositorios de Ubuntu y, por lo tanto, se puede instalar desde el gestor gráfico de aplicaciones (Synaptic en el caso de Ubuntu) o desde consola mediante la orden:

```
apt-get install postgresql-9.1
```

Después de instalar tanto el motor como el gestor gráfico `pgAdmin`, se crearán las bases de datos necesarias para el funcionamiento de la aplicación, como son la de Usuarios, Histórico, Puestos, bicicletas_Puestos, etc. que se describen en detalle más adelante. La Figura 5 muestra el aspecto gráfico de las bases de datos.

6 *La licencia BSD es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Es una licencia de software libre permisiva, estando muy cercana al dominio público. La licencia BSD permite el uso del código fuente en software no libre.*

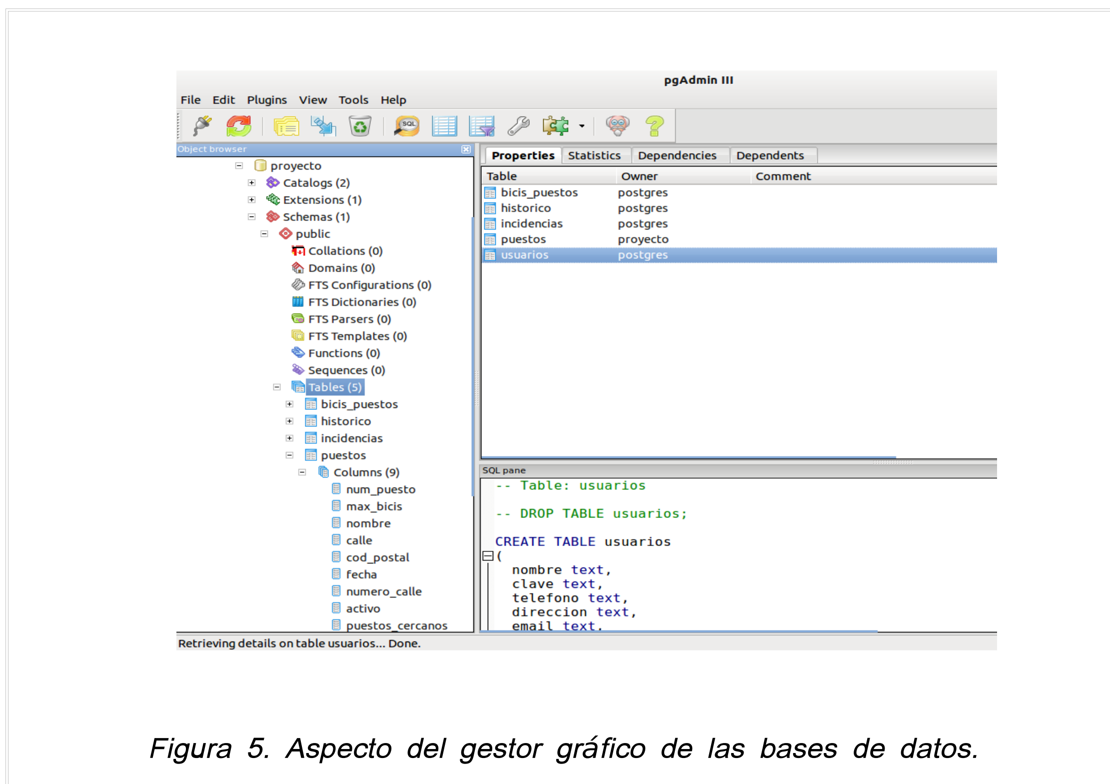

Figura 5. Aspecto del gestor gráfico de las bases de datos.

Seguridad

La seguridad en esta capa recae en mecanismos propios de la gestión de base de datos, como son el password de acceso, que en nuestro caso es "proyecto". El resto de opciones de seguridad, así como las copias de seguridad periódicas de las bases de datos, la redundancia, etc. no forman parte de este proyecto.

La siguiente porción de código muestra la conexión a la base de datos.

```
static public final String BD_URL="jdbc:postgresql://localhost/proyecto";

static public final String BD_USR="proyecto";

static public final String BD_PWD="proyecto";

...

Class.forName("org.postgresql.Driver");

db = DriverManager.getConnection(url,usr,pwd);
```

pagina en blanco
intencionada

Bases de datos utilizadas

Para la gestión y control de los datos necesarios se han creado distintas bases de datos, la funcionalidad y descripción de la mismas se detallan a continuación. Para cada base de datos se describe el propósito de la misma, detallando a continuación los campos que la componen con una explicación de los mismos.

Base de datos "usuarios"

Es la encargada de contener los usuarios válidos del sistema así como la información de los mismos.

Campos que la componen:

- nombre_usu: Contiene el nombre único de usuario válido del sistema.
- clave: La clave de acceso al sistema.
- telefono: Teléfono de contacto.
- direccion: Dirección donde enviar la correspondencia.
- email: Dirección de correo electrónico.

Base de datos "Histórico"

Contiene un relación de la actividad del alquiler de bicicletas. Cada vez que se alquile una bicicleta aparecerán dos entradas, una para la recogida y otra cuando ésta se deje.

Campos que la componen:

- idt: Identifica la transacción, es decir recogida y devolución tendrán el mismo idt. *Este dato es el "idSesion" del momento del alquiler*.
- usuario: Nombre del usuario.
- fecha_coge: Hora y fecha de la acción de alquilar.
- fecha_deja: Hora y fecha de la acción de devolver.
- donde_coge: Número de puesto de la acción de alquilar.

- donde_deja: Número de puesto de la acción de devolver.
- nbici: Número de bicicleta que se alquila o devuelve.

Base de datos "puestos"

Mantiene la información de todos los puestos de alquiler, incluido el número de bicicletas máximo que puede tener.

Campos que la componen:

- num_puesto: Identificador del puesto.
- max_bicicletas: Capacidad máxima de bicicletas del puesto.
- nombre: Nombre del puesto.
- calle: Calle en la que está situado el puesto.
- numero_calle: Número de la calle en la que se ubica el puesto.
- cod_postal: Código postal del puesto.
- Fecha: Fecha en la que se dio de alta dicho puesto.
- Activo: Indica si el puesto está activo o no.
- puestos_cercanos: lista, separada por comas, con los números de puestos cercanos.

Base de datos "bicicletas_puesto"

Contiene la información de las bicicletas que están aparcadas en el puesto en ese momento. Cada registro contendrá una relación "puesto-numero_bicicleta" único.

Campos que la componen:

- num_puesto: Número de puesto (debe estar dado de alta en puestos).
- num_bicicleta: El número de bicicleta que está aparcada.
- fecha_deja: Fecha y hora en la que la bicicleta se dejó en el puesto.

Base de datos "incidencias"

En el caso de producirse y detectarse situaciones anómalas, incongruentes o de inconsistencia de datos relativas al cliente, el sistema grabará una entrada en esta base datos, indicando el problema y almacenando los datos necesarios para el seguimiento de la incidencia.

Campos que la componen:

- fecha: Fecha y hora en la que se produjo la incidencia.
- incidencia: Descripción de la incidencia.
- datos: Recopilación de datos relativos a dicha incidencia.
- solución: Descripción de la solución proporcionada.
- fecha_solucion: Fecha y hora de la solución de la incidencia.

En nuestro caso la única incidencia contemplada es la producida cuando el usuario no tiene sesión abierta pero sí tiene un *e-ticket* guardado en el dispositivo, indicativo de que tiene una bicicleta en alquiler. El problema surge cuando al reconstruir el *e-ticket* se producen incongruencias con la información guardada previamente en la base de datos. Como la información válida es la existente en la base de datos, se utilizará ésta para reconstruir el *e-ticket*, finalizar el alquiler y completar la incidencia con la información recogida del usuario y de ambos *e-ticket*.

pagina en blanco
intencionada

DETALLE DEL PROCEDIMIENTO

La Figura 6 muestra el detalle de funcionamiento de la infraestructura. A continuación, se va a describir detalladamente este funcionamiento.

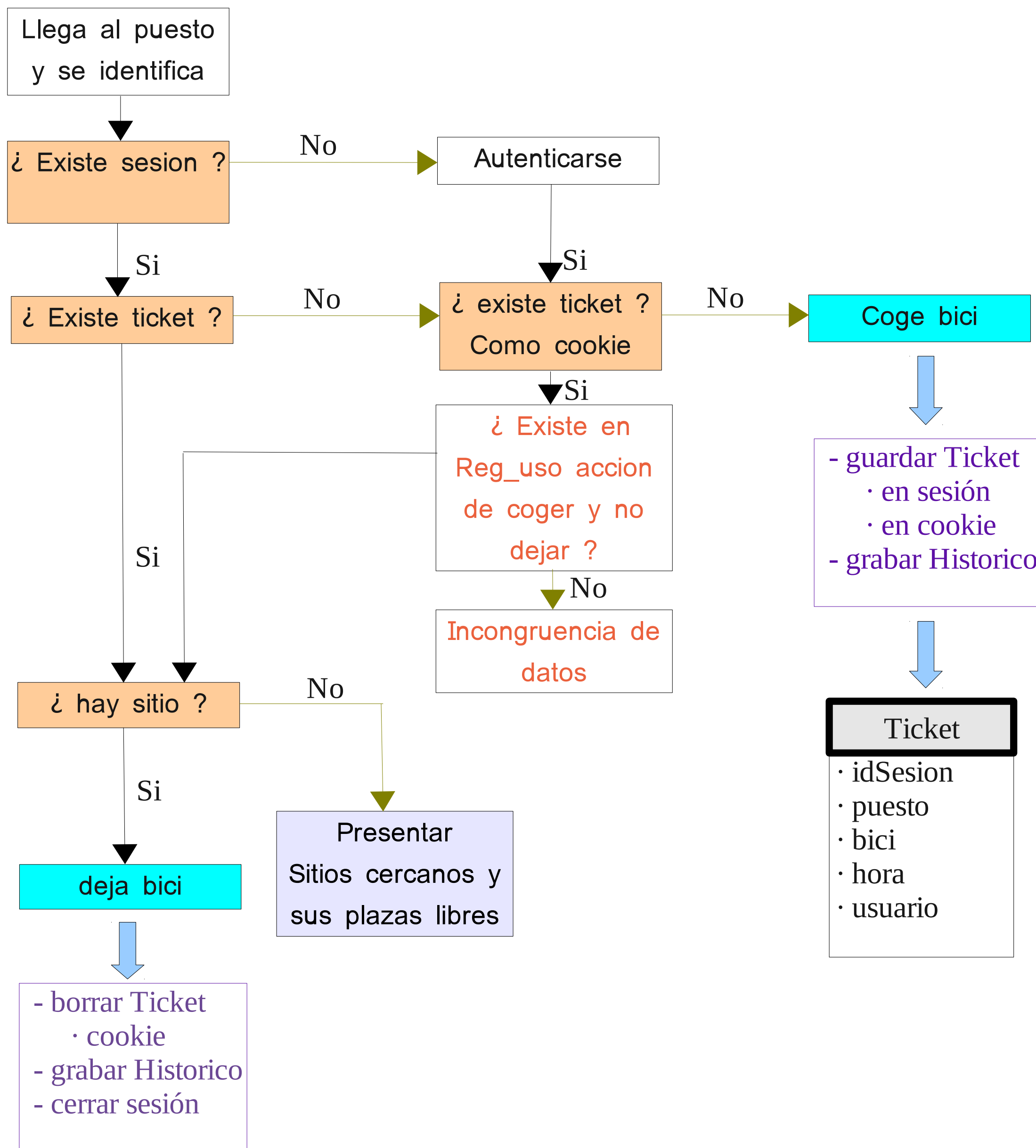

Figura 6 Funcionamiento general de la infraestructura.

Como se puede observar en la Figura 6, el primer paso es el alquiler de la bicicleta previa validación del usuario en el sistema. La Figura 7 muestra una imagen de la pantalla de identificación del usuario. En ella se muestra también el puesto donde se va a realizar la acción (alquiler o devolución). Este dato es obligatorio, ya que la siguiente acción es alquilar o dejar una bicicleta y hace referencia al puesto en el que se realiza la acción.

Figura 7:imagen de la pantalla de identificación del usuario

Si no se pone el puesto, el sistema devuelve esta misma página con una aviso. Se ha tenido en cuenta que si se ha introducido el nombre de usuario, éste no hace falta volver a introducir, ya que el sistema lo presenta automáticamente. Este funcionamiento se repite si se produce un error de autenticación, aunque en este caso sí se presenta una pantalla adicional con un aviso. La Figura 8 presenta dichas pantallas.

Figura 8: Pantallas de error al iniciar la sesión

Después de realizar la autenticación de usuario, tal y como se observa en la figura 6, el sistema comprueba si existe un *e-ticket* como cookie para determinar si el usuario realizó un alquiler en una sesión que ya no existe.

Si existiera tal *e-ticket*, éste se reconstruirá a partir de la información guardada en la base de datos, ya que existirá una entrada de recogida de bicicleta de la que no existe una devolución. Si los datos del *e-ticket* guardado como cookie y el reconstruido no fueran iguales, se generará una línea el la base de datos de "incidencias" para su posterior análisis y solución. El código encargado de dichas comprobaciones se muestra a continuación.

```
(...)
sesion = request.getSession(true);
if ( sesion.isNew() ){
     sesion.setAttribute("usuarioBC", login);
     if (puesto.equals("0000")){
          sesion.setAttribute("PuestoBC", null); // obligara a preguntarlo
     else
           sesion.setAttribute("PuestoBC", puesto);
     GestionLogs.escribirLog(sesion, "Autentificación del usuario "
                        + login);
     consulta.cerrarConexion();
     clsTicket ticket=(clsTicket) sesion.getAttribute("ticketBC"+
                  login);
     clsTicket ticketCookie=LCookie.creaTicketDesdeCookie(sesion,
                  request);
     if ( ticket==null && ticketCookie==null){
      // entro y no tiene ticket luego va a coger bicicleta
           response.sendRedirect("SeleccionarBici");
     }else if (ticket!=null && ticketCookie!=null) {
     // se ha encontrado un ticket cookie de este usuario lo normal es
     //   que se halla perdido la sesion y tenga un bicicleta alquilada
     //   que va a dejar
     ticket=consulta.crearTicketDesdeBD(login);
     if ( ticket!=null){
        if ( ticket.equals(ticketCookie)){
           response.sendRedirect("DejarBici");
        }else{
           String txtIncidencia=" …..
```

```
            String datos="

                  … recopilacion de datos de la incidencia …

            consulta.escribirIncidenciaBD(datos+
                  datosTicketCookie+datosTicket, txtIncidencia);
(...)
```

Después de estas acciones, si el usuario no tuviera ninguna bicicleta alquilada se presentará una pantalla en la que podrá poner el puesto en el que se encuentra (si no lo introdujo en la pantalla de inicio), y seleccionar una de las bicicletas disponibles, o bien la pantalla de confirmación de devolución de la bicicleta.

En ambos casos, si no existieran bicicletas o no existieran plazas libres para dejar la bicicleta, se presentará un listado de los puestos cercanos con el estado de ocupación o el de plazas libres. La Figura 8 muestra un ejemplo de las pantallas que se presentarán en ambos casos.

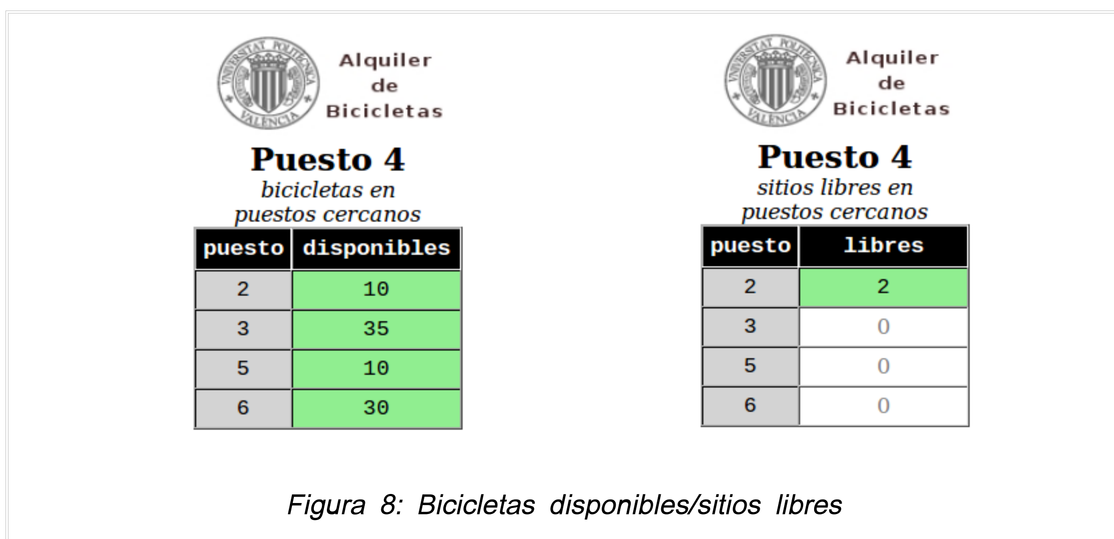

Figura 8: Bicicletas disponibles/sitios libres

Una vez cumplimentada la acción del alquiler, la aplicación devolverá un *e-ticket* con la información necesaria para su control, registrándose esta operación también en el sistema.

La información contenida en el *e-ticket* es la siguiente:

- Usuario: identificador del usuario (único en el sistema).

- Fecha y Hora: Fecha y hora de la creación del *e-ticket.*
- Identificador de bicicleta: Número de bicicleta alquilada.
- Identificador de puesto de alquiler: Número del puesto donde se recogió la bicicleta.
- Identificador de control: Es el "idTicket" y se corresponde con el "idSesion" abierta por el usuario al validarse en el sistema. Este dato no es visible para el usuario.

Transacciones

Denominaremos transacciones de servlets a aquella secuencia lógica, más o menos forzada por la ejecución de los propios servlets, que se ejecutan conjuntamente para poder lograr un único o mismo fin.

La existencia de estas transacciones deriva de la conveniencia de hacer modulares los servlets, y de evitar servlets con páginas muy recargadas que presentan los siguientes problemas:

- más difíciles de programar y de depurar,
- más difíciles de comprobar la existencia y corrección de los datos introducidos o las acciones solicitadas por el usuario,
- genera páginas más recargadas, que posiblemente ya no quepan en una única pantalla, por lo que al usuario le es más difícil de entender/darse cuenta de todos los pasos o datos que se le piden para conseguir su fin,

A continuación se van a detallar las principales transacciones existentes en el proyecto, que son el alquiler de la bicicleta y su devolución. Ambas transacciones parten de la introducción de los datos del usuario y del puesto en el que éste se encuentra.

Alquiler

El alquiler se realiza enlazando tres servlets. El primero es el encargado de presentar las bicicletas disponibles en el puesto seleccionado. El segundo servlet presenta un resumen de la opción seleccionada (podemos volver a seleccionar otra bicicleta simplemente volviendo atrás en el navegador). Y el tercero termina la

transacción con la presentación del resumen de la operación. La Figura 9 muestra los efectos de esta operación.

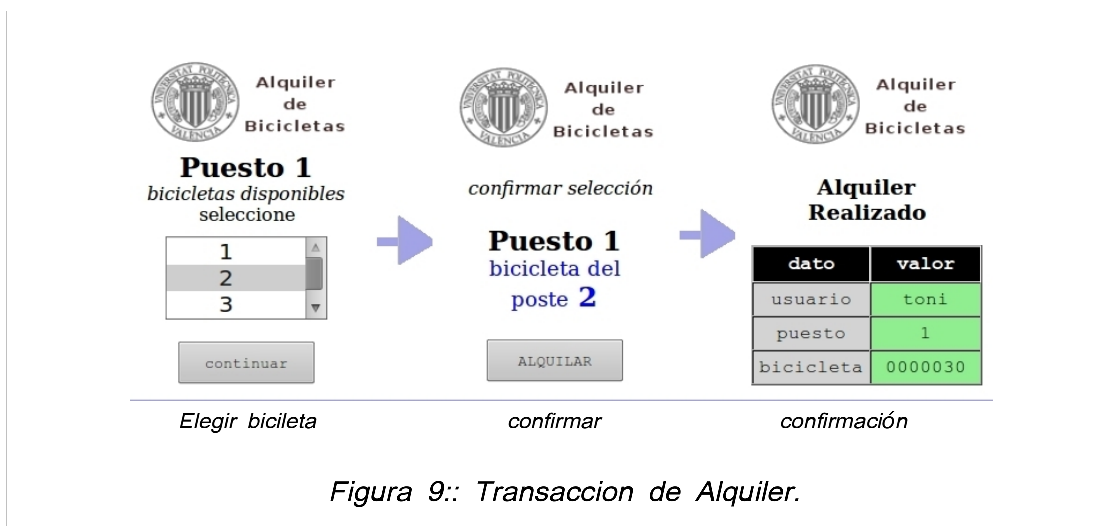

Figura 9:: Transaccion de Alquiler.

Devolución

Recordemos que en el *e-ticket* guardado en el usuario se guarda también un resumen (generado por una función hash) del identificador de sesión, de manera que no se puede saber realmente cuál es dicho identificador salvo que se realice de nuevo ese resumen con el dato original (que está en la base de datos) y se compare.

Así pues, una vez comprobado el identificador de sesión al devolver la bicicleta, el *e-ticket* se borra del teléfono, registrándose esta operación también en el sistema. En este caso, la devolución pone en marcha el proceso de facturación, calculando el precio del alquiler en función del tiempo de préstamo de la bicicleta.

Por otro lado, también se tiene que actualizar la base de datos del puesto receptor de la bicicleta, para reflejar la disponibilidad de ésta, y la base global de usuarios para reflejar la nueva situación del usuario. La Figura 10 muestra dicha transacción de devolución de una bicicleta alquilada.

Figura 10: Transacción de devolución

Recordamos que este tipo de *e-ticket* es temporal, ya que su validez tiene un periodo de tiempo limitado (puede caducar al día siguiente, solo servir por la mañana, etc.). El mismo sistema podría utilizarse para otro tipo de situaciones similares, como pudieran ser la gestión de colas, la reserva de recursos, etc. Es decir, en todas aquellas aplicaciones en las que sea necesario un control del tiempo de entrada de un usuario en el sistema y la salida del sistema de dicho usuario.

También es posible realizar, mediante accesos al registro de incidencias (también llamado histórico), estadísticas de uso del sistema, tanto desde el punto de vista del usuario como desde el punto de vista de la gestión de los recursos disponibles.

Así pues, cuando un usuario se identifica en el puesto, el sistema puede detectar mediante dos procedimientos distintos si éste ya tiene una bicicleta en alquiler:

- Si existe una sesión asociada a dicho usuario y comprobando que existe en dicha sesión un *e-ticket*.
- Si no existe sesión se comprueba si hay una cookie con un *e-ticket*. Este hecho podría indicar que existió un fallo en el servidor mientras una bicicleta está alquilada. Para asegurarlo, se comprueba en la base de datos si existe una bicicleta en alquiler para ese cliente que no tiene devolución.

Una vez realizada dicha comprobación, las acciones a realizar pueden ser dos:

Recoger una bicicleta (dar de alta el alquiler):

- ✔ Crear una sesión y guardar en ella el *e-ticket*.
- ✔ Guardar una cookie del *e-ticket* para evitar la pérdida del mismo por problemas en el servidor.
- ✔ Grabar en la base de datos la acción.

Dejar una bicicleta (dar de baja el alquiler):

- ✔ Grabar en la base de datos la acción.
- ✔ Borrar la cookie con la información del *e-ticket*.
- ✔ Cerrar la sesión.

Si un usuario tuviera en alquiler una bicicleta, se produjera un fallo en el servidor y se perdiera la sesión creada, éste dispone de la información de la misma grabada mediante cookies. Las acciones a realizar en este caso serían:

- ✔ Recuperar el *e-ticket* desde la cookie.
- ✔ Comprobar que dicho usuario tiene una bicicleta en alquiler en la base de datos (posee una entrada de recogida pero no una entrada de devolución).
- ✔ Comprobar que el hash del identificador de la base de datos se corresponde con el guardado en el *e-ticket* recuperado.
- ✔ Cerrar la sesión.

Integridad del e-ticket

Dada la particularidad de la aplicación, un aspecto fundamental es la integridad de los datos guardados. Se han definido dos caminos para proporcionar la integridad del *e-ticket*, ambos en el ámbito del usuario: uno para asegurar que el usuario posee un *e-ticket* en caso de caída del servidor y otro para los datos en sí mismos a través del idTicket contenido en el mismo, ya que este dato es una función hash del original que se comprobará con la información guardada en la base de datos.

Dado que el identificador de la sesión es un identificador único para cada transacción, éste dato se incluye, como se ha mencionado, en el *e-ticket* como identificador del mismo. Esta acción asegura todavía más la unicidad del par "usuario-idTicket", ya que en el caso de la improbable repetición del identificador de sesión, el usuario asegurará la unicidad de la acción. La Figura 11 presenta la cookie del *e-ticket* guardada.

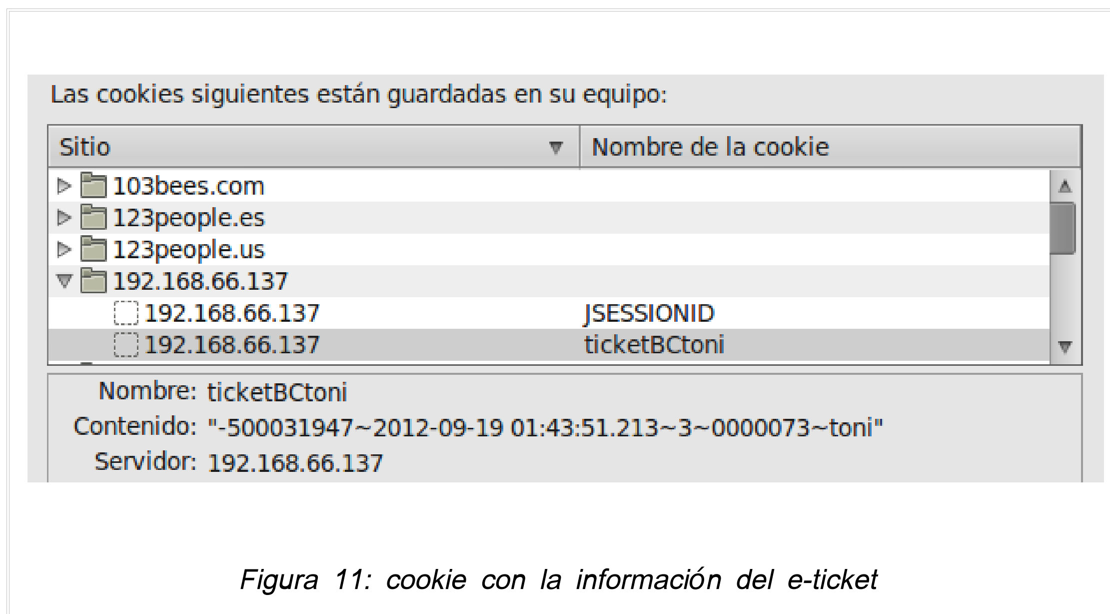

Figura 11: cookie con la información del e-ticket

Mencionar que para el manejo de un *e-ticket* se ha creado la clase clsTicket. Esta clase se detalla posteriormente en el aparatado clases Java especiales, aunque podemos mencionar que mientras los datos de obtención de datos de dicha clase son públicos, como `getUsuario()`, `getidTicket()`, etc... los métodos de grabación son privados, de manera que solo se pueden rellenar esos datos en la inicialización de la clase, evitando así una manipulación de los mismos.

Datos en el cliente

Para evitar la pérdida de datos debidos a problemas con el servidor web (principalmente por la pérdida de la sesión ya iniciada), se guardan los datos como cookie en el cliente. De esta manera, al acceder de nuevo al sistema puede detectarse la existencia del *e-ticket* mediante la información guardada en el cliente, y reconstruirlo con la información que debe estar guardada previamente en el servidor.

A diferencia del *e-ticket* de sesión (al que solo puede acceder el usuario validado), el *e-ticket* guardado como cookie no contiene el dato "idTicket" sino su resumen mediante una función de hash.

Este *e-ticket* solo se consultará, como se ha comentado anteriormente, cuando la sesión se haya perdido, generalmente debido a problemas en el servidor, y más raramente por caducidad de la misma. Esto proporciona la seguridad de que es

improbable que se pueda manipular el *e-ticket* entregado, ya que los datos correctos son los guardados previamente en la base de datos, y solo se puede acceder a ellos mediante el "idTicket", dato que el cliente desconoce.

En el caso de tener que recuperar los datos del *e-ticket* del cliente, el procedimiento sería:

- ✔ Generar un *e-ticket* a través de las cookies del cliente.
- ✔ Comprobar que existe una entrada de alquiler de ese usuario que no tiene asociada una acción de devolución.
- ✔ Contrastar el código hash guardado en el *e-ticket* del cliente con el calculado del *e-ticket* guardado el base de datos que coincide con la situación anterior.
- ✔ Comprobar que todos datos son iguales. Si no lo fueran, y basándose en que la información correcta es la que existe en la base de datos, se generará un informe del error para poder realizar un seguimiento del incidente.

Estructura de los servlets

Todos los servlets que se han programado utilizan un patrón general a la hora de generar la página HTML que se le mostrará al usuario. Se han identificado tres partes que contienen todas las páginas HTML y, por ende, los servlets deben mostrarlas y respetar sus ámbitos, no solapándose o desordenándolos. Así, toda página HTML tiene la siguiente estructura:

Cabecera HTML: lo único que varía de una página a otra es la particularización del título.

Parte central: es particular de cada servlet y lo que los diferencia, tanto por lo que muestran, por lo que ejecutan o por los datos y acciones que solicitan.

Cierre de HTML: común para todos los servlets (básicamente compuesto por las etiquetas `</BODY></HTML>`)

Para aplicar sistemáticamente esta estructura, se ha implementado una clase Java denominada Cadenas, que se estudiará en su propio apartado.

Dependencias externas

En los servlets presentados se importan las siguientes clases:

`import javax.servlet.ServletException`:

para tratar las excepciones de los servlets

`import javax.servlet.http.HttpServlet`:

para gestionar el protocolo de la comunicación

`import javax.servlet.http.HttpServletRequest`:

para recoger la información enviada por el contenedor

`import javax.servlet.http.HttpServletResponse`:

objeto donde se escribe la respuesta para devolvérselas al cliente

`import javax.servlet.http.HttpSession`:

para poder utilizar las sesiones

Sesiones

Las sesiones se utilizan con dos objetivos:

- Primero, para el funcionamiento del control de usuarios registrados. De esta manera solo los usuarios válidos tendrán una sesión.
- Segundo, para el control del *e-ticket*. Dado que el *e-ticket* es temporal, éste estará activo mientras dure la sesión. De esta manera se utilizará el 'idSesion' como identificador único del *e-ticket* en curso. Como veremos posteriormente, éste se guarda también con una cookie en el cliente.

Con este fin se ha utilizado el objeto **`session`** proporcionado por el servlet. Para ello hay que importar **`httpSession`** (el de antes). De esta manera, se puede controlar tanto si el usuario actual está identificado (o no) como el alquiler de la

bicicleta. La sesión se crea y accede con :

```
sesion = request.getSession(true)
```

y al crearse graba una **cookie**, que en este caso tiene este aspecto:

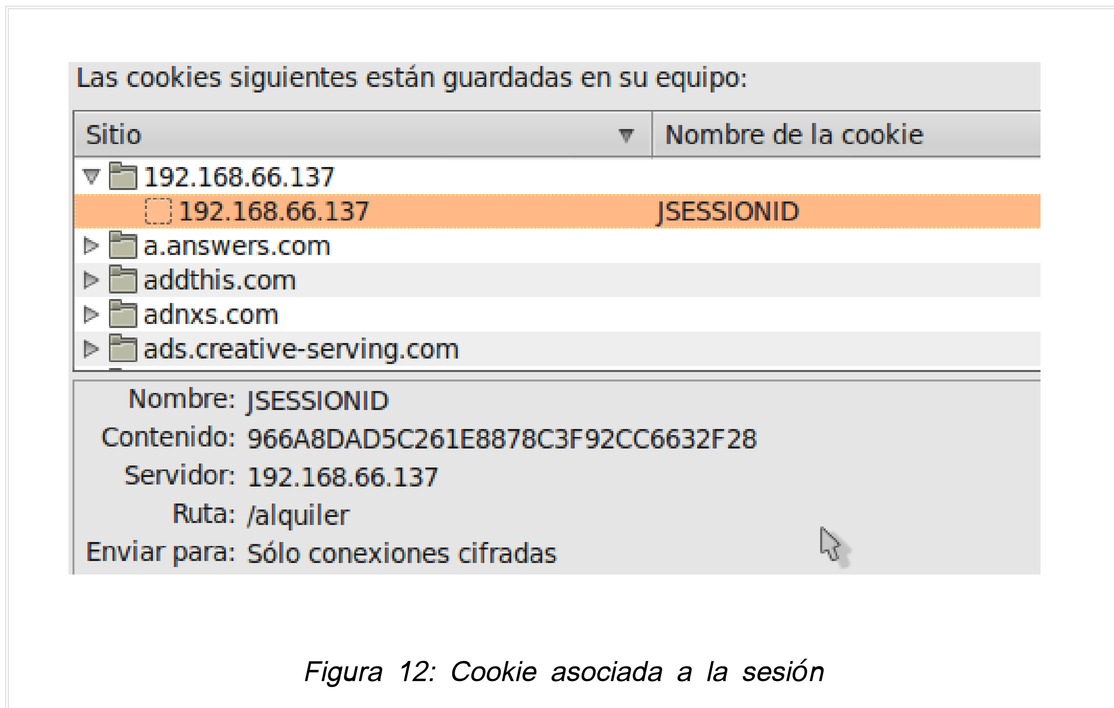

Figura 12: Cookie asociada a la sesión

Clases Java especiales

La clase Cadenas

Para ayudar a construir las páginas tal como se ha indicado en "Estructura de los servlets", se crea una clase Java, denominada Cadenas, que proporciona métodos estáticos que permiten conseguir esta estructura modularmente. Para cada parte se invoca a su respectivo método, que genera automáticamente el mismo código para todos los servlets, con las personalizaciones ya indicadas (título en la cabecera). Evidentemente, la parte central no tiene un método universal para su contenido, pero sí que lo tiene para marcar su inicio y su finalización, al objeto de estandarizar la estructura y formato.

También se ha utilizado para incluir procedimientos estáticos de utilidad para escribir

fechas, horas, etc.

A continuación se describen brevemente los métodos estáticos de la clase Cadenas que sirven para generar la estructura básica HTML de los servlets:

`Cadenas.htmlIni(String nombrePagina, int indiceSeleccion, HttpSession sesion)`

Método padre que invoca a otros, y que genera la parte superior de la página HTML, concretamente, la cabecera HTML y la inserción del logotipo.

Cadenas.htmlLogo(String nombrePagina):

Genera la cabecera (**<head></head>**) del documento HTML, declara el **<div id='principal'>**, y también añade el primer **<div class='titulo'>** que inserta un logotipo de la página HTML.

Recibe como parámetro una cadena, **nombrePagina**, que sirve para mostrar información sobre qué página es cada una en el título, para que, por ejemplo, se muestre esa información en el título de la ventana del navegador Web.

Cadenas.htmlIniDivCentral():

Simplemente declara el tipo **<div>** para la parte programática del servlet.

Cadenas.htmlFinDivCentral():

Similarmente al anterior, termina el **<div>** central, el de la parte programática del servlet.

Cadenas.htmlFin():

El que se invoca al final, para terminar la página HTML.

De esta manera la composición de una página sin información dinámica generada por el servlet se realiza de una manera sencilla y limpia:

```
protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws
                 ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    pagOK  = Cadenas.htmlIni("Inicio", “”, “”,“”);
```

```
    pagOK += Cadenas.htmlIniDivCentral();

          //  … información generada dinamicamente si existe...

    pagOK += Cadenas.htmlFinDivCentral();
    pagOK += Cadenas.htmlFin();
    out.println(pagOK);
    out.close();
}
```

Cadenas.escribeFecha():

Se utiliza para devolver un string con la fecha en formato 'dd mmm yyyy'.

Cadenas.escribeHora():

Similar al anterior, pero para escribir la hora en formato 'H:mm'

La clase LCookies

Se han encapsulado los accesos y grabaciones de cookies en una clase para conseguir una mayor simplicidad y legibilidad del código resultante.

Esta clase tiene también, al igual que en *Cadenas*, sus métodos estáticos. Son los siguientes:

`LCookie.dameArrayCookies(HttpServletRequest request):`

Devuelve el array de cookies guardadas en el cliente.

`LCookie.creaCookie(HttpServletResponse response, String txtClaveCookie,String txtValorCookie)`

Graba una cookie en el cliente a través del objeto de respuesta del servlet.

`LCookie.dameCookie(HttpServletRequest request,String  mCookieBuscar)`

Devuelve la cookie con nombre *mCookieBuscar*.

`Lcookie.creaCookieTicket(clsTicket mTicekt,HttpServletResponse response)`

Graba una cookie con la información de un *e-ticket* con nombre "ticketBC", el

código del mismo se detalla a continuación:

```
        public static void creaCookieTicket(clsTicket mTicekt,
                              HttpServletResponse response){
          String txtCookie=mTicekt.getidSesion()+"~"+
                             mTicekt.getCuando().toString()+"~"+
                             mTicekt.getPuesto()+"~"+
                             mTicekt.getnumBici()+"~"+
                             mTicekt.getUsuario();
          creaCookie(response,"ticketBC",txtCookie);
    }
```

`LCookie.creaTicketDesdeCookie(HttpSession sesion,HttpServletRequest request)`

Crea y devuelve un *e-ticket* con la información guardada previamente en la cookie "ticketBC".

`public static void borraCookie(HttpServletResponse response, String txtClaveCookie,String txtValorCookie)`

Borra la cookie asignado su periodo de validez a -1.

`public static void eliminaCookieTicket(clsTicket mTicekt, HttpServletResponse response){`

Cambia los datos de la cookie guardada poniendo sus datos a cero `"0~0~0~0~0". De esta manera, si la cookie no se borrara, ésta sería identificada como inválida.`

La clase ConexionBD

En esta clase se centralizan todas las acciones que hacen referencia, modifican o consultan la base de datos. Los servlets quedan así descargados de la necesidad de controlar la apertura y cierre de conexiones con la base de datos, o de invocar expresamente las consultas que necesitan.

Esta clase es dinámica, por lo que es preciso crear un objeto **`ConexionBD`** para poder trabajar con la base de datos:

```
(...)
    ConexionBD consulta = new ConexionBD(); // Antes de precisar
                                               utilizar la BD
    (...)
    consulta.cerrarConexion(); // Cuando deje de ser necesario
                                  realizar consultas a la BD
}
```

Al terminar el servlet, debe procurarse siempre invocar al método **cerrarConexion()** de dicho objeto para asegurarse de que no quedan conexiones abiertas huérfanas con la base de datos. Así se evita el peligro de dejar demasiadas conexiones abiertas con el servidor y de llegar al número máximo de conexiones permitidas, pues llegado ese caso, las siguientes conexiones se rechazarían, dejando en la práctica de ser operativa la infraestructura.

Las acciones a realizar por esta clase se dividen en varios grupos:

- Apertura de la base de datos mediante el método **IniBD()**. Esta acción se realiza cuando se crea el nuevo objeto **Consultas**, en su constructor, y debe ser el primero en invocarse.
- Cierre de la conexión con la BD mediante el método **cerrarConexion()**. Este método debe invocarse al finalizar las operaciones con la BD, al final del servlet, típicamente.
- Consultas propiamente dichas (**SELECT**), que devuelven un conjunto de datos (**Resultset**). Preparan la sintaxis de la consulta a realizar e invocan a **laConsulta()**
- Modificaciones (**UPDATE**), borrados (**DELETE**) e inserciones (**INSERT INTO**) sobre la base de datos, que devuelven un booleano indicativo de si la transacción ha sido exitosa (**true**) o si ha fracasado (**false**). Preparan la sintaxis de la consulta a ejecutar e invocan a **laAsignacion()**
- Métodos auxiliares privados que son los que efectivamente ejecutan las consultas SQL preparadas por los métodos anteriores, y que comprueban el resultado de tal operación, devolviendo **true** o **false** según sea éxito o fracaso. Son dos:
    - **laConsulta()**, encargada de consultas que devuelven datos

(**SELECT** …).

  - **laAsignacion()**, que se encarga del resto de operaciones **SQL (UPDATE, INSERT** …).

- Métodos lanzaderas para otras consultas o conjunto de consultas, que no son tan sólo una mera consulta a la base de datos, sino que incorporan una lógica para la realización de una acción compleja, como son el alquiler o devolución de una bicicleta.

Como ejemplo de utilización de la clase ConexionBD.java se muestra parte del servlet encargado de listar los puestos cercanos:

```
(...)
for (int i=0;i<puestosCercanos.length;i++){
 numPuesto    = puestosCercanos[i];
 if (consulta.existePuesto(numPuesto)){
  if ( consulta.existePuestoEnBiciPuestos(numPuesto)){
    int maxBicisPuesto=consulta.dimeNumMaxBicisEnPuesto(numPuesto);
    int numBicisEnPuesto=consulta.numeroBicisOcupadasEnPuesto
                     (numPuesto);
    if (maxBicisPuesto>0){
     if (idAccion.equals("disponibles"))
       pagOK += crearFila(numPuesto,numBicisEnPuesto);
     else
       pagOK += crearFila(numPuesto,maxBicisPuesto-
                     numBicisEnPuesto);
(...)
```

El código de las consultas involucradas en la acción anterior en esta clase son:

```
public int numeroBicisLibresEnPuesto(String puesto) {
  int nLibres=0;
  sentenciaSQL = "SELECT COUNT(num_puesto) FROM bicis_puestos WHERE
     num_puesto='"+puesto+"' AND num_bici='-'";
 try {
```

```
        rs= laConsulta();
        if ( rs.next() )
                nLibres=rs.getInt(1);
        }
        (...)
}


public int numeroBicisOcupadasEnPuesto(String puesto) {
        int nOcupadas=0;
        sentenciaSQL ="SELECT COUNT(num_puesto) FROM bicis_puestos
                WHERE num_puesto='"+puesto+"' AND num_bici<>'-'";
                try {
                        rs= laConsulta();
                        if ( rs.next() )
                                nOcupadas=rs.getInt(1);
        (...)
}
```

La clase GestionLogs

Dentro del marco de operaciones de administración, la clase **`GestionLogs`** se encarga de generar un archivo de texto plano donde se reflejan todas aquellas operaciones que modifican el estado del servidor o de las bases de datos. Dicho archivo se crea por defecto en la carpeta **`/tmp`** de nuestro equipo, teniendo como nombre **`AÑO-MES-DIA.txt`**. En cada entrada generada se denota la fecha y hora de la operación, el usuario que provoca dicha operación, así como una breve descripción de la misma.

A continuación se detallan los puntos más importantes de esta clase.

Constantes

Mediante **`caminoFich`** se especifica el directorio y el nombre del fichero de logs.

```
public static final String caminoFich="/tmp/AÑOMESDIA.txt";
```

Método escribirLogs

El método escribirLogs recibe como argumento la sesión y el texto que deberá imprimir en el fichero.

```
public static void escribirLog(HttpSession sesion, String texto) {
```

La fecha y hora se formatea mediante la clase SimpleDateFormat. Esta clase identifica cada componente de tiempo (día, mes, año, horas y minutos) como un par de letras, teniendo el año el par **YY**, el mes **MM**, el día **DD**, las horas **HH** y minutos **mm**. El formato utilizado es el siguiente:

```
SimpleDateFormat formato = new SimpleDateFormat("yyyyMMdd_HHmm");
```

La clase y el método invocante se captura de esta manera:

```
Throwable t = new Throwable();
StackTraceElement[] elements = t.getStackTrace();
String invocante = elements[1].getClassName() + "." +
elements[1].getMethodName();
```

De la sesión recibida, se obtiene el usuario que realizó la modificación mediante:

```
login = (String) sesion.getAttribute("usuario");
```

Dado que el fichero de logs debe permitir la introducción de nuevas entradas aprovechando el fichero ya existente, la estructura de apertura, escritura y cerrado del mismo es la siguiente:

```
try {
    fichero = new FileWriter(caminoFich,true);
    pw = new PrintWriter(fichero);
    pw.println(instante + "\t" + invocante + "\t" + login  + "\t" +
          texto);
    (...)
```

```
} catch (IOException e) {
        e.printStackTrace();
} finally {
    try {
        if (null != fichero) fichero.close();
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
```

Finalmente, un ejemplo de entrada del registro de sucesos:

```
20120919_0232    OcupacionPuestosCercanos.doPost
     [01314DE9EB48A9793BEF7A60063C3882] toni   Buscando Puestos
cercanos al puesto (33)
```

La clase Ticket

Aprovechando la capacidad de los servlets, esta clase es la encargada de gestionar la información referente a un ticket.

Recordamos que el e-ticket tiene los siguientes datos ;

- Usuario, Fecha y Hora, Identificador de bicicleta, Identificador de puesto de alquiler, Identificador de control.

Mencionar que mientras la información de obtención de datos son públicos, como `getUsuario()`, `getidTicket()`, etc... los métodos de grabación son privados, de manera que solo se puede rellenar esos datos en la inicialización de la clase, evitando así una manipulación del mismo una vez se ha creado,

```
clsTicket T=new clsTicket(idSesion,cuando,puesto,numBici,usuario)
```

También se han reescrito los siguiente métodos heredados para facilitar el manejo del *e-ticket*. A continuación se detallan estos métodos.

`public boolean equals(Object Obj)`

Compara si dos *e-ticket* son iguales. Recordemos que el *e-ticket* guarda como idTicket el hash del original (que está en la base de datos), para evitar

un orden de comparación determinado. El método equals realiza las tres comparaciones posibles. Para dos tickets `T2 y T1`

- Si T2.isTicket==T1.idTicket
- Si T2.idTicket.hash()==T1.idTicket
- Si T1.idTicket.hash()==T2.idTicket

Si se cumple una de estas tres condiciones sus idTicket serán iguales.

```
public boolean equals(Object Obj){
      clsTicket T2=(clsTicket) Obj;
       boolean dev=( T2.getnumBici().equals(this.numBici)
            && T2.getPuesto().equals(this.puesto)
            && T2.getCuando().equals(this.cuando)
            && ( T2.getidTicket().equals(this.idTicket) ||
                   T2.mismoHash(this.idTicket)          ||
                   this.mismoHash(T2.getidTicket()) )
                  );
       return dev;
}
public boolean mismoHash(String HashIdTicket){
      try{
            return ( this.idTicket.hashCode()==
                         Integer.parseInt(HashIdTicket) ) ;
...
```

`public String toString()`

Imprime la información del *e-ticket*.

Filtros

Los filtros son unos servlets especiales que pueden insertarse transparentemente en medio de una cadena de petición-respuesta en el contenedor de tomcat. Tienen la propiedad de que permiten reaprovechar servlets anteriores sin precisar modificación, añadiendo código suplementario que dota a la nueva cadena de ejecución de

características adicionales deseables. Varios filtros pueden concatenarse en la cadena de llamadas, por lo que su diseño puede ser modular: cada filtro añade una nueva capa de funcionalidad propia e independiente de la de los demás. Pueden aparecer tanto a la entrada de un servlet existente, entre el proceso de petición del cliente y su efectiva ejecución, como a la salida del mismo, y previo al envío de respuesta al cliente.

En este proyecto se ha identificado la conveniencia de utilizar filtros, en concreto se ha implementado FiltroAcceso.java :

- FiltroAcceso: Intercepta las peticiones del usuario para la realización del Alquiler/Devolución. Desde la vista del sistema se comprueba si el usuario está autentificado en el sistema, dejándolo continuar si efectivamente lo está. Por el contrario, el filtro redirige la petición a la página de autentificación (inicio de sesión, *servlet* IniciarSesion) si no estaba todavía autentificado.

Para la implementación del filtro denominado FiltroAcceso se pueden seguir dos caminos diferentes, pero convergentes en el mismo resultado: uno manual, y otro utilizando las características propias de Eclipse, el cual, evidentemente, es el que se recomienda para un uso futuro y es el que se explica a continuación.

Situándose en el paquete correspondiente (alquiler) , se procede a crear un filtro mediante el asistente, utilizando para ello el botón derecho del ratón, seleccionando la opción "Create", desmarcando la opción "utilizar uno existente", definiendo el nombre y pulsando "next". A continuación, se realizan las siguientes acciones:

- En "filter Mappings" se añaden las clases donde se aplicará dicho filtro (clic en "next").
- Finalmente, como se desea extender la interfaz "Filter" se cierra el asistente (clic en "Finish").

Una vez generada la nueva clase "NombreDelFiltroIntroducido", destacamos estos dos métodos:

- **`init()`**: Aquí se inicializarán las variables, caso de ser necesarias.
- **`doFilter()`**: Acciones que realizará el filtro, permitiendo su terminación normal o su redirección en caso de que determinemos que no se

cumplen determinadas condiciones.

Un ejemplo de redirección a la página inicial tras intentar una operación no permitida sería:

```
response.sendRedirect("InicioSesion");
```

De forma paralela, por cada filtro y servlet alcanzado, el fichero "**web.xml**" muestra este aspecto:

```
<filter-mapping>
    <filter-name>FiltroDePrueba</filter-name>
    <servlet-name>Buscar</servlet-name>
</filter-mapping>
```

CONFIGURACIÓN DEL FILTROACCESO.JAVA

Como debe invocarse con carácter previo, la configuración de este filtro consiste en indicar que debe filtrar anticipadamente a los siguientes servlets:

- **/SeleccionarBici.java**, **/ConfirmarBiciSeleccionada.java** y **/AlquilarBici,java** en el caso de Alquiler
- **/ConfirmasDevolverBici.java** y **/DevolverBici.java** en el caso de devolución.

A continuación se muestra la parte del fichero **web.xml** encargado de dicha configuración:

```
<filter>
  <display-name>FiltroAcceso</display-name>
  <filter-name>FiltroAcceso</filter-name>
  <filter-class>FiltroAcceso</filter-class>
</filter>
<filter-mapping>
  <filter-name>FiltroAcceso</filter-name>
  <url-pattern>/SeleccionarBici</url-pattern>
  <url-pattern>/ConfirmarBiciSeleccionada</url-pattern>
  <url-pattern>/AlquilerBici</url-pattern>
</filter-mapping>
```

pagina en blanco
intencionada

Referencias

[Fundación Apache]
http://tomcat.apache.org/tomcat-6.0-doc/index.html

[eclipse.org]
www.eclipse.org/documentation/
http://www.eclipse.org/forums/

[postgresSQL]
http://wiki.postgresql.org/wiki/Main_Page

[otros]
http://memex.dsic.upv.es/pbs/Documentacion/
www.forosdelweb.com

pagina en blanco
intencionada

PALABRAS CLAVE

apache tomcat, cookie, dispositivo móvil, eclipse, java, nfc, postgresql, servltes, sesión, servidor web