Capítulo 5

Desarrollo de la aplicación

> A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.
>
> Douglas Adams

Nuestra aplicación se encuentra estructurada en tres niveles distintos:

- El primer nivel es la interfaz. Las funciones que se realizan en este nivel se limitan únicamente a enviar las peticiones/consultas que genera el usuario, obtener los datos generados por la aplicación web (servidor) y finalmente representar los datos en el navegador.

- En el segundo nivel situaríamos la lógica. En este nivel se encuentra el núcleo principal de la aplicación y es el encargado de dotar a la aplicación web del contenido dinámico. También es el responsable de la comunicación con los elementos de las subredes de prácticas.

- En el tercer nivel estarían los datos. Éste está formado por una base de datos que hemos creado anteriormente y que contiene toda la información relativa a los usuarios, turnos, reservas y demás datos empleados por nuestro sistema.

A continuación explicaremos el diseño e implementación de los niveles aun no definidos: la interfaz de usuario y el núcleo de la aplicación.

5.1. La interfaz de usuario

Podemos ver la interfaz de usuario como la capa lógica encargada de dar soporte al diálogo con el usuario. Sus funciones son básicamente dos:

- **Entrada:** adquirir las órdenes, información y comandos expresados por el usuario a través de diversos dispositivos de interacción.
- **Salida:** presentar resultados, retroalimentación y cooperar para facilitar al usuario la realización de las tareas que pretende resolver el sistema.

Deberemos pues analizar con detalle cuál es la información a la que el usuario necesitará tener acceso en cada momento y cuáles son todas las acciones que deberá poder realizar a través de nuestra aplicación.

Una vez establecido el qué se debe mostrar habrá que decidir el cómo, ésto es, el aspecto visual de la interfaz. Se ha tomado como premisa básica la sencillez. El diseño deberá ser comprensible, amigable, claro, intuitivo y de fácil aprendizaje para el usuario.

No hay que olvidar por otro lado que las interfaces web tienen ciertas limitaciones en las funcionalidades que ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio que no están soportadas por las tecnologías web estándares. Seremos capaces de suplir estas carencias mediante el empleo de scripts en el lado del cliente. Además el uso de AJAX nos permitirá ofrecer una experiencia interactiva que no requiera recargar la página cada vez que el usuario lleve a cabo una acción.

5.1.1. Funcionalidad

Será indispensable que el usuario conozca la estructura de la subred de prácticas con la que se encuentre trabajando, ésto es, todos y cada uno de los elementos que conforman la red (PCs y conmutadores) y las conexiones entre los mismos. Para tal fin se ha optado por la creación de una representación gráfica de dicha subred en formato SVG (Scalable Vector Graphics).

El por qué del empleo de este formato se debe a que pretendemos que el usuario pueda interactuar con esta imagen, pudiendo así por ejemplo seleccionar el elemento a configurar haciendo click directamente sobre el mismo o ver su configuración actual al pasar el ratón sobre él. Combinando el uso del formato SVG con el condigo JavaScript en el cliente podemos lograr esta interacción.

Si bien el uso de un mapa de imagen se presentó como una alternativa más que viable a la opción finalmente elegida, en el caso de futuras modificaciones en la estructura de la red de prácticas, es mucho más sencillo la creación de una nueva imagen SVG que la de crear un nuevo mapa de imagen (ver apéndice A.2).

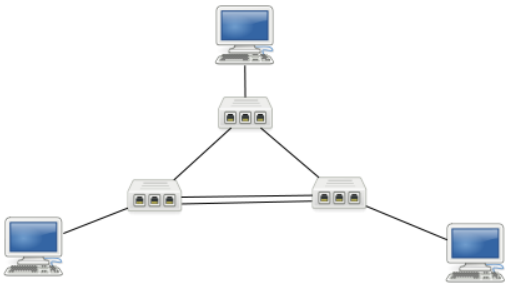

Figura 5.1: Esquema de la subred de prácticas

El usuario no solo necesita conocer cómo es la red con la que está trabajando, sino que también deberá poder ver en todo momento la configuración de cada uno de los elementos que la componen. Para disponer de toda esta información en el lado del cliente haremos uso de las cookies. Éstas serán creadas y modificadas según proceda en el servidor y enviadas al navegador del usuario. En el lado del cliente los script harán uso de las mismas según lo requieran.

Con toda la información que necesita ya en su poder, solo resta ofrecer al usuario los mecanismos necesarios que le permitan actuar sobre la red. Lo primero será establecer que acciones se le permitirá al usuario llevar a cabo. Éstas son:

- **Configurar un PC:** ésto incluye única y exclusivamente los siguientes parámetros:
  - Dirección IP.
  - Máscara de red.
  - Puerta de enlace predeterminada.
  - Servidores DNS.
- **Configurar un conmutador:** esta acción se llevará a cabo mediante el envío al mismo de un archivo de configuración que el usuario deberá de crear previamente de acuerdo al manual del modelo de conmutador del que se trate.

- **Administrar configuraciones:** cada usuario podrá almacenar las configuraciones que haya creado para la red en el servidor. Posteriormente dichas configuraciones podrán ser cargadas de nuevo en la red, alteradas o borradas a petición de éste.

- **Ping a una dirección IP:** el uso del comando ping permitirá al usuario el probar el correcto funcionamiento de la red tras haber modificado su configuración.

El tratamiento de cada una de estas acciones por parte del servidor se verá mas adelante (ver apartado 5.2). En el lado del cliente todo lo que deberemos hacer es enviar toda los datos introducidos por el usuario y necesarios para llevar a cabo cada una de las mismas y, una vez recibida la respuesta del servidor, mostrar el resultado por pantalla. Para ello se emplearán conjuntamente una serie de formularios donde introducir los datos requeridos para cada una de las posibles peticiones y un conjunto de scripts que envíen dicho datos y traten con la respuesta recibida.

5.1.2. Aspecto visual

A la hora de diseñar el aspecto visual de la interfaz, hay que tener claro todo lo que es necesario incluir en ella:

- El esquema de la subred de prácticas.

- La configuración actual de los elementos de la red.

- Los formularios de entrada de datos.

- El resultado de las peticiones al servidor.

- Botones/Menús para seleccionar la acción a llevar a cabo.

Teniendo esta lista presente se ha diseñado la interfaz mostrada en la imagen 5.2. En ella podemos distinguir las siguientes secciones:

- **Menú de navegación:** a través del cual se tiene acceso a todas las acciones que el usuario puede llevar a cabo a excepción de la configuración de los PCs y conmutadores. Ésto incluye: iniciar/cerrar la aplicación, guardar/cargar/administrar las configuraciones del usuario y el comando ping.

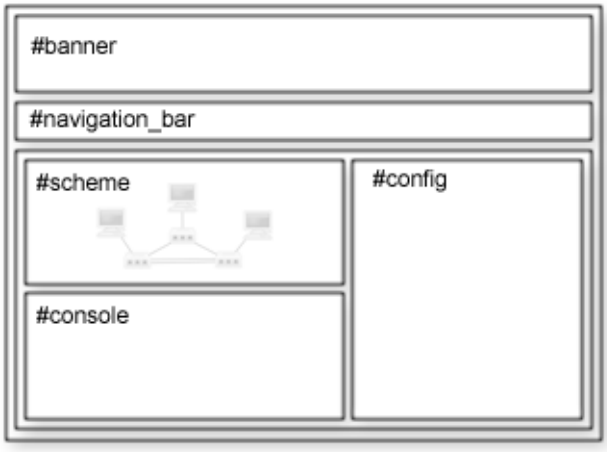

Figura 5.2: Boceto de la interfaz de usuario

- **Esquema de la red:** además de mostrar la estructura de la red, y mediante el empleo de scripts, podremos acceder a la configuración actual de cada uno de los nodos de la red con solo pasar el ratón por encima de ellos o cargar el formulario correspondiente para la modificación de su configuración con un click en la imagen.

- **Consola:** aquí es donde se muestran los resultados de todas y cada una de las acciones que lleve a cabo el usuario. Dispone de un botón que permite reiniciar la consola para evitar la acumulación de datos antiguos.

- **Formularios:** cada vez que se seleccione un elemento de la red o el menú de navegación, el formulario correspondiente se cargará en el panel derecho.

5.1.3. Scripts para el control de la interfaz

Como se ha podido observar, todo el peso del funcionamiento de la interfaz recae sobre un conjunto de scripts, los cuales tendrán que controlar tanto el dialogo con el usuario como con el servidor. El código responsable de todo ello lo encontraremos en el archivo `interfaz.js`. En éste se encuentra definida la clase `aplicación` la cual incluye los siguientes métodos:

- `gui()`: se llama al iniciar la aplicación y es el encargado de inicializar la interfaz llevando a cabo tareas tales como enlazar diferentes acciones a distintos elementos de la interfaz o arrancar el reloj que muestra el tiempo restante

hasta la finalización del turno. Esta última función hará uso del método `time()`.

- `listarConfiguraciones()`: es usada cada vez que se necesite obtener la información almacenada en las cookies acerca de las configuraciones creadas por el usuario.
- `mostrarConfiguracion()`: encargado de reiniciar los mensajes que contienen la configuración de cada uno de los nodos de la red. Se llama a este método al iniciar la aplicación y cada vez que es modificada alguna de estas configuraciones.
- `mostrarFormulario()`: cargará en el panel derecho el formulario correspondiente a la acción indicada.
- `mostrarErrores()`: en caso de que la respuesta del servidor indique la existencia de algún error ya sea a causa de la validación de los campos de un formulario, errores en la propia aplicación o una excepción grave, este método es llamado para mostrar estos errores según corresponda.
- `start()`, `end()`, `setPC()`, `setConmutador()`, `ping()` y `configs()`: todos estos métodos se encargan de realizar la petición al servidor (mediante AJAX) para cada una de las distintas acciones que el usuario puede llevar a cabo, y de tratar según proceda con la respuesta recibida.

5.2. El núcleo de la aplicación

Toda la lógica de la aplicación se encuentra básicamente en el código escrito en estos dos archivos: `aplicacion.php` y `red.php`. Si bien estos hacen uso de una serie de clases y métodos definidos en otros archivos distintos y que veremos más adelante.

Todas las peticiones que un usuario lleva a cabo desde la interfaz de la aplicación son tratada por el código en `aplicacion.php`. Tras comprobar siempre primero que el usuario tenga acceso a la aplicación, se cargan si existen los datos almacenados en la variable `$_SESSION['red']`. En esta variable se almacena una instancia de la clase `Red` creada al iniciar la aplicación y que se borrará al finalizar ésta. La clase `Red`, como ya veremos, contiene toda la información sobre la red necesaria para la aplicación.

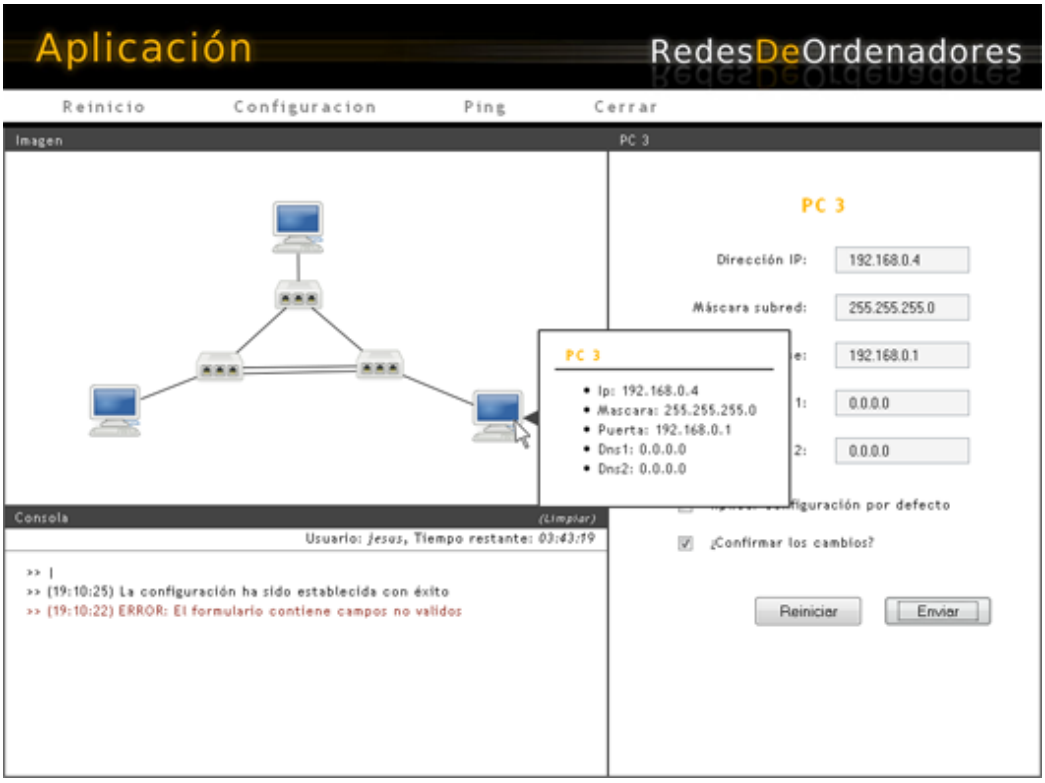

Figura 5.3: Interfaz de la aplicación en funcionamiento

Una vez cargado el contenido de esta variable, un `switch` será el encargado de ejecutar el código correspondiente a la acción requerida por el usuario. En todos los casos el proceso es básicamente el mismo: se validan los datos enviados por el cliente, si no hay problema se ejecuta la acción requerida y finalmente se crea la respuesta a devolver al usuario. Es aquí también donde se crean las cookies enviadas al navegador del usuario con todo la información necesitada por el cliente de la aplicación.

Son las clases y métodos definidos en `red.php` los que realmente llevan a cabo la acción y no el código en el archivo `aplicacion.php`, el cual se limita a llamar a estos métodos.

También necesita una especial mención el archivo `configuracion.php`, el cual contiene todos los datos necesarios para el correcto funcionamiento de la aplicación. Ésto incluye datos tales como los datos para la conexión con la base de datos o toda la información acerca de la red de administración o la estructura de las subredes de práctica. Se trata pues de una archivo extenso que si bien requerirá de tiempo y

atención a la hora de modificarlo, su estructura hace posible que se puedan llevar a cabo multitud de cambios en la estructura de la red de prácticas sin que ello afecte al correcto funcionamiento de la aplicación. Para mas información acerca de este archivo y el cómo modificarlo ver el apéndice A.

5.2.1. Las clases `Red`, `PC` y `Conmutador`

Dentro del archivo `red.php` podemos encontrar la definición de las clases `ElementoRed`, de la cual heredaran las clases `PC` y `Conmutador`, y la clase `Red`.

La clase `ElementoRed` es una clase abstracta que define los métodos y las propiedades de los elementos de la red (PC y conmutadores). En esta clase se definen las propiedades que comparten estos elementos: la id del elemento, su dirección IP dentro de la red de administración así como su configuración actual y la última configuración cargada con éxito. Los métodos principales de esta clase son los siguientes:

- `sendConfig()`: es el encargado de comunicarse con el elemento de la red y enviarle la nueva configuración. La forma de comunicación empleada es distinta en función de si estamos tratando con un PC o un conmutador. Las clases correspondiente deberán sobrescribir este método según sus necesidades. Más adelante se ve con detalle como se comunica la aplicación con cada uno de estos elementos.

- `loadConfig()`: obtiene la configuración indicada de la base de datos.

- `storeConfig()`: almacena la configuración del elemento en la base de datos. En el caso de un conmutador, también se almacena su archivo de configuración en un directorio dentro del servidor creado para tal efecto.

Las clases `PC` y `Conmutador`, añaden otras propiedades y métodos a las definidas en esta clase. Así por ejemplo, la clase `PC` define el método `ping()` encargado de enviar al PC la petición de ejecutar el comando del mismo nombre; y la clase `Conmutador` define el método `generateConfig()` cuya función es la de generar el archivo de configuración a enviar el conmutador combinando la configuración enviada por el usuario con la configuración de administración correspondiente (aquella encargada de crear la red de administración en la red de prácticas).

La clase `Red` contendrá los objetos que representan los distintos elementos que conforman la red y los métodos para tratar con ellos de manera conjunta. También

posee los métodos necesarios para guardar, cargar o borrar las configuraciones creadas por el usuario. Cuando se inicia la aplicación se crea una instancia de esta clase y mediante el método `add` se almacenan en las variables `pc` y `conmutador` las instancias de cada uno de los elementos de la red. El objeto `Red` contendrá por tanto toda la información acerca de la red, tanto los datos para la conexión con los distintos nodos de la misma como la configuración actual de cada uno de ellos. Este objeto se almacena como una variable de sesión y se recupera cada vez que se ejecute una nueva acción sobre la red. Esta variable se borra al cerrar la aplicación.

5.2.2. Comunicación con los PCs: XML-RPC

Para la comunicación entre el servidor y los PCs que conforman las distintas subredes de prácticas se optado por el uso de un sistema cliente-servidor basado en XML-RPC[1]. La razón principal para optar por este mecanismo y no otro es la ya existencia de un proyecto para la creación de un sistema de procedimientos remotos basado en éste mismo protocolo y escrito en Python del autor Silvio Fernández Marín. Incorporaremos el servidor creado para este proyecto en nuestro propio sistema. Al tratarse de un servidor modular solo tendremos que crear una nueva clase en el servidor que lleve a cabo en el PC las acciones requeridas en nuestro caso.

Tendremos que crear pues una clase que contenga los métodos necesarios para: modificar la dirección IP, máscara de subred y puerta de enlace de una interfaz de red, cambiar los servidores DNS y ejecutar el comando ping. Para ello se ha hecho uso de una función definida en el servidor que nos permite la ejecución de comandos del shell del sistema. Haciendo uso de esta función nuestro trabajo se limitará básicamente a establecer el listado de comandos que nos permita llevar a cabo nuestra acción. Éstos son:

```
# Configurar una nueva red
ifconfig eth0 <IP> netmask <Mascara>

# Establecer una nueva puerta de enlace por defecto
route del default dev eth0
route add default gw <Puerta> dev eth0
```

[1] XML-RPC es un protocolo de llamada remota a procedimientos que funciona mediante intercambio de mensajes entre cliente y servidor, utilizando el protocolo HTTP para el transporte de los mensajes. XML-RPC utiliza peticiones POST de HTTP para enviar un mensaje, en formato XML, señalando el procedimiento que se va a ejecutar en el servidor y los parámetros necesarios. El servidor devuelve el resultado en formato XML.

```
# Establecer dos nuevos servidores DNS
cat /dev/null > /etc/resolv.conf
echo "nameserver <DNS1>" >> /etc/resolv.conf
echo "nameserver <DNS2>" >> /etc/resolv.conf

# Ejecutar el comando ping
ping -c 5 <IP Destino>
```

Con este listado de comandos y la función antes mencionada, tendremos todo lo necesario para crear nuestro propio modulo para servidor XML-RPC. La definición del mismo se encuentra en el archivo `configeth.py`.

5.2.3. Comunicación con los conmutadores: Telnet

Para la comunicación con los conmutadores haremos uso de la clase `PHPTelnet` escrita por Antone Roundy. Esta clase permitirá automatizar en PHP una sesión telnet con el conmutador a través de la cual podremos ejecutar el comando encargado de copiar un nuevo archivo de configuración al conmutador:

```
copy tftp startup-config <IP servidor TFTP> conm_config.txt
```

El archivo enviado al conmutador (`conm_config.txt`) es generado por la función `generateConfig()`. Éste se crea a partir de la configuración enviada por el el usuario y la configuración de administración para ese conmutador. Una vez creado, se copia en la carpeta de inicio del servidor TFTP.

En el caso de que el archivo contenga errores, el conmutador devolverá un error en lugar de reiniciarse. El código analizará entonces el log del conmutador con el objetivo de poder mostrar al usuario el listado de los errores detectados por el conmutador en el archivo de configuración.

5.2.4. Otras clases y métodos empleados

Junto a las clases ya comentadas, se han definido otra serie de clases y métodos para ser usados por la aplicación. Todos ellos se encuentran en el archivo `func_comunes.php`.

Una de estas clases es `Validar`, usada para validar todos los datos enviados por el usuario. De ésta cabe destacar los métodos `esAdministracion()` y `esConmutador()`. El primero de ellos es usado para comprobar que no se emplee el uso del rango de direcciones pertenecientes a la red de administración. El segundo es usado para verificar que en los archivos de configuración enviados por el usuario

se emplean solo los comandos permitidos y no se modifica la configuración de la vlan de administración ni la de las interfaces que pertenecen a la misma. La función principal de estos dos métodos es por tanto la de lograr mantener separadas la red de administración de las subredes de prácticas independientemente de las acciones que los usuarios lleven a cabo sobre estas últimas.

También se han definido un conjunto de métodos estáticos dentro una nueva clase llamada `Archivo`. El objetivo de estos métodos es el de ampliar la funcionalidad de las funciones de manejo de archivos de PHP, dándoles la capacidad de lanzar una excepción del tipo `FileException` en caso de error.

Además de la ya mencionada `FileException`, se han creado también dos tipos nuevos de excepciones: `DBException` y `NetworkException`. Éstas están relacionadas con los errores en la conexión con la base de datos y con los elementos de la red respectivamente. Las tres excepciones heredan de `LoggedException` la cual registra la excepción en el log cuando es instanciada.

Por último se ha creado la clase `LogFile`, empleada para el registro de eventos en el archivo de log. Ésta ha sido creada siguiendo el patrón singleton y hace uso de la función de PHP `flock()` para evitar los conflictos de lectura/escritura.