Especialista Universitario en Computación Móvil y Ubicua

Introducción al desarrollo de web apps

Laboratorio 1.
Diseño de una webapp sencilla: el gestor de tareas

Contenido

1. Introducción

Esta práctica persigue los siguientes objetivos:

1.- Comprender y aplicar los principios que afectan al display de una webapp.
2.- Comprender y aplicar las herramientas que nos proporciona CSS3 para dotar de estilo a nuestras webapps.
3.- Comprender y aplicar las novedades de HTML5 relacionadas con persistencia de datos

En esta práctica vamos a diseñar un sencillo gestor de tareas, poniendo en práctica los conceptos vistos en las clases de teoría. Para ello, primero presentaremos los requisitos de la aplicación. Después diseñaremos las distintas ventanas de la aplicación aplicando estilos adecuados para un entorno móvil. Tras el diseño, añadiremos comportamiento y funcionalidad utilizando JS y jQuery. Por último, se plantearán diversas extensiones.

2. El Gestor de Tareas

El objetivo de esta práctica consiste en diseñar un pequeño gestor de tareas completamente funcional.

El usuario será capaz de crear tareas. Las tareas poseerán las siguientes propiedades:

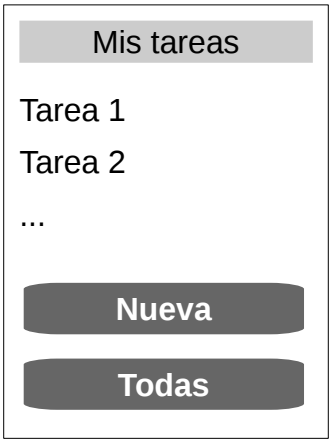
- Título: texto que representa la tarea
- Fecha de creación
- Estado: pendiente o completada

Se diseñará una aplicación multipágina que contemple un mínimo de cuatro secciones:

- La ventana principal.
- La ventana de nueva tarea.
- La ventana de todas las tareas.
- La ventana de editar tarea.

2.1 La ventana principal

Esta ventana mostrará un listado de tareas pendientes, con un máximo de elementos. Las tareas se mostrarán por orden cronológico (primero la más antigua). Cuando se presione sobre una tarea se abrirá la ventana de detalles de tarea mostrando la información sobre la tarea presionada. Asimismo, esta ventana mostrará un par de botones que dará paso a la ventana de nueva tarea y a la ventana de listado de tareas. A continuación se muestra un boceto de esta ventana:

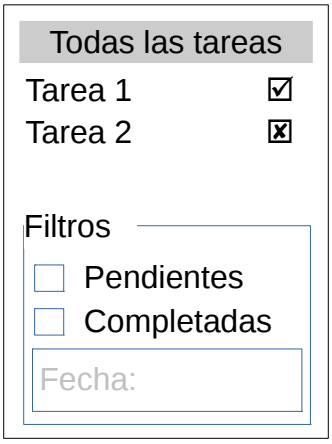


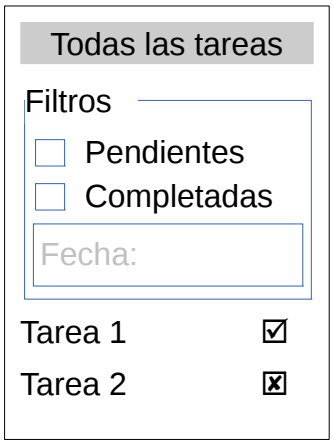
2.2 La ventana de nueva tarea

En esta ventana el usuario podrá introducir toda la información relevante a una tarea. En esta primera versión del Gestor de Tareas, incluirá únicamente el título, pues la fecha se obtendrá de manera automática. El estado de toda tarea nueva pasa a pendiente. El usuario podrá aceptar o cancelar la operación. A continuación se muestra un boceto de esta ventana:

2.3 La ventana de todas las tareas

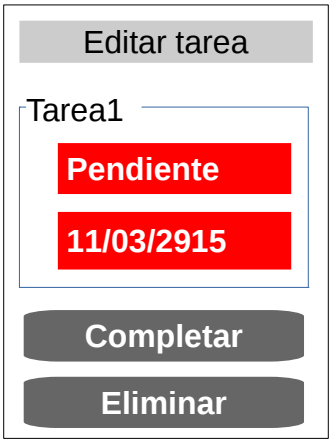
Aquí será posible acceder a todas las tareas que se introdujeron en la aplicación, por orden cronológico (primero la más nueva), y será posible filtrarlas por estado o por fecha. Para mejor identificación, las tareas con distinto estado serán visualizadas de manera diferente. Al presionar sobre una tarea se mostrará la ventana de detalles de tarea mostrando información sobre la tarea seleccionada. A continuación se muestra un posible boceto de esta ventana:





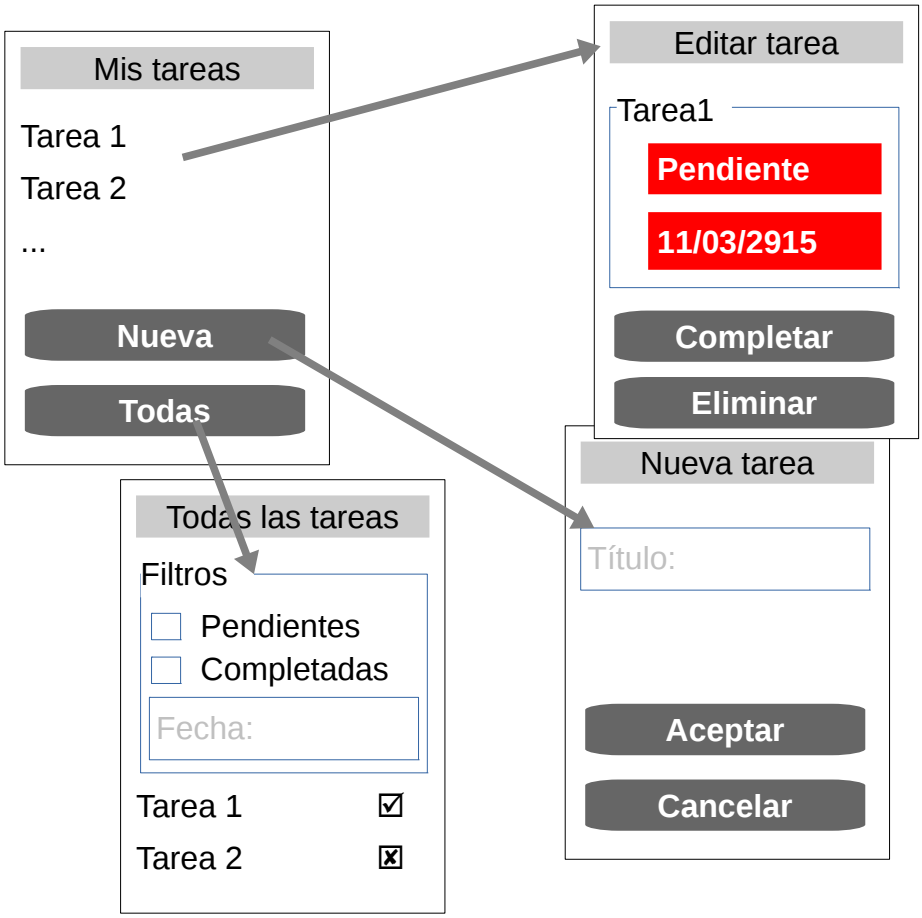
2.4 La ventana de editar tarea

En esta ventana el usuario podrá visualizar toda la información relevante sobre la tarea, así como efectuar acciones. Respecto a la información mostrada, deberá de aparecer el título de la tarea, su fecha de introducción, y su estado actual. Respecto a las acciones, el usuario podrá completar la tarea o eliminarla. A continuación se muestra un posible boceto de esta ventana:



2.5 Arquitectura global

A continuación se muestra la arquitectura global de la aplicación.

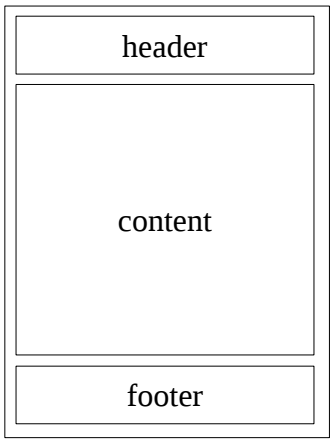


3. Diseño de la aplicación

En esta sección procederemos a diseñar las distintas ventanas de nuestra aplicación, trabajando de manera intensa con los estilos CSS. En esta primera aproximación diseñaremos cada ventana en un documento HTML independiente y definiremos una maqueta con unos datos de ejemplo. Todas partirán de una misma plantilla. En los diseños que se proponen se sigue una estrategia sobria, con poco color, dejando al alumno aventajado personalizar a su gusto.

3.1 Plantilla

Para mantener consistencia, todas las ventanas de la aplicación tendrán una apariencia similar. Diferenciaremos tres secciones principales, tal y como se muestra a continuación:



Sobre el diseño, tendremos en cuenta las siguientes cuestiones:

1. Todas las secciones deben de adaptarse automáticamente al tamaño de pantalla. Esto nos obliga a trabajar en la medida de lo posible con valores relativos (%), en lugar de absolutos (px).

2. La cabecera (header) y pie de página (footer) estarán visibles en todo momento, a modo de “barras de herramientas”.

3. La cabecera poseerá únicamente un título <h1> y, en caso de resultar necesario, un enlace <a> que permita volver atrás alineado a la izquierda.

4. El pie de página puede contener un título <h1> y cualquier otro contenido, que nosotros utilizaremos para ubicar todas las acciones a realizar en la página actual (botones, controles, etc.). Como el pie de página siempre está visible, el usuario siempre podrá efectuar todas las acciones disponibles.

Diseñaremos un esqueleto básico template.html que contendrá esta estructura básica, y que cargará una hoja de estilos styles.css que habilitará esta configuración. A continuación mostramos una primera versión de template.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link type="text/css" rel="stylesheet" href="styles.css">
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
</script>
</head>
<body>
  <div class="page">
      <div class="header"><a href="#" >Atr&aacute;s</a><h1>Header</h1></div>
      <div class="content">Content</div>
      <div class="footer"><h1>Footer</h1></div>
  </div>
</body>
</html>
```

En esta plantilla ya hemos incluido <meta name=”viewport”> para su correcta visualización en dispositivos móviles. A continuación debemos añadir los estilos necesarios para que la cabecera y el pie de página estén permanentemente visibles, además de adecuarlos para una visualización “más contemporánea”. Para mantenerlos visibles en todo momento es necesario utilizar la estrategia de posicionamiento CSS fixed.

```
body {
      margin: 0px;
      font-family: Arial;
}

.page {
      margin: 0 6px 0 6px;
      height: 100%;
}

.header {
      position: fixed;
```

```
      top: 0;
      left: 0;
      height: 40px;
      width: 100%;
}

.header a {
      position: absolute;
      top: 0;
      left: 0;
      padding: 10px 8px;
      color: white;
      text-align: left;
      font-weight: bold;
      text-decoration: none;
}

.header a:hover {
      text-decoration: underline;
}

.header h1 {
      background: -webkit-gradient(linear, left top, left bottom, color-
stop(0.0, #666666), color-stop(0.5, #3A3A3A), color-stop(0.5, #222222), color-
stop(1.0, #000000));
      -webkit-box-shadow: 0 2px 1px #AAAAAA;
      -webkit-border-bottom-left-radius: 6px;
      -webkit-border-bottom-right-radius: 6px;
      font-size: 1.1em;
      color: white;
      padding: 10px 8px;
      text-align: center;
      margin: 0;
}

.content {
      margin-top: 50px;
}

.footer {
      position: fixed;
      background-color: white;
      bottom: 0;
      left: 0;
      width: 100%;
}

.footer h1 {
      background: -webkit-gradient(linear, left top, left bottom, color-
stop(0.0, #666666), color-stop(0.5, #3A3A3A), color-stop(0.5, #222222), color-
stop(1.0, #000000));
      -webkit-box-shadow: 0 2px 1px #AAAAAA;
      -webkit-border-bottom-left-radius: 6px;
      -webkit-border-bottom-right-radius: 6px;
      font-size: 1.1em;
      color: white;
      padding: 10px 8px;
      text-align: center;
}
```

Con estos estilos podemos obtener un resultado similar al siguiente, lo cual promete ser un buen punto de partida:

El problema de utilizar la estrategia de posicionamiento fixed con las barras de herramientas (cabecera y pie de página) es que salen del flujo de posicionamiento normal (static) y por tanto cualquier contenido con posicionamiento normal (en content) quedará tapado por las barras de herramientas. Para evitar esto, daremos una altura fija a la cabecera (40px) y dejaremos en blanco la parte que queda detrás del contenido (margin-top: 50px). Esta limitación no nos afectará ya que como hemos comentado anteriormente en la cabecera sólo vamos a ubicar títulos. El problema lo tendremos con el pie de página: no podemos utilizar una altura fija que funcione en todas las ventanas, pues en cada ventana ubicaremos unos controles diferentes. Por ello, necesitamos calcular la altura del pie de página dinámicamente, con jQuery, y después añadir ese mismo espacio en blanco al final del contenido, para que el pie de página no tape el contenido.
Para ello, tras cargar la librería jQuery, añadiremos el siguiente script a template.html:

```
<script>
$(function() {
    $('.content').height($('.content').height() + $('.footer').height());
    window.scrollTo(0, 1);
});
</script>
```

En el script incrementamos la altura de content tantos pixels como altura posea el pie de página, y así nos aseguramos que el pie de página no tapará nada de contenido. Por otro lado, utilizamos la estrategia vista en clase para ocultar la barra de herramientas en dispositivos móviles (window.scrollTo(0,1)).

Ahora la plantilla está lista. Partiremos de ella y diseñaremos el resto de ventanas, tal y como se muestra en las siguientes secciones. Evitaremos listar el código relativo a la plantilla, y nos centraremos en el diseño de las distintas páginas, es decir, del contenido del .

3.2 La ventana principal

Esta ventana será implementada en el fichero principal.html. Partiremos de template.html y lo renombraremos. En esta ventana debemos presentar un listado de tareas, así como un par de acciones. Introduciremos el siguiente contenido en principal.html.

```
<div id="pgPrincipal" class="page">
  <div class="header"><h1>Mis tareas</h1></div>
  <div class="content">
    <ul class="lista-tarea">
      <li>Tarea 1</li>
      <li>Tarea 2</li>
      <li>Tarea 3</li>
      <li>Tarea 4</li>
      <li>Tarea 5</li>
      <li>Tarea 6</li>
      <li>Tarea 7</li>
      <li>Tarea 8</li>
    </ul>
  </div>
  <div class="footer">
    <a id="btNuevaTarea" class="boton" href="#">Nueva tarea</a>
    <a id="btTodasTareas" class="boton" href="#">Todas las tarea</a>
  </div>
</div>
```

El código es muy sencillo, ahora debemos definir las clases lista-tarea y boton. A continuación se muestra una posible definición de dichas clases en styles.css:

```
.lista-tarea {
      list-style-type: none;
      margin: 0;
      padding: 0;
}

.lista-tarea li {
      background-color: lightgray;
      -webkit-border-radius: 6px;
      border: solid 2px lightgray;
      margin: 10px 0px;
      padding: 10px 8px;
}

.lista-tarea li:hover {
      border: solid 2px darkgray;
}

.lista-tarea li.pendiente:hover {
      border: solid 2px darkred;
}

.lista-tarea li.completada:hover {
      border: solid 2px darkgreen;
}

a.boton {
      display: block;
      background: -webkit-gradient(linear, left top, left bottom, color-
stop(0.0, #b3b3b3), color-stop(0.4, #666666), color-stop(1.0, #333333));
      color: white;
      text-decoration: none;
      padding: 8px;
      margin: 0px 10px 10px 10px;
      text-align: center;
      -webkit-box-shadow: 0 2px 2px #333333;
      -webkit-border-radius: 6px;
```

```
      border: solid 1px gray;
}

a.boton:hover {
      border: solid 1px white;
}
```

Con estos estilos obtendremos un resultado similar al siguiente:

3.3 La ventana de nueva tarea

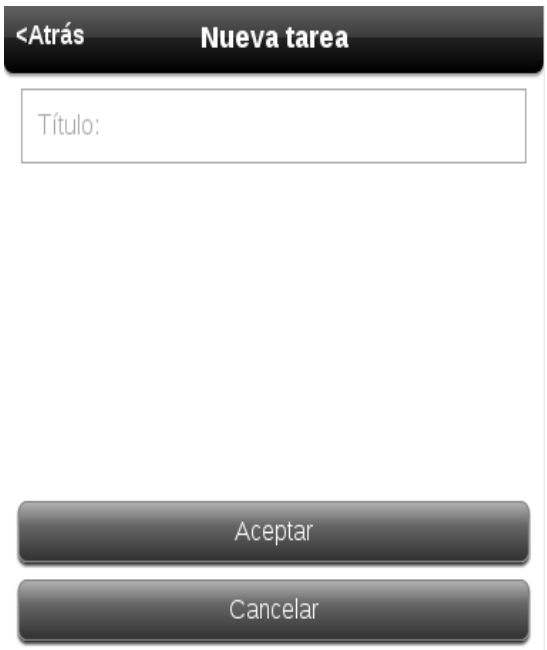
Nuevamente, partiremos de template.html y lo renombraremos como nuevatarea.html. En esta ventana es necesario incluir una caja de texto para introducir el título de la tarea y un par de botones. El contenido HTML es muy simple:

```
<div id="pgNuevaTarea" class="page">
  <div class="header"><a href="#">&lt;Atr&aacute;s</a><h1>Nueva tarea</h1></div>
  <div class="content">
    <input id="txtTitulo" type="text" placeholder="T&iacute;tulo:" >
  </div>
  <div class="footer">
    <a id="btAceptar" class="boton" href="#">Aceptar</a>
    <a id="btCancelar" class="boton" href="#">Cancelar</a>
  </div>
</div>
```

Sólo es necesario añadir algo de estilo para que la caja de texto resulte algo más accesible.

```
input[type="text"] {
      width: 90%;
      display: block;
      font-size: 1.05em;
      margin: 10px auto;
      padding: 10px;
}
```

El resultado sería como sigue:



3.4 La ventana de todas las tareas

Partimos de template.html y lo renombramos como todastareas.html. En esta ocasión, la ventana contendrá un conjunto de tareas, algunas en estado pendiente y otras en estado completada. Además, será posible filtrarlas por estado o por fecha. El siguiente código HTML será apropiado:

```
<div id="pgTodasTareas" class="page">
  <div class="header"><a href="#">&lt;Atr&aacute;s</a><h1>Todas las
tareas</h1></div>
  <div class="content">
    <ul class="lista-tarea">
      <li class="completada">Tarea 1</li>
      <li class="completada">Tarea 2</li>
      <li class="completada">Tarea 3</li>
      <li class="pendiente">Tarea 4</li>
      <li class="pendiente">Tarea 5</li>
      <li class="pendiente">Tarea 6</li>
      <li class="pendiente">Tarea 7</li>
      <li class="pendiente">Tarea 8</li>
    </ul>
  </div>
  <div class="footer">
    <fieldset>
      <legend>Filtros</legend>
       <ul class="filtros">
         <li><input id="chkPendientes" type="checkbox" value="pendientes"
checked>Pendientes</li>
         <li><input id="chkCompletadas" type="checkbox" value="completadas"
checked>Completadas</li>
         <li><input id="txtFecha" type="datetime" placeholder="Fecha:
DD/MM/AAAA"></li>
       </ul>
    </fieldset>
  </div>
</div>
```

Y añadimos los siguientes estilos a styles.css:

```
ul.filtros {
	list-style-type: none;
	margin: 0;
	padding: 0;
}

ul.filtros li {
	margin: 10px 0;
}

input[type="datetime"] {
	width: 100%;
	display: block;
	font-size: 1.05em;
}

.lista-tarea li.pendiente {
	background-color: tomato;
	border: solid 2px tomato;
}

.lista-tarea li.completada {
	background-color: lightgreen;
	border: solid 2px lightgreen;
}
```

Obteniendo un resultado similar al siguiente:

3.5 La ventana de editar tarea

Partimos de template.html y lo renombramos como editartarea.html. En esta ventana se listarán los detalles de la tarea, así como un par de acciones que permitan completarla (si está pendiente) o eliminarla.

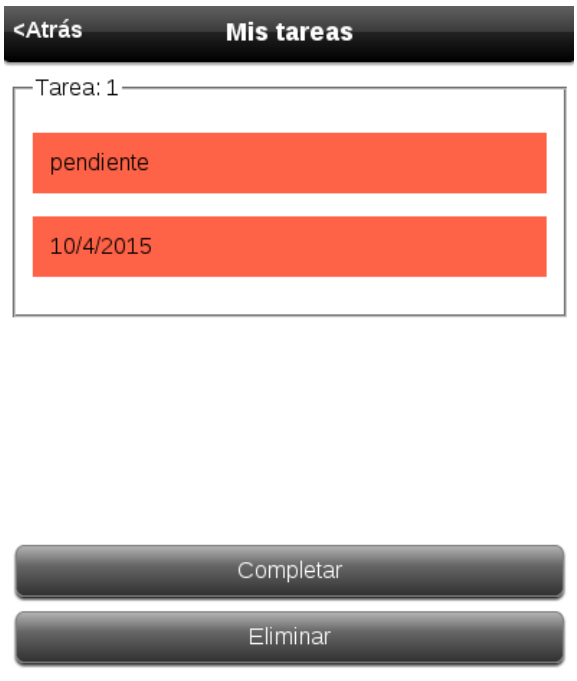
El código HTML sería similar al siguiente:

```
<div id="pgEditarTarea" class="page">
  <div class="header"><a href="#">&lt;Atr&aacute;s</a><h1>Mis tareas</h1></div>
  <div class="content">
    <fieldset>
      <legend>Tarea 1</legend>
        <p class="pendiente">Pendiente</p>
        <p class="pendiente">11/03/2015</p>
    </fieldset>
  </div>
  <div class="footer">
    <a id="btCompletar" class="boton" href="#">Completar</a>
    <a id="btEliminar" class="boton" href="#">Eliminar</a>
  </div>
</div>
```

Mientras que los nuevos estilos a añadir serían:

```
p.pendiente {
      padding: 10px;
      background-color: tomato;
      border: solid 2px tomato;
}

p.completada {
      padding: 10px;
      background-color: lightgreen;
      border: solid 2px lightgreen;
}
```

Con el siguiente resultado:



3.6 Juntándolo todo: Single Page Application

Como ya hemos visto en las sesiones de teoría, una web app multipágina suele estar contenida en un único documento HTML. Con este espíritu deberemos crear un único documento HTML que

recoja todas las páginas definidas en las secciones anteriores. Para ello, debemos cerciorarnos de que cada página posea un identificador único, para poder seleccionarla de manera individual. Revisar esta condición en nuestros diseños. El siguiente paso consiste en crear un documento HTML gestor-tareas.html e insertar en él todas nuestras páginas que incluíamos en los distintos ficheros principal.html, nuevatarea.html, editartarea.html y todastareas.html.

Una vez nuestra aplicación multipágina reside en un único documento HTML será necesario ocultar todas las secciones, excepto la principal, y habilitar un mecanismo de navegación entre páginas. Para ocultar todas las páginas bastará con añadir a la clase CSS .page la propiedad:

```
display:none
```

Respecto al mecanismo de navegación, éste es uno de los objetivos de la siguiente sección.

4. Funcionalidad de la aplicación

Una vez finalizadas las maquetas de nuestra aplicación, llega el momento de dotarlas de funcionalidad. Para ello, lo primero que haremos será definir el modelo de datos, es decir, la estrategia que se seguirá para almacenar los datos de la aplicación, así como su estructura. Una vez el modelo de datos esté claro, el siguiente paso consiste en diseñar un mecanismo de navegación entre páginas que sea consistente. Por último será necesario definir el código JS necesario para visualizar y actualizar el modelo de datos en las distintas páginas de la aplicación. Revisamos todas estas cuestiones en las siguientes secciones.

4.1 Modelo de datos

Crearemos un fichero JavaScript denominado modelo.js donde incluiremos toda la funcionalidad relativa al modelo, y lo ubicaremos en el directorio js/ de nuestra web app (js/modelo.js). Incluiremos este fichero desde gestor-tareas.html.

La aplicación posee un modelo de datos muy sencillo: una base de datos de tareas.

Las tareas podrían ser objetos JS con los siguientes campos:

- id: nos permitirá identificar a las tareas de manera unívoca
- titulo: el título de la tarea
- ts: timestamp de creación de la tarea
- estado: 'pendiente' o 'completada'

Los identificadores de las tareas deben ser únicos, y mantendremos un contador para garantizarlo: nextId. Para facilitar la creación de las tareas definiremos un constructor Tarea que recibirá como parámetro el título de la tarea y que rellenará el resto de propiedades de manera automática, tal y como se muestra en el siguiente código:

```
var nextId = 1;

function Tarea(titulo) {
      this.id = nextId++;
      this.titulo = titulo;
      this.estado = 'pendiente';
      this.ts = new Date().getTime();
}
```

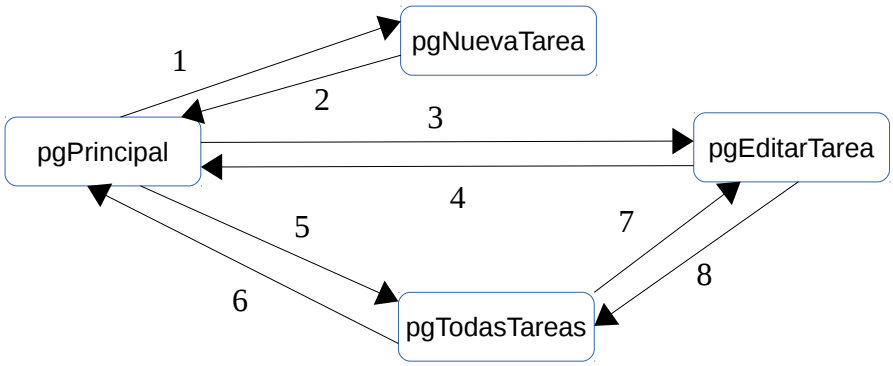
Las tareas se almacenarán en un array JS, que denominaremos tareasDB. Desarrollaremos un conjunto de rutinas que nos permitirán manipular esta base de datos de una manera sencilla. A continuación las mostramos.

```
var tareasDB = [];

/* crea una nueva tarea en la bd */
function nuevaTarea(title) {
      console.log('nuevaTarea(' + title + ')');
      var tarea = new Tarea(title);
      tareasDB.unshift(tarea);
      return tarea;
}

/* busca una tarea en la bd */
function buscarTarea(id) {
      console.log('buscarTarea(' + id + ')');
      for (var i = 0; i < tareasDB.length; i++)
            if (tareasDB[i].id == id) return tareasDB[i];
      return null;
}

/* completa una tarea */
function completarTarea(id) {
      console.log('completarTarea(' + id + ')');
      var tarea = buscarTarea(id);
      if (tarea) tarea.estado = 'completada';
}

/* elimina una tarea de la bd */
function eliminarTarea(id) {
      console.log('eliminarTarea(' + id + ')');
      for (var i = 0; i < tareasDB.length; i++) {
            if (tareasDB[i].id == id) {
                  tareasDB.splice(i, 1);
                  return;
            }
      }
}
```

4.2 Navegación entre páginas

Crearemos un fichero JavaScript js/app.js que contendrá el resto de la web app, incluyendo tanto el sistema de navegación entre páginas, como las rutinas que permitan visualizar y actualizar el modelo de datos.

En nuestra web app disponemos de 4 páginas diferentes, que pueden representarse por medio del siguiente grafo de flujo. Rotulamos las transiciones entre páginas con números para una explicación más detallada.



A continuación describimos las distintas transiciones:

1. El usuario presiona sobre el botón de nueva tarea (btNuevaTarea).

2. El usuario presiona sobre el botón de cancelar (btCancelar) o bien presiona sobre el enlace “Atrás” ubicado en la cabecera.

3. El usuario presiona sobre una tarea.

4. El usuario presiona sobre el botón de completar (btCompletar), eliminar (btEliminar), o bien selecciona el enlace “Atrás” ubicado en la cabecera, siempre y cuándo saltó a esta página desde la página principal (pgPrincipal).

5. El usuario presiona el botón de todas las tareas (btTodasTareas).

6. El usuario presiona sobre el botón de completar (btCompletar), eliminar (btEliminar), o bien selecciona el enlace “Atrás” ubicado en la cabecera, siempre y cuándo saltó a esta página desde la página de todas las tareas (pgTodasTareas).

Como se puede observar, incluso con 4 simples páginas el diagrama no resulta del todo sencillo. En particular resaltamos las transiciones (4) y (8). ¿Cómo saber cuándo hemos accedido a la página pgEditarTarea desde pgPrincipal o desde pgTodasTareas? Por otro lado, es importante indicar que es posible ejercitar la misma transición por varias causas. Por ejemplo, las transiciones (4) y (8) se pueden disparar porque el usuario ha presionado el enlace de “Atrás” o bien porque ha presionado uno de los botones btCompletar o btEliminar. Si no tenemos cuidado con el sistema de navegación, podemos llegar a replicar código que dificulte sobremanera el mantenimiento de la aplicación.

Por todas las cuestiones anteriores, vamos a diseñar un mecanismo de navegación genérico, basado en tres piezas fundamentales: la pila de navegación navPila, la rutina navSaltar(pag) que nos permite ir hacia delante y la rutina navAtras() que nos permite volver hacia atrás. En las siguientes secciones se describen estas tres piezas.

4.2.1 La pila de navegación navPila

Se trata de una pila (en JS un array) en la que ubicaremos las distintas ventanas que se van mostrando en la aplicación. Si lo analizamos, podemos comprobar que estas ventanas muestran un orden LIFO. La última ventana es la ventana que actualmente se está visualizando. La penúltima ventana es aquella ventana desde la que se abrió la ventana actual y así …

Cuando desde una ventana se salta a otra ventana, la nueva ventana se apila en navPila y se muestra. Cuando desde una ventana volvemos hacia atrás se desapila la ventana actual y se muestra la siguiente ventana apilada.

Declarar la pila en el código:

```
var navPila = [];
```

4.2.2 La rutina navSaltar(pag)

Esta rutina permitirá saltar a una página destino, aquella que se pasa como parámetro (se pasa su identificador). Cuando se salta a una nueva página siempre se siguen los mismos pasos:

1. Se apila la página destino en navPila.

2. Se refresca el contenido de la página destino, regenerando todos los elementos HTML que sea necesario.

3. Se oculta la página anterior y se muestra la página destino.

Para realizar (2) es necesario definir una rutina de “refresco” para cada página de la aplicación. Esta rutina de refresco permitirá regenerar el contenido de dicha página cuando sea necesario. Para ello, se definen 4 nuevas rutinas: refrescarPrincipal(), refrescarNuevaTarea(), refrescarEditarTarea(id) y refrescarTodasTareas(). El contenido de estas rutinas se presentará en una sección posterior. Por ahora basta con saber que existirán. Para automatizar el salto entre páginas, se definirá una tabla fnRefrescar, que dado el identificador de una página devolverá la rutina responsable de refrescarla, tal y como se muestra en el siguiente código:

```
var fnRefrescar = {
      'pgPrincipal': refrescarPrincipal,
      'pgNuevaTarea': refrescarNuevaTarea,
      'pgEditarTarea': refrescarEditarTarea,
      'pgTodasTareas': refrescarTodasTareas
};
```

Para realizar (3) se diseña una rutina cambiarPagina(pag, pagAnterior) que permitirá ocultar la página anterior y mostrar la siguiente. Se lista a continuación:

```
function cambiarPagina(pag, pagAnterior) {
      console.log('cambiarPagina(' + pag + ',' + pagAnterior + ')');
      if (pagAnterior) $('#' + pagAnterior).css('display', 'none');
      $('#' + pag).css('display', 'block');
      // ajustar altura
      $('#' + pag + ' .content').height('auto');
      // añadir pie de página
      $('#' + pag + ' .content').height($('#' + pag + ' .content').height() + $
('#' + pag + ' .footer').height());
      // ocultar barra de navegación
      window.scrollTo(0, 1);
}
```

En esta rutina se incluye el código que permitía redefinir la altura del contenido de la página en función de la altura del pie de página, con el objetivo de que no se tape nada como consecuencia de la estrategia de posicionamiento fixed definida sobre todos los footer.

Por último, se lista la rutina navSaltar(pag):

```
function navSaltar(pag) {
      console.log('navSaltar(' + pag + ')');
      // 1. recuperar página anterior
```

```
	var pagAnterior;
	if (navPila.length > 0) pagAnterior = navPila[navPila.length - 1];
	// 2. apilar página nueva
	navPila.push(pag);
	// 3. refrescar página nueva
	var args = [];
	for (var i = 1; i < arguments.length; i++) args[i-1] = arguments[i];
	fnRefrescar[pag](args);
	// 4. cambiar página
	cambiarPagina(pag, pagAnterior);
}
```

4.2.3 La rutina navAtras()

Esta rutina permitirá navegar hacia atrás, en caso de existir una página anterior. Para ello, se siguen los siguientes pasos:

1. Se desapila la página actual.
2. Se obtiene la página anterior, que ahora es el tope de la pila.
3. Se oculta la página actual y se muestra la página anterior.

A continuación se lista la rutina:

```
function navAtras() {
	console.log('navAtras()');
	// 1. desapilar actual
	var pgActual = navPila.pop();
	if (navPila.length > 0) {
		// 2. obtener anterior
		var pgAnterior = navPila[navPila.length-1];
		// 3. refrescar
		fnRefrescar[pgAnterior]();
		// 4. mostrar
		cambiarPagina(pgAnterior, pgActual);
	}
}
```

4.3 Visualizar y actualizar el modelo de datos

Una vez el sistema de navegación ha quedado definido, el último paso consiste en diseñar el código necesario para visualizar y actualizar el modelo de datos.

Como se ha visto en la sección anterior, cada vez que se abre una nueva página, es necesario regenerarla. En general, cuando se regenera una página, se incluye en su interior tanto los datos como los mecanismos que permiten actualizar dichos datos. Por lo tanto, el grueso de esta sección se llevará a cabo en las funciones refrescarXXX() que se presentaron anteriormente, pero que no se definieron. Las definimos a continuación.

La función refrescarPrincipal() es la encargada de generar la página principal pgPrincipal de la web app. La parte dinámica de esta página es esencialmente el listado de tareas. Los botones btNuevaTarea y btTodasTareas existirán siempre. Por lo tanto, esta función se encargará de recorrer la base de datos de tareas y mostrar las 5 últimas tareas pendientes, refrescando el listado de tareas. Refrescar los datos consiste esencialmente en destruirlos y volver a generarlos. Cuando generamos el nuevo listado de tareas es importante incluir no sólo los datos adecuados, sino también el comportamiento que se espera obtener de dicho listado. En nuestro caso, cuando

se presiona sobre una tarea de la lista, se espera navegar a la página pgEditarTarea que muestre los detalles de dicha tarea. Este comportamiento también debe ser incluido en el listado generado de manera dinámica, tal y como se muestra en el siguiente código:

```
function refrescarPrincipal() {
  console.log('refrescarPrincipal()');
  $('#pgPrincipal .lista-tarea').empty();
  var numTareas = 0;
  var i = 0;
  while (numTareas < 5 && i < tareasDB.length) {
    if (tareasDB[i].estado == 'pendiente') {
      $('#pgPrincipal .lista-tarea').append('<li ' +
         'onclick="navSaltar(\'pgEditarTarea\',' +
          tareasDB[i].id + ')">Tarea: ' + tareasDB[i].titulo + '</li>');
      numTareas++;
    }
    i++;
  }
}
```

La función refrescarNuevaTarea() únicamente debe inicializar los campos que permiten introducir información acerca de la nueva tarea a crear. El contenido de esta página es estático, no cambia. A continuación se muestra una posible implementación:

```
function refrescarNuevaTarea() {
      console.log('refrescarNuevaTarea()');
      $('#pgNuevaTarea #txtTitulo').val('');
}
```

La función refrescarEditarTarea(id) recibe como parámetro el id de la tarea sobre la que debe mostrar detalles. El objetivo de esta rutina consiste en recuperar de la base de datos la tarea con el identificador especificado y mostrar sus datos en la página pgEditarTarea. Si la tarea está pendiente entonces se podrá completar o eliminar. Si la tarea está completada únicamente se podrá eliminar. Por lo tanto, tanto la generación de los datos, como las acciones debe ser dinámica y depende de la tarea mostrada. A continuación se lista la implementación de esta rutina:

```
function refrescarEditarTarea(id) {
  console.log('refrescarEditarTarea(' + id + ')');
  if (!id) return;
  var tarea = buscarTarea(id);
  if (!tarea) {
    alert('error, tarea con id ' + id + ' no existe');
    return;
  }
  // detalles tarea
  var html = '<legend>Tarea: ' + tarea.titulo + '</legend>';
  var date = new Date(tarea.ts);
  html += '<p class="' + tarea.estado + '">' + tarea.estado + '</p><p class="' +
           tarea.estado + '">' +
           [date.getDate(), date.getMonth()+1, date.getFullYear()].join("/") +
           '</p>';
  $('#pgEditarTarea .content fieldset').html(html);
  // botón completar
  html = tarea.estado == 'pendiente'? '<a id="btCompletar"' +
        'onclick="completarTarea(' + id + '); navAtras();" class="boton"' +
        ' href="#">Completar</a>': '';
  // botón eliminar
  html += '<a id="btEliminar" onclick="eliminarTarea(' + id + '); navAtras();"'
          + ' class="boton" href="#">Eliminar</a>'
```

```
  $('#pgEditarTarea .footer').html(html);
}
```

La función refrescarTodasTareas() debe listar todas las tareas almacenadas en la base de datos en función de los filtros activados. Para ello, lo primero es recuperar el estado actual de los filtros y después recorrer la base de datos de tareas, comprobando qué tareas verifican los filtros, y generando el contenido HTML correspondiente. Nuevamente, al igual que en refrescarPrincipal(), cuando se presiona sobre una tarea es necesario navegar a la página pgEditarTarea, y este comportamiento debe ser incluido en el HTML generado, tal y como se muestra a continuación:

```
function refrescarTodasTareas() {
  console.log('refrescarTodasTareas()');
  $('#pgTodasTareas .lista-tarea').empty();
  // filtrar
  var pendientes = $('#pgTodasTareas #chkPendientes').is(":checked");
  var completadas = $('#pgTodasTareas #chkCompletadas').is(":checked");
  var fecha = $('#pgTodasTareas #txtFecha').val();
  try {
    fecha = fecha && fecha.split('/');
    fecha = new Date(fecha[2], fecha[1]-1, fecha[0]);
  } catch (e) {
    console.log('Fecha no valida')
  }
  console.log('pendientes:' + pendientes + ',completadas:' + completadas +
              ',fecha:' + fecha);
  for (var i = 0; i < tareasDB.length; i++) {
    var tarea = tareasDB[i];
    if (tarea.estado == 'pendiente' && !pendientes) continue;
    if (tarea.estado == 'completada' && !completadas) continue;
    if (fecha && tarea.ts < fecha.getTime()) continue;

    $('#pgTodasTareas .lista-tarea').append('<li class="' +
            tarea.estado  +
            '" onclick="navSaltar(\'pgEditarTarea\',' + tarea.id + ')">Tarea: '
            + tarea.titulo + '</li>');
  }
}
```

Por último, una vez las rutinas de refresco han sido definidas, sólo falta definir el comportamiento de las partes estáticas de la aplicación (los botones que no se generan dinámicamente y que permanecen estables desde el principio hasta el fin) así como saltar a la página principal para comenzar la ejecución de la aplicación. Esto lo haremos una vez jQuery se haya cargado y garantice que el DOM está dispuesto para ser manipulado. A continuación listamos el último código de js/app.js, que da el pistoletazo de salida de la web app:

```
$(function() {

    // eventos pgPrincipal
    $('#pgPrincipal #btNuevaTarea').click(function() {
      navSaltar('pgNuevaTarea');
    });
    $('#pgPrincipal #btTodasTareas').click(function() {
      navSaltar('pgTodasTareas');
    });

      // eventos pgNuevaTarea
    $('#pgNuevaTarea #btAceptar').click(function() {
      // guardar tarea
      nuevaTarea($('#txtTitulo').val());
```

```
      navAtras();

    });
      $('#pgNuevaTarea #btCancelar').click(function() {
            navAtras();
    });

      // eventos pgTodasTareas
      $('#pgTodasTareas input').change(function() {
            refrescarTodasTareas();
            cambiarPagina('pgTodasTareas');
    });

      // eventos atrás
      $('.header a').click(function() {
            navAtras();
      });

    navSaltar('pgPrincipal');

});
```

5. Extensiones

Una vez implementada la funcionalidad básica de la web app se plantean una serie de extensiones, con el objetivo de que el alumno aventajado profundice en algunos conceptos presentados en clase o adquiera soltura con las tecnologías presentadas en el curso. Las extensiones se presentan en los siguientes apartados.

5.1 Almacenamiento local

En la web app implementada en las secciones anteriores, la base de datos es volátil, es decir, se crea cuando se carga la web app y se destruye automáticamente cuando la web app se cierra.

Para evitar este problema, proponemos implementar la base de datos en almacenamiento persistente, para que la información persista entre distintas ejecuciones de nuestra web app, utilizando para ello el API Web Storage presentado en clase. Web Storage proporciona una implementación persistente de un diccionario, donde tanto las claves como los valores deben ser String.

Para implementar esta mejora, deberemos modificar el fichero modelo.js y cualquier acceso directo a nuestra base de datos tareasDB. Veremos que el diseño actual de la aplicación no es todo lo adecuado para efectuar esta migración de manera sencilla ya que desde distintas rutinas refrescaXXX() accedemos directamente a tareasDB y esto no es muy deseable. Habría sido mejor estrategia encapsular cualquier acceso al modelo de datos en modelo.js. Se propone al alumno efectuar esta mejora.

5.2 Fecha de vencimiento

En esta extensión se propone ampliar la definición de tarea, incluyendo también una fecha de vencimiento. En la página principal pgPrincipal se mostrarán primero las tareas que hayan vencido, con un código de color llamativo. En la página de todas las tareas pgTodasTareas también se mostrará un código de color especial para este tipo de tareas, para que seamos capaces de identificarlas fácilmente.

5.3 Prioridades

Nuevamente, se propone ampliar la definición de tareas para que incluya una prioridad. Se deja al alumno aventajado la especificación concreta de dicha prioridad (un número, un String, …). La página principal pgPrincipal mostrará primero las tareas con mayor prioridad, pudiendo definir un código de colores, por ejemplo una intensidad de color, de manera que cuanto más intenso sea, más prioridad tendrá la tarea. En la página de todas las tareas pgTodasTareas se habilitarán mecanismos de filtrado teniendo en cuenta también la prioridad.