Ejercicios de visor web

Índice

1. Integración de contenido web

Vamos a crear una aplicación que integre contenido web. Deberemos decidir, en función del contenido con el que contamos, la forma más adecuada de integrarlo. Partiremos de la plantilla `TitulosEPS`, que nos dará información sobre las titulaciones ofrecidas por la politécnica. En la aplicación tenemos varias pestañas, una por cada titulación que ofrece dicha escuela. Vamos a mostrar en ellas una serie de visores web con la ficha de cada titulo (en Android en lugar de pestañas tenemos un menú de opciones). Se pide:

a) Las páginas que debemos mostrar son las siguientes:

Estudio	URL
Ingeniería en Informática	`http://cv1.cpd.ua.es/webcvnet/planestudio/planestudiond.aspx?plan=C203`
Ingeniería en Telecomunicaciones	`http://cv1.cpd.ua.es/webcvnet/planestudio/planestudiond.aspx?plan=C201`
Ingeniería Multimedia	`http://cv1.cpd.ua.es/webcvnet/planestudio/planestudiond.aspx?plan=C205`

¿Qué tipo de escalado será más adecuado para mostrar estás páginas en un dispositivo móvil? Implementalo en la aplicación y comprueba que se visualizan correctamente.

Ayuda iOS

Cargaremos la URL (`self.url`) en el visor web en el método `viewDidLoad` del controlador `UAViewController`. Establece ahí el tipo de escalado del visor.

Ayuda Android

Trabajaremos con la actividad `TitulosEPSActivity`. En `onCreate` inicializaremos el visor: estableceremos el tipo de escalado, activaremos Javascript (muy importante para las páginas de la UA), y vincularemos el progreso de carga del visor web con la barra de progreso de la actividad. Tras esta configuración cargaremos la URL de Informática en el visor. En `onOptionsItemSelected` cargaremos en el visor la URL asociada a la opción del menú que se haya pulsado.

b) Tenemos una cuarta pestaña con ayuda sobre la aplicación. En este caso la URL donde tenemos el documento a mostrar no es remota, sino que debemos hacer referencia a un fichero dentro de nuestra aplicación. Debemos acceder al fichero `/www/index.html`, que en Android estará en el directorio de *assets*, y en iOS estará en el raíz del *bundle* principal.

Ayuda iOS

En el método `application: didFinishLaunchingWithOptions:` de

`UAAppDelegate`, asignar a la propiedad `controllerAcerca.url` una URL que dé acceso al recurso local `/www/index.html`.

Ayuda Android

En `TitulosEPSActivity` hay que dar una valor a la constante `URL_ABOUT`, para que haga referencia al recurso `/www/index.html` ubicado en el directorio de *assets* de la aplicación.

Este documento ha sido creado específicamente para mostrarlo como ayuda de nuestra aplicación móvil. ¿Qué escalado resultará más apropiado en este caso? Impleméntalo de forma adecuada y comprueba que funciona correctamente.

2. Temporizador web

Vamos a implementar un temporizador web integrado en nuestra aplicación. En nuestra aplicación nativa tendremos una pantalla que constará de un `WebView` y un botón con el que poner en marcha el temporizador. Mientras el temporizador esté funcionando dicho botón debe quedar deshabilitado, y cuando el temporizador termine volveremos a habilitarlo. Partiremos de la plantilla `TemporizadorWeb`, que contiene la pantalla nativa y el código Javascript del temporizador. Se pide:

a) Hacer que al pulsar el botón nativo se ponga en marcha el temporizador. Para ello, en el evento del botón se deberá llamar a la función Javascript `inicializaTemporizador` de nuestro visor. También haremos que el botón se deshabilite al producirse dicho evento.

Ayuda iOS

En el método `inicializarTemporizador:` del controlador `UAViewController`, llamaremos a la función Javascript `inicializaTemporizador()`.

Ayuda Android

En la actividad `VisorWebActivity`, en el evento del botón llamaremos a la función Javascript `inicializaTemporizador()`.

b) Hacer que el botón vuelva a habilitarse al finalizar el temporizador. Para ello, en la función Javascript `actualizaContador` deberemos notificar a la aplicación nativa que el temporizador ha terminado, y cuando la aplicación nativa reciba dicha notificación deberá volver a habilitar el botón.

Ayuda iOS

Desde la función `actualizaContador` del Javascript enviaremos una notificación de temporizador finalizado a la aplicación, y dicha notificación deberá recibirse en el método `webView: shouldStartLoadWithRequest: navigationType:` del controlador `UAViewController`.

Ayuda Android

En `onCreate` crearemos una interfaz Javascript con un objeto nativo de tipo `JavascriptCallbackInterface`, al que se referenciará en Javascript mediante una variable de nombre `webview`. En la función `actualizaContador` de Javascript llamaremos a la función `temporizadorFinalizado()` de dicha interfaz cuando termine el contador.

3. Alarma con PhoneGap

Con un código Javascript similar al del ejercicio anterior, vamos a implementar una aplicación PhoneGap que nos permita programar un temporizador y que cuando el tiempo llegue a su fin suene la alarma y vibre el dispositivo. Podemos partir de la plantilla `AlarmaPhoneGap`, que en este caso no contiene ningún proyecto Android ni iOS, sino que simplemente contiene un directorio `www` que podremos agregar a cualquier proyecto PhoneGap. Se pide:

a) Crea un proyecto iOS de tipo PhoneGap, e incluye en él el directorio `www`. Es importante que dicho directorio quede incluido como carpeta, no como grupo. Es decir, `www` debe aparecer con un icono de carpeta azul, no amarilla.

b) Implementa en la función Javascript `temporizadorFinalizado` el código PhoneGap necesario para que muestre una alerta en pantalla, suene una alarma, y vibre el dispositivo.

c) Porta la aplicación PhoneGap a Android y comprueba que funciona correctamente.