Estudio de caso

Jesús Mejuto
Marc Verdera

PID_00198059

Universitat Oberta de Catalunya

Índice

1. Diseño de una intranet

1.1. Introducción

Se trata de un proyecto real que desarrollamos dentro de una agencia para un gran cliente. El proyecto duró unos tres meses (trabajando un equipo de cinco personas) y fue un caso de éxito con el que el cliente quedó satisfecho. Gran parte de este éxito lo conseguimos gracias a las metodologías ágiles, que nos ayudaron a entendernos con el cliente, obtener su confianza e implicación.

Este era un proyecto de gran envergadura, así que la colaboración entre el equipo fue vital para poder desarrollarlo con la excelencia que buscábamos.

1.2. Objetivos

A continuación, os mostraremos recursos, metodologías, pequeñas problemáticas y conocimientos necesarios para poder desarrollar un proyecto de este tipo.

Los objetivos de este documento es que podáis tener una visión global de cómo enfocar un proyecto de tal envergadura. Pondremos énfasis también en la metodología de trabajo que hemos visto en esta asignatura para que podáis ver cómo se aplica a un caso real.

Aun así, entraremos en un nivel de detalle suficiente y os daremos recursos y herramientas para poder ser capaces de llevarlo a cabo vosotros mismos de una manera profesional.

1.3. Equipo

A continuación, definiremos el equipo encargado para el proyecto y las características de cada uno de sus miembros:

- ***Product owner***. Es el miembro del equipo que está involucrado en el proyecto y con el cliente a la vez. Se encargará de gestionar/motivar al equipo y de captar las necesidades del cliente para poder optimizar los *sprints* y las prioridades en cada momento del proyecto.

Web recomendada

Para más información:
http://en.wikipedia.org/wiki/product_owner-Core_Scrum_roles

- ***Scrum master***. Este miembro es el que conoce la metodología ágil y ayuda a aplicarla para optimizar los resultados del proyecto. Se encarga de que el equipo llegue a los objetivos establecidos por el *product owner*.

Web recomendada

Para más información:
http://en.wikipedia.org/wiki/scrummaster#Core_Scrum_roles

- **UX/UI *designer***. En realidad, diferenciaríamos entre un diseñador de UX (*user experience*) y uno de UI (*user interface*), pero en la actualidad una figura polivalente nos aporta mucho más valor para este proyecto. Se encargará de analizar la intranet actual, llevar a cabo un análisis de la competencia y las metodologías de creación de contenidos para intranets y crear un árbol de contenidos y *wireframing* para el proyecto.

Webs recomendadas

Para más información:
http://en.wikipedia.org/wiki/user_experience_design
http://en.wikipedia.org/wiki/ui_designer

- **IxD/ *visual designer***. En realidad, diferenciaríamos entre un diseñador de IxD (*interaction designer*) y uno visual (*visual designer*), pero en la actualidad una figura polivalente nos aporta mucho más valor para este proyecto. Se encargará de interpretar los *wireframes* con la colaboración del UX/UI *designer* y del *web developer* y de ejecutar los diseños.

Webs recomendadas

Para más información:
http://en.wikipedia.org/wiki/interaction_design
http://en.wikipedia.org/wiki/visual_designer

- ***Web developer***. Un *web developer* se encarga de programar tanto el *FrontEnd* como del *BackEnd* de una página web. Se encargará de tomar los diseños y transformarlos en HTML/CSS siguiendo los estándares web y las normativas de accesibilidad W3C.

Web recomendada

Para más información:
http://en.wikipedia.org/wiki/frontend

En este caso, tenemos un equipo en el que cada uno de ellos tiene una especialidad, pero conoce el trabajo de los demás miembros del equipo. De este modo, conseguimos una muy buena comunicación entre ellos y tendremos presente el estado del proyecto global en las etapas de *planning*.

1.4. *Briefing*

En este proyecto, recibimos un *briefing* de proyecto donde se nos especificaban las necesidades del cliente. Estas necesidades constaban de una remodelación de la intranet que tenían actualmente puesto que ya era antigua, habían ido modificando y añadiendo funcionalidades y les funcionaba muy lenta.

A partir de ahí, teníamos que analizar las funcionalidades de la intranet actual, añadir las nuevas funcionalidades que especificaban en el *briefing* y crear el primer esbozo para poder elaborar el nuevo árbol de contenidos.

Web recomendada

Para más información:
http://es.wikipedia.org/wiki/briefing

Briefing

El ***briefing*** es la parte estratégica de la preparación de una acción publicitaria. Es la elección ordenada, estratégica y creativa de los datos que nos permitirá definir los objetivos publicitarios de forma correcta y medible. Es un documento escrito donde el Departamento de Marketing tiene que incluir toda la información necesaria para dejar claras las diferencias comerciales y definir lo que se quiere conseguir con la publicidad. Lo crea la empresa cliente con su información del mercado y con las líneas básicas del plan de

marketing de la marca que desea publicitar. Es un documento resumen, muy sintético, que facilita la tarea de la agencia.

Extraído de la Wikipedia

1.4.1. Resumiendo este *briefing*

Nuestro cliente es una gran marca que da licencias a productores de cualquier tipo de producto (como camisetas, cascos, zapatos o tazas) para basarse en su marca.

Los licenciatarios (productor que quiere crear el producto) se tendrán que registrar en la intranet para que el cliente haga la validación del producto. El productor se registrará y subirá la propuesta de diseño del producto a la intranet, donde el cliente la validará o hará ciertas observaciones. Si no se aprueba el producto, este procedimiento se irá repitiendo de forma cíclica hasta que el producto sea aprobado por el cliente.

Aparte de esto, la intranet es un espacio donde estarán todos los documentos corporativos de la marca, como notas de prensa y noticias.

También tendremos que plantear la opción de que haya un usuario visitante, que tendrá acceso a ciertos documentos y a la información donde se explica el procedimiento antes descrito. Esta figura se contempla, ya que es una forma de captación de nuevos licenciatarios, como fuente de ingresos de la marca.

Desde el punto de vista formal, se quiere respetar la línea gráfica y el tono de comunicación de la marca, así que tendremos que revisar también los documentos corporativos a la hora de crear el diseño visual.

1.5. Iteraciones

1.5.1. Iteración 1: análisis del proyecto

Antes de llevar a cabo la primera reunión de ***kick-off*** nos repartimos el trabajo entre el equipo para poder tener una visión global entre todos del estatus del proyecto en cada una de sus fases. Una vez nos reunimos con el cliente, presentamos el equipo y exponemos la metodología que se va a seguir. El cliente conocerá así al equipo y el planteamiento del proyecto.

Kick-off

El ***kick-off*** es la primera reunión con el equipo de proyecto y el cliente del proyecto. Esta reunión trata la definición de los elementos de base para el proyecto y otras actividades de planificación del mismo. Esta reunión presenta a los miembros del equipo del proyecto al cliente y proporciona la oportunidad de hablar de la función de cada miembro del equipo. Otros elementos de base en el proyecto también se pueden hablar en esta reunión (como el horario o el estado que informa).

En este punto es cuando nos ponemos a trabajar. Para empezar, miraremos la intranet que tienen actualmente, donde veremos (junto con las nuevas necesidades que nos ha aportado el cliente en el *briefing*) cuáles son las premisas en las que nos tenemos que basar.

Para empezar, el *product owner* establecerá los objetivos del primer *sprint*. En este caso, será la presentación de la primera fase del árbol de contenidos. En esta fase, el UX/UI *designer* tendrá la responsabilidad de liderar el proyecto y de desarrollar la primera propuesta.

Para hacer el análisis, el equipo mirará recursos en Internet como otras intranets o en este caso aplicaciones de gestión de proyectos, puesto que esta intranet al fin y al cabo tratará de eso.

Aplicaciones de gestión de proyectos

Nosotros nos hemos fijado en aplicaciones como las siguientes:

- BaseCamp
- HiTask
- Omniplan

En la actualidad, estas aplicaciones son líderes en el mercado y muchas empresas las utilizan para su gestión.

Una vez hecho este análisis, decidiremos qué prácticas nos interesan o podemos reaprovechar para crear nuestra intranet y, a partir de aquí, nos pondremos a crear el árbol de contenidos.

1.5.2. Iteración 2: arquitectura de la información general

Para el árbol de contenidos, el UX/UI *designer* lo hará en papel y lápiz para plantear una estructura jerárquica de la información, empezando desde la *home page* y bajando categóricamente en cada una de las diferentes pantallas de la intranet.

En este caso, tendremos cuatro grandes áreas de la intranet, que explicamos a continuación:

1) Usuarios. Existen tres tipos de usuarios:

a) Administradores. Los administradores tendrán todos los permisos para manipular usuarios y evaluar los proyectos que subirán los licenciatarios. Podrán aprobar o denegar proyectos, dar de alta o dar de baja a usuarios, subir documentos y noticias, entre otras funciones que definiremos a continuación.

b) Licenciatarios. Serán capaces de crear un proyecto nuevo, subir imágenes y hacer comentarios. Aparte de esto, como los visitantes, también podrán consultar toda la documentación de la intranet, ya sean vídeos, documentos en pdf o Excel.

c) Visitantes. Podrán consultar toda la documentación de la intranet, ya sean vídeos, documentos en pdf o Excel con un cierto límite, delimitado por el administrador cuando dé de alta a un visitante.

2) Documentación. En esta sección, tenemos documentación de diferente tipo como la siguiente:

- guías de estilo,
- vídeos corporativos, o
- documentos corporativos de diferente tipo (como inspiración o publicidad).

Tendremos que ser capaces de crear un nuevo documento como administradores y mostrar el detalle de cada uno de ellos.

3) Proyectos. En esta sección, tendremos que tener en cuenta que hay una gran cantidad de proyectos y que el cliente los quiere clasificar antes que nada por licenciatarios y después como:

- aprobados,
- en proceso, o
- denegados.

De todos modos, también querrá ver un listado con fotografías de todos los productos y que se puedan reconocer cuáles están aprobados, en proceso o denegados.

4) Prensa. Esta área la tendremos que clasificar por notas de prensa y noticias. Tendremos que ser capaces de crear un nuevo documento como administradores y mostrar el detalle de cada uno de ellos.

Una vez definidas las necesidades de estructura de la intranet, el ejercicio será crear un árbol de contenidos que englobe todas esas funciones. Esta será la primera entrega, donde validaremos con el cliente la estructura base antes de avanzar con el árbol de contenidos.

Webs recomendadas

Para crear este árbol, os recomendamos que os miréis una serie de recursos que podéis encontrar en Internet. Para este tipo de proyectos hay programas especializados en elaborar árboles de contenidos como los siguientes:

Omnigraffle

Axure

Irise

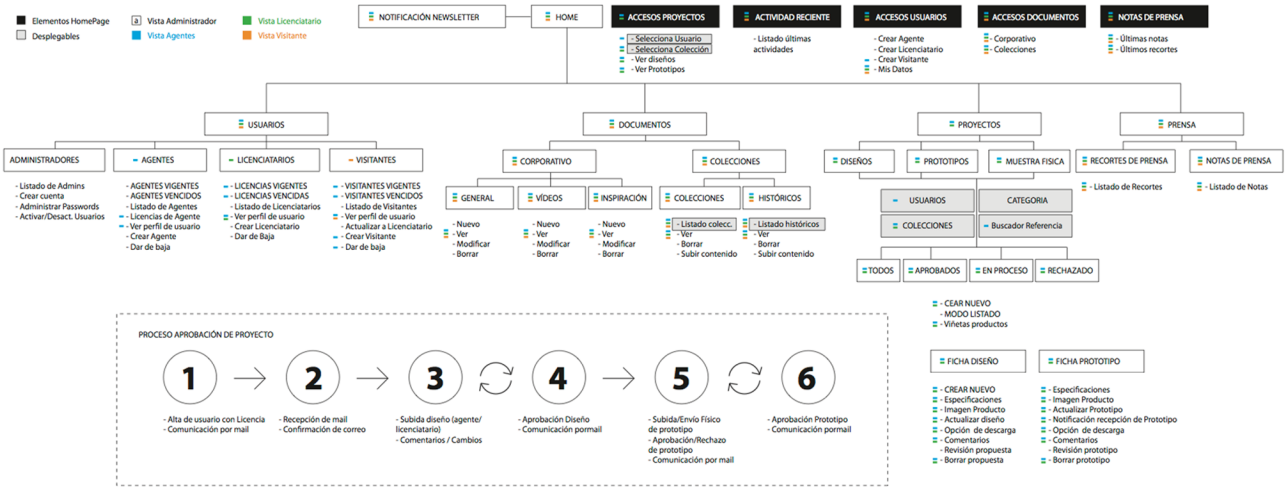
smartdraw

conceptdraw

Pero también podemos hacerlo con programas vectoriales como Flash, Illustrator o Fireworks.

1.5.3. Iteración 3: arquitectura de la información detalle

A partir de la estructura base ya aprobada por el cliente, desarrollaremos en esta iteración el detalle de las áreas y los contenidos para cada una de las secciones.



1.5.4. Iteración 4: *wireframing home*

Para crear los *wireframes*, tendremos que tener en cuenta una serie de normas de usabilidad.

Usabilidad

La usabilidad (del inglés *usability*) es la facilidad con la que las personas pueden usar una herramienta o un *widget* para lograr un objetivo concreto. El término también puede hacer referencia a los métodos para medir la usabilidad y al estudio de los principios que dotan a un objeto de usabilidad.

La usabilidad, tradicionalmente ligada al diseño de paneles de control o interfaces por parte de los ingenieros, se ha popularizado con la llegada de gran cantidad de dispositivos electrónicos (como móviles, cajeros automáticos, ordenadores o automóviles) y el uso de la informática (como sistemas operativos, software o páginas web). Para hacerlos intuitivos y fáciles de utilizar, pueden requerir disciplinas tan diversas como la ingeniería, la psicología, el diseño y la comunicación visual, la informática, la etnografía o los estudios de mercado.

La usabilidad permite:

- Más eficiencia; con menos tiempo se puede acabar una tarea concreta.
- Facilidad de aprendizaje; el funcionamiento de un objeto se puede deducir observándolo.
- Más satisfacción del usuario.

Jakob Nielsen y Steve Krug son dos de los expertos destacados en usabilidad.

Extraído de la Wikipedia

Tendremos que tener presentes todas estas normas y recomendaciones para crear los *wireframes* de la intranet, pero todo esto llevado a la práctica se resume en una sola cosa: **sentido común**.

Los *wireframes* dibujan a modo de esquema todos los elementos que tendrá la intranet. Normalmente podemos utilizar diferentes elementos esquemáticos para dibujar cada una de las pantallas, como por ejemplo:

- textos descriptivos,
- cuadrados,
- triángulos,
- redondas,
- flechas,
- texto falso, e
- iconos.

Pero siempre acostumbran a ser en blanco y negro y sin tener en cuenta proporciones, tamaños o alineaciones exactas.

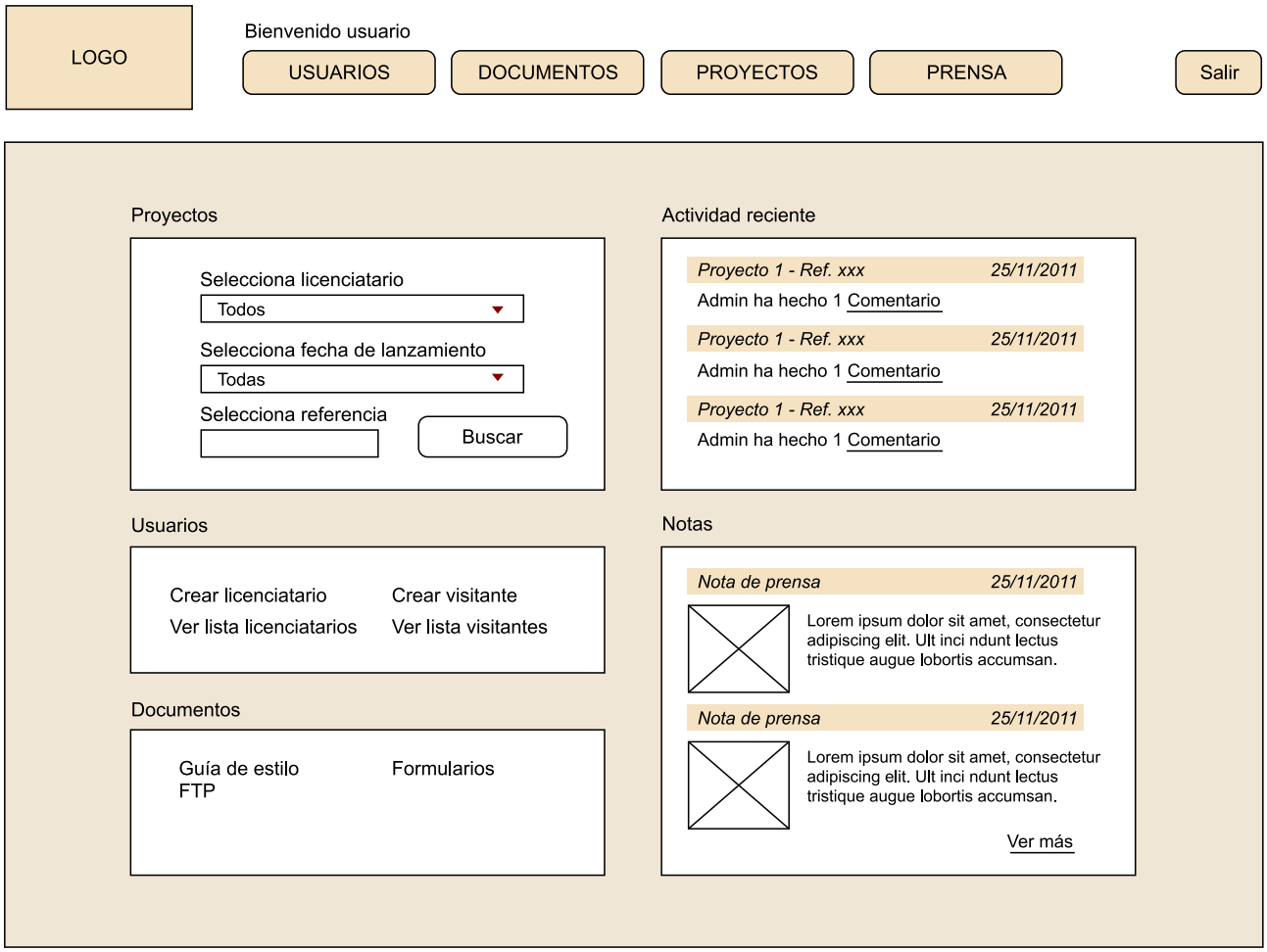
Crear wireframes

Para crear estos *wireframes*, podéis usar cualquier programa vectorial. Sin embargo, igual que con la arquitectura de información, os podrían ayudar ciertas herramientas como las siguientes:

- Omnigraffle
- Axure
- Irise
- smartdraw
- conceptdraw

En la Red, podréis encontrar recursos varios para Illustrator, Fireworks y Omnigraffle.

En esta fase, basándonos en el árbol de contenidos ya aprobado por el cliente, plantearemos los *wireframes* de la *home*. Este *wireframe* es importante por sí solo porque es donde planteamos el menú de navegación y la estructura formal del *site*.



1.5.5. Iteración 5: *wireframing* distribuidores y detalle (primera fase)

Una vez aprobado el *wireframe* de la *home page* y una vez esté aprobada la navegación principal y la estructura del *site*, trabajaremos las pantallas distribuidoras.

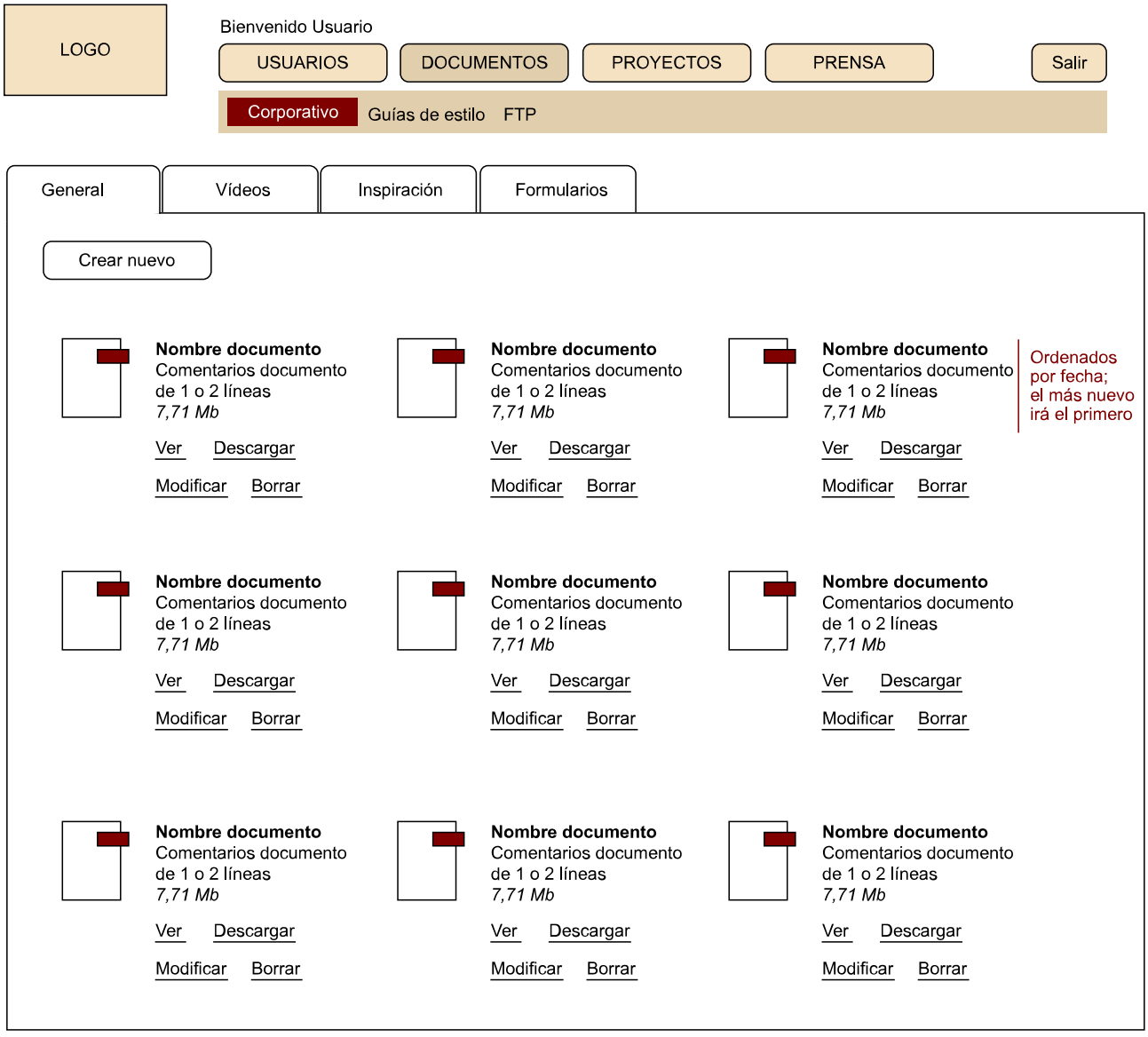
Estos *wireframes* también son primordiales en un proyecto de esta envergadura, ya que estamos pensando la distribución en detalle del contenido y los menús de segundo nivel.

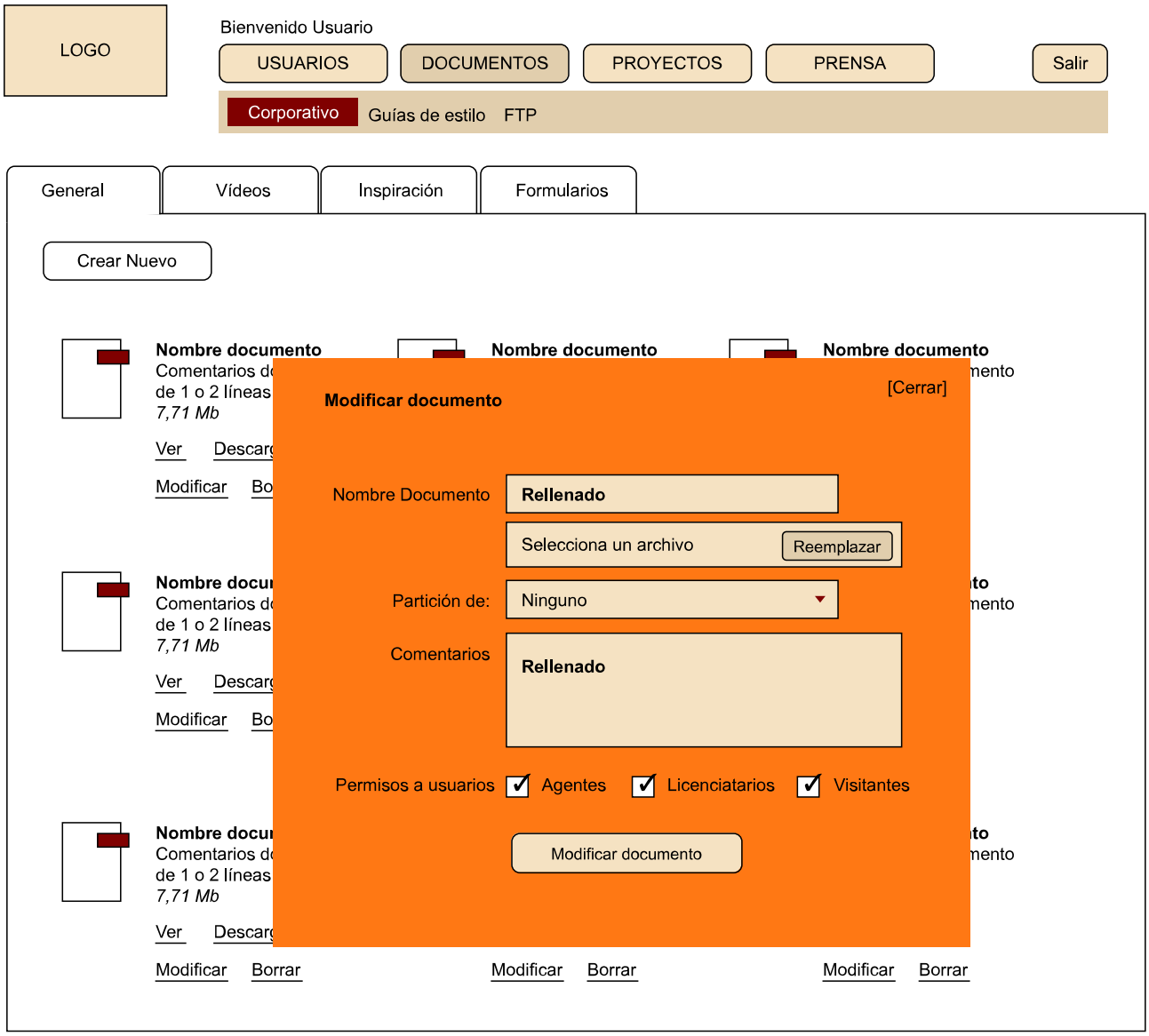
El ejercicio aquí será presentar los *wireframes* de las pantallas de usuarios, proyectos, documentación y prensa. Estos *wireframes* no estarán pulidos en detalle pero, a modo generalista, mostrarán cómo está distribuida la información y las funcionalidades que aplicaremos a la intranet.

1.5.6. Iteración 5: *wireframing* distribuidores y detalle (segunda fase)

En esta fase, una vez nos hayan aprobado la iteración anterior, desarrollaremos los *wireframes* ahora ya sí con todo detalle de funcionalidades y contenidos.

El ejercicio aquí será presentar los *wireframes* de las pantallas de usuarios, proyectos, documentación y prensa completos.





1.5.7. Iteración 6: *Look&Feel home page*

Paralelamente a la creación de *wireframes*, podemos ir pensando en el ***Look&Feel*** de la intranet. Cuando tengamos la *home* planteada a escala de *wireframes* ya podremos empezar a trabajar en este ámbito.

> ***Look&Feel***
>
> En el mundo interactivo, ***Look&Feel*** es un término utilizado respecto a una interfaz gráfica y comprende aspectos de su diseño, incluyendo elementos como colores, formas, trazado y tipos de letra, así como el comportamiento de elementos dinámicos como botones, cajas y cartas.
>
> Extraído de la Wikipedia

En el caso de crear el ***Look&Feel***, el *visual designer* tendrá en cuenta todas las cosas vistas hasta ahora:

- **El análisis de la competencia** (también a escala gráfica), donde el diseñador aportará ideas, recursos y pondrá en uso las **mejores prácticas**. Tener

una idea de a quién va dirigido el proyecto, qué necesidades se plantean, entre otros. Es muy importante a la hora de crear un diseño visual.

- **La arquitectura de la información**, donde el diseñador tendrá una idea global del alcance del proyecto. Es importante que el diseñador tenga conciencia de este alcance para poder plantear de manera exitosa un diseño.

- **Los *wireframes***, donde se verá detallado pantalla por pantalla cómo irá organizada la información y qué habrá en cada una de las plantillas. A partir de aquí, es trabajo del diseñador definir la **jerarquíavisual** y la importancia que tendrán cada uno de los elementos en el diseño.

Una mejor práctica

Una mejor práctica es un método o técnica que ha demostrado de forma consistente resultados mejores que los conseguidos con otros medios, y que se utiliza como punto de referencia. Además, una "mejor" práctica puede evolucionar con las mejoras que se descubren. La mejor práctica es considerada por algunos como una palabra de moda de negocios, que se utiliza para describir el proceso de desarrollo y seguimiento de una manera estándar de hacer las cosas que varias empresas pueden utilizar.

Las mejores prácticas se utilizan para mantener la calidad como una alternativa a las normas obligatorias legisladas y pueden estar basadas en la autoevaluación o la evaluación comparativa.

Extraído de la Wikipedia

Jerarquía visual

En toda composición gráfica se tiene que crear una jerarquía visual adecuada, con objeto de que los elementos más importantes de la misma se muestren debidamente acentuados.

Mediante un adecuado diseño se puede establecer un camino visual que dirija el ojo del espectador y le vaya mostrando la información contenida en la composición de forma organizada, lógica y fiable, que dirija su percepción por la ruta más idónea.

Extraído de DesarrolloWeb

Una vez hecho este recordatorio, antes de ponerse a diseñar, tendremos que hacer un *benchmarking* a escala visual para ver qué recursos/gráficos/estilos serán los más idóneos para aplicar al diseño visual de la intranet.

Webs recomendadas

Hay muchísimos recursos en cuanto a gráficos, iconos, menús, texturas, tipografías y demás en Internet. A continuación, ponemos algunos portales de referencia que nos ayudarán a la hora de encontrar los recursos que necesitamos.

http://www.smashingmagazine.com/

http://www.alvit.de/handbook/

http://skout.co.za/

http://www.bluevertigo.com.ar/

Una vez nos hayamos hecho una idea de qué estilo queremos dar al diseño pasaremos a la acción. Para diseñar, podremos usar Photoshop o Fireworks, dependiendo del programa con el que nos sintamos más a gusto. Muchos diseñadores usan Photoshop, pero la verdad es que Fireworks es el programa que Adobe prepara específicamente para el diseño web.

Antes de empezar a trabajar con el programa, es muy importante tener en cuenta dos cosas principales:

1) La resolución de pantalla: la resolución de pantalla estándar es a 1.024 x 768, en la que el área visual real será de aproximadamente unos 960 x 600. La de nuestro diseño, a diferencia de un diseño destinado a imprenta, tendrá que tener 72 pp (píxeles por pulgada).

2) La cuadrícula: una herramienta que nos ayudará mucho a la hora de diseñar es un buen sistema de cuadrículas o *grid system*. Normalmente, estos sistemas tienen posibilidades y se suelen medir con número de columnas.

Webs recomendadas

Para escoger la mejor cuadrícula que se adecue al diseño, podemos buscar en algunos de los enlaces que mostramos a continuación:

http://www.thegridsystem.org/

http://960.gs/

http://modulargrid.org/#app

Resolución de pantalla

La resolución de pantalla es el número de píxeles que puede ser mostrado en la pantalla. Viene dada por el producto del ancho por el alto, medidos los dos en píxeles, con lo que se obtiene una relación, llamada relación de aspecto. En esta relación de aspecto, se puede encontrar una variación, de acuerdo con la forma del monitor y de la tarjeta gráfica. Se pueden diferenciar dos tamaños de pantalla diferentes:

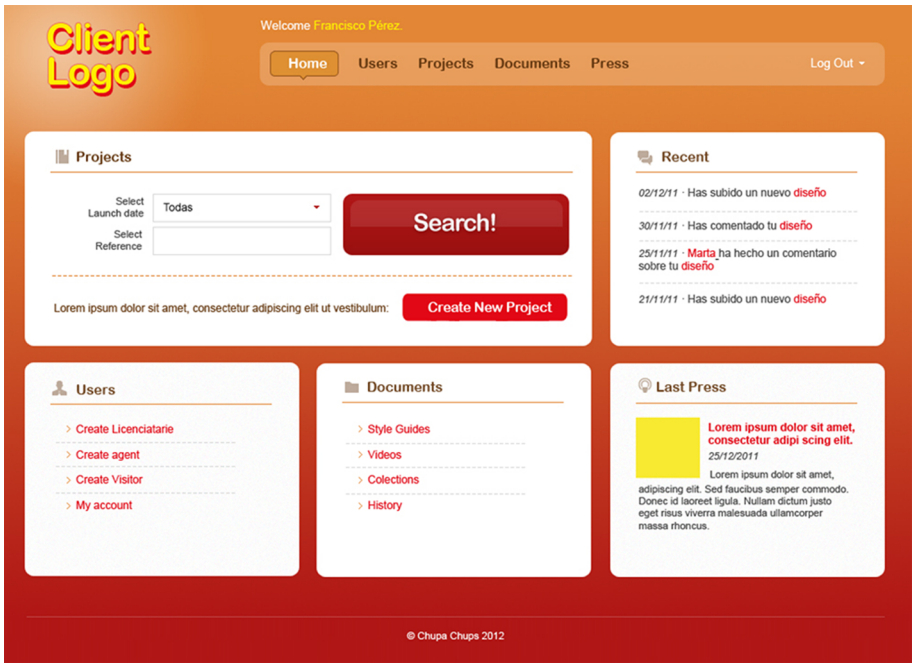
1) **Tamaño absoluto**: son las dimensiones anchura y altura de la ventana del monitor, medidas generalmente en pulgadas. Depende del monitor.

2) **Resolución o tamaño relativo**: viene determinada por el número de píxeles que se muestran en la ventana del monitor y el píxel es la unidad mínima de información que se puede presentar en pantalla, de forma generalmente rectangular. Depende de la tarjeta gráfica.

Extraído de la Wikipedia

En esta iteración, vamos a hacer solo el diseño de la *home page*. Este es normalmente el diseño más complicado porque tiene que entrar por los ojos al cliente. En cuanto a la estructura y contenidos ya lo tenemos todo definido, pero ahora nos toca convencer al cliente a escala gráfica.

El ejercicio aquí será hacer un diseño de la *home page*.



1.5.8. Iteración 7: *Look&Feel* distribuidores y detalle (primera fase)

Una vez el cliente haya dado su aprobación al diseño de la *home page*, haremos el diseño de las pantallas interiores.

Este es un proyecto de mucha envergadura, así que tendremos que escoger estratégicamente qué pantallas vamos a diseñar. Escogeremos las plantillas básicas para dotar a los programadores de todos los elementos gráficos para desarrollar toda la intranet. Normalmente, escogeremos la *home* y quince pantallas que en global reúnan todos los elementos que salgan en toda la intranet.

En esta primera fase, crearemos el diseño de la pantalla de proyectos.

1.5.9. Iteración 8: *Look&Feel* distribuidores y detalle (segunda fase)

En la iteración anterior, habremos diseñado la pantalla de proyectos, así que habremos definido el submenú y muchos de los elementos principales de la intranet.

En esta iteración, ya haremos el diseño del resto de pantallas, donde utilizaremos muchos de los elementos utilizados en la *home page* y la pantalla de proyectos.

1.5.10. Iteración 9: maquetación HTML/CSS Base

Como vamos indicando en todo el caso práctico, este es un proyecto que tiene hitos con el cliente y que se basa en las metodologías ágiles, por lo que mantendremos al cliente al corriente de cómo va avanzando el proyecto.

La parte de la maquetación no será una excepción, así que también iremos mostrando si hay evolución en los diferentes *sprints*.

La parte de maquetación web la podremos ir avanzando una vez tengamos aprobado el diseño principal, puesto que nos podremos basar en los estilos generales y guiarnos con los *wireframes* desarrollados previamente.

Para hacer la maquetación tendremos en cuenta varias cosas que son imprescindibles en la actualidad. La maquetación se hará SEO-friendly y respetando los estándares W3C. Ahora ya nos pondremos con HTML5 y CSS3, puesto que empieza a convertirse en el estándar.

W3C

Pastilla del W3C de recomendaciones de accesibilidad

El World Wide Web Consortium (W3C) es un consorcio internacional que trabaja para desarrollar y promocionar estándares para la *World Wide Web*. Lo dirige Tim Berners-Lee, creador de la WWW y autor de las especificaciones del *Uniform Resource Locator*, el HTTP y el HTML, que son las principales tecnologías de esta red.

El consorcio se administra de forma conjunta entre el MIT (en los Estados Unidos), el ERCIM (en Europa) y la Universidad de Keio (en el Japón). Aun así, el W3C también tiene repartidas oficinas en todo el mundo.

Este fue creado inicialmente para garantizar una mayor compatibilidad y acuerdo entre los diferentes miembros del sector a la hora de adoptar nuevas tecnologías web. Antes de su creación, por ejemplo, ya existían problemas de compatibilidad por culpa de diferentes versiones de HTML específicas de cada vendedor. Hay que decir que estos problemas todavía perduran.

Un documento del W3C sigue diferentes **etapas de maduración** antes de convertirse en una recomendación propiamente dicha:

- *Working Draft* (WD) (borrador de trabajo),
- *Candidate Recommendation* (CR) (recomendación candidata),
- *Proposed Recommendation* (PR) (propuesta de recomendación), y
- finalmente una *W3C Recommendation* (REC) (recomendación).

Las recomendaciones pueden actualizarse de forma separada en documentos de *Errata*, hasta el momento en que haya suficientes cambios significativos como para producir una nueva edición de la recomendación. Este es el caso del XML, que se encuentra actualmente en la tercera edición. Asimismo, el W3C también publica diferentes tipos de notas informativas, que en principio no habría que considerar como estándares.

Extraído de la Wikipedia

HTML 5

Web recomendada

Para más información sobre SEO-Friendly:

http://webdesign.about.com/od/seo/a/seo_contento.htm

El HTML 5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas `<div>` y `<span>`, pero tienen un significado semántico, como por ejemplo `<nav>` (bloque de navegación del sitio web) y `<footer>`. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos `<audio>` y `<video>`. También presenta mejoras en el elemento `<canvas>`.

Algunos elementos de HTML 4.01 han quedado obsoletos, incluyendo elementos puramente de presentación, como `<fuente>` y `<center>`, puesto que los efectos son manejados por el CSS. También se pone énfasis en la importancia del scripting DOM para el comportamiento de la web.

Diferencias con HTML 4 y XHTML

Como nuevos elementos: *article*, *dialog*.

Como nuevos atributos: *media*, *ping*, *autofocus*, *inputmode*, *min*, *max*, *pattern*.

Los elementos eliminados son: *acronym*, *applet*, *basefont*, *big*, *center*, *dir*, *font*, *frame*, *frameset*, *isindex*, *noframes*, *s*, *strike*, *tt*, *u*.

Atributos eliminados:

- *rev* y *charset* en `<link>` y `<a>`
- *target* en `<link>`
- *nohref* en `<area>`
- *profile* en `<head>`
- *version* en `<html>`
- *name* en `<map>`
- *scheme* en a `<meta>`
- *archive*, *classid*, *codetype*, *declare* y *standby* en `<object>`
- *valuetype* en `<param>`
- *charset* en `<script>`
- *summary* en `<table>`

- *header*, *axis* y *abbr* en `<td>` y `<th>`

Extraído de la Wikipedia

Nosotros nos ocuparemos del *FrontEnd* de la intranet y, para ayudarnos en este desarrollo, podemos conseguir varios recursos. También hay que estar informados de todas las novedades del sector, por lo que es importante estar muy conectados a la Red para no quedarnos atrás en un mundo tan cambiante como este.

Webs recomendadas

Framework HTML/CSS:

http://www.blueprintcss.org/

Portales de referencia donde mantenernos informados:

http://www.html5rocks.com/es/

http://www.css3.info/

http://www.css3.com/

También podemos buscar varios recursos en portales como los que hemos comentado antes:

http://www.smashingmagazine.com/

http://www.alvit.de/handbook/

http://skout.co.za/

http://www.bluevertigo.com.ar/

Muchos *FrontEnd developers* emplean bastante los *frameworks*, pero nosotros somos más de la idea de empezar una maquetación de cero o hacernos una estructura propia personalizada. Después nos será más fácil adaptar el código.

El ejercicio aquí será construir la maquetación de las pantallas básicas (*home page* y proyectos) y crearemos los estilos suficientes para que los desarrolladores del *FrondEnd* del proyecto puedan empezar a trabajar con la programación de la intranet.

1.5.11. Iteración 9: maquetación HTML/CSS pantallas

Una vez los programadores nos hayan aprobado la estructura de la maquetación, nos pondremos a maquetar el resto de pantallas para entregar el paquete completo en HTML y CSS.

El ejercicio aquí consistirá en crear la maquetación de las pantallas restantes (documentos, usuarios y prensa).

1.5.12. Iteración 10: fin del proyecto

El proyecto acabará en cuanto esté entregado al cliente el árbol de contenidos, los *wireframes*, el diseño y la maquetación *FrontEnd* de la intranet. Una vez tengamos todo esto, entregaremos al cliente y a sus desarrolladores de *BackEnd* todos los archivos.

Aunque parezca que aquí se acaba el proyecto, no es realmente cierto, puesto que en un proyecto siempre acostumbra a haber pequeños ajustes en todas las fases que habremos ejecutado.

Gracias a las metodologías ágiles, seguramente no nos encontraremos con muchos de estos ajustes y los que nos encontremos serán fáciles de solucionar.

Esto será debido a la alta implicación del cliente en cada fase del proyecto y a que el cliente tendrá la sensación (y de hecho será verdad) de que el proyecto es un poco suyo y le convencerá el resultado.

1.6. Enlaces de interés

http://es.wikipedia.org/wiki/wikipedia:Portada

http://www.desarrolloweb.com

http://basecamp.com/

http://hitask.com/

http://www.omnigroup.com/

http://www.conceptdraw.com/

http://www.axure.com/

http://www.irise.com/

http://www.smartdraw.com/

http://www.smashingmagazine.com/

http://www.alvit.de/handbook/

http://skout.co.za/

http://www.bluevertigo.com.ar/

http://www.thegridsystem.org/

http://960.gs/

http://modulargrid.org

http://webdesign.about.com

http://www.blueprintcss.org/

http://www.html5rocks.com/es/

http://www.css3.info/

http://www.css3.com/

2. Diseño de una aplicación para iPhone

2.1. Introducción

Este es el claro ejemplo de un proyecto de diseño de una aplicación móvil partiendo de un encargo real de una empresa de telefonía móvil. Normalmente los encargos para elaborar aplicaciones móviles vienen de clientes que no tienen ninguna idea sobre el entorno.

Sin embargo, en este caso, el encargo viene dado por un cliente que ya tiene conocimientos del entorno en cuanto a programación, pero necesita asesoramiento en cuanto a conceptualización de la aplicación, así como para crear los *wireframes* y el diseño.

2.2. Objetivos

Los objetivos para este proyecto serán poder desempeñar un trabajo rápido y eficaz, utilizando los menores recursos para obtener un máximo rendimiento. En un mundo tan cambiante como el nuestro, es importante saber gestionar más de un proyecto a la vez y, por eso, hay que saber distribuir muy bien los recursos.

2.3. Equipo

A continuación, definiremos el equipo encargado para el proyecto:

- *product owner*
- *Scrum master*
- UX/UI designer
- IxD/visual designer
- *mobile developer*

En este caso, contamos con un equipo en el que cada uno de ellos tiene una especialidad, pero conoce el trabajo de los demás miembros del equipo. De este modo, conseguimos una muy buena comunicación entre ellos y tendremos presente el estado del proyecto global en las etapas de *planning*.

2.4. *Briefing*

En este proyecto, recibimos un *briefing* de proyecto donde se nos especificaban básicamente las necesidades técnicas a la hora de desarrollar el diseño del proyecto.

El *briefing* obtenido

El sistema debe proveer una interfaz adaptada a las capacidades y limitaciones del canal, así como del terminal por el que se esté accediendo. Para el canal **móvil**, se deberá detectar el tipo de terminal que utiliza el usuario, para seleccionar entre tres tipos de plantillas:

1) ***Mobile web smartphone:*** destinado a terminales que tengan navegadores que soporten HTML 5.0, CSS 3.0 y Javascript 1.6 o superior y que en su mayoría están basados en la plataforma Webkit.

2) ***Mobile web featurephone***: destinado a terminales que tengan soporte a navegación 3G, pero que no brindan un soporte completo a HTML, por lo que se limitan a HTML 4.0, CSS 2.0 y una baja compatibilidad en Javascript 1.2.

3) ***Wap simplephone***: destinado a terminales con una capacidad de navegación limitada (2G) y con pantallas reducidas, así hay que evitar el uso de imágenes en lo posible, así como el uso de tecnologías de *client-side script* y una compatibilidad limitada de CSS y HTML MP.

Para el canal **web**, se utilizará una sola interfaz, con compatibilidad *cross-browsers* en los navegadores más populares del mercado.

A continuación, describimos con mayor detalle los requisitos técnicos de cada interfaz. Cualquier otra especificación o tecnología no expresada en este documento podrá ser asumida y considerada por el *factory*, siempre y cuando no irrumpa con las expuestas.

1) ***Mobile web smartphone***

Esta interfaz deberá implementarse mediante una plantilla Drupal, que cumpla con los siguientes requisitos técnicos:

- Debe estar diseñada y optimizada para pantallas de 320 px de ancho, pero a la misma vez deberá tener la habilidad de adaptarse de forma automática a pantallas de 640 px de ancho. Mediante *CSS float width* 100%.
- El peso de las imágenes no debe superar los 10 kilobytes.
- El peso de una página promedio (sin el contenido plano) no debe superar los 30 K (incluye js, css, html e imágenes).
- No debe utilizar imágenes en formato SVG.
- Podrá estar basada en el Drupal jQuery Mobile Theme.
- Deberá incluir el meta tag *viewport* con *width = device - width*, lo que garantiza que el diseño y la maquetación puedan adaptarse al ancho de la pantalla del dispositivo.
- Debe estar desarrollada bajo HTML 5.0.
- No debe utilizar los elementos HTML 5.0: *websocket*, *canvas*, *media*, *audio*, *video* ni *object*.
- No deberá utilizar *iframes* ni *popup windows*.
- Podrá utilizar CSS 3.0.
- Deberá utilizar CSS Media Queries, *max/min-device-width*, y *orientation*, para adaptar el diseño del portal cuando se gira el móvil de forma horizontal o vertical (*Landscape*, *Portail*).
- Se permite el uso de Javascript 1.6 o superior, librerías jquery, llamadas *ajax* o cualquier otra tecnología *client-side script* que garantice su funcionamiento a través de las distintas plataformas y *browsers* especificados en este documento.
- Podrá utilizar los *frameworks* y librerías jQuery Mobile, jQTouch, Sencha Touch para el desarrollo de interfaces móviles.
- No debe utilizar Adobe Flash, Flash Lite, Shockwave, Microsoft Silverlight ni cualquier otra herramienta que necesite la preinstalación de un *browser plugin*.
- Debe ser *cross-platform* compatible para los siguientes sistemas operativos y *browsers* móviles:

- Android 1.6 o superior,
- Apple Safary iOS 3.x o superior,
- Backberry Browser 5.x o superior,
- Nokia Symbian^3 y Serie 60 3th,
- Browsers 7.x o superior,
- Nokia MeeGo Browser,
- Opera Mobile (no Opera Mini),
- Firefox Mobile 1.x o superior,
- Windows Phone 6 o superior, y
- otros navegadores móviles basados en Webkit & Webkit2.

2) ***Mobile web featurephone***

Esta interfaz deberá implementarse mediante una plantilla Drupal, que cumpla con los siguientes requisitos técnicos:

- El diseño debe estar optimizado para pantallas de 220 px de ancho.
- El peso de las imágenes no debe superar los 8 kilobytes.
- El peso de una página promedio (sin el contenido plano) no debe superar los 15 K.
- No debe utilizar imágenes en formato SVG.
- No debe utilizar llamadas *Ajax*.
- El desarrollo deberá estar basado en HTML 4.0 y CSS 2.0.
- Debe hacer un uso limitado de Javascript 1.4.
- Se deberán hacer pruebas de compatibilidad y *cross-platform* a los siguientes dispositivos:
  - Blackberry OS 4.x y superior,
  - teléfonos Symbian Serie 40, serie N o 60 en su primera, segunda y tercera edición, y
  - Windows Mobile y Windows CE o ambos.

3) ***Wap simplephone***

Esta interfaz deberá implementarse mediante una plantilla Drupal, que cumpla con los siguientes requisitos técnicos:

- El diseño debe estar optimizada para pantallas de 128 px de ancho.
- El peso de las imágenes no deberá superar los 5 kilobytes.
- El peso de una página promedio (sin el contenido plano) no deberá superar los 8 K.
- Solo debe estar basado en xHTML MP 1.0.
- Solo debe utilizar Wireless CSS 1.1.
- No debe utilizar ningún código Javascript o código *script client-side*.
- No podrá utilizar imágenes en formato SVG o PNG, solo JPG y GIF.
- El diseño deberá ser *crowss-browser* para cualquier terminal compatible con WAP 2.0.

Por otro lado, tuvimos una reunión con el cliente en la que nos explicó qué tipo de aplicación querían desarrollar y qué objetivos teníamos que lograr.

Se trataba de una aplicación móvil para un portal de telefonía que ofrecía juegos, tonos y salvapantallas, entre otros, y era un portal competencia de Movistar emoción.

2.5. Iteraciones

2.5.1. Iteración 1: análisis del proyecto

A partir del *briefing* obtenido, tendremos que llevar a cabo un análisis general de la competencia y de las *apps* parecidas que podamos encontrar en el mercado.

Normalmente, el cliente siempre nos dirá cuál es su competencia y cuáles son las preferencias/ideas para desarrollar su propio proyecto. Sin embargo, nuestro trabajo consistirá en ir más allá y buscar recursos, otras páginas similares y métodos interesantes que podamos aplicar en el proyecto.

Es bueno buscar en mercados más avanzados al nuestro para inspirarnos. Normalmente, en páginas americanas, francesas, inglesas o japonesas será donde encontraremos los métodos más interesantes.

Webs recomendadas

Algunos portales para ver tendencias y recursos móviles son los siguientes:

http://cssiphone.com/

http://www.mobileawesomeness.com/

http://www.refinedmobile.com/

http://tapfancy.com/

http://www.tappgala.com/

http://www.iosinspires.me/category/appinterfaces/

Una vez hecho este análisis, decidiremos qué prácticas nos interesan o podemos reaprovechar para crear la *app* y, a partir de ahí, nos pondremos a crear el árbol de contenidos.

Web recomendada

Para más información:
http://en.wikipedia.org/wiki/mobile_apps

2.5.2. Iteración 2: arquitectura de la información

Para elaborar el árbol de contenidos, el UX/UI *designer* lo hará en papel y lápiz para plantear una estructura jerárquica de la información, empezando desde la *home page* y bajando categóricamente en cada una de las diferentes pantallas de la aplicación.

Como hemos dicho en el apartado del *briefing*, se trata de crear un portal para una compañía de telefonía donde el sistema de negocio es muy claro: **la descarga de contenidos** (como tonos, música y juegos).

Para acceder a este portal, será necesario un registro y, una vez descargada la aplicación, el portal mostrará los contenidos organizados por:

- Destacados: en la *home* se mostrarán los contenidos más descargados o más novedosos.

- Noticias: noticias relacionadas con los contenidos o con el portal móvil.

- Archivos para descargar: donde se podrá valorar, compartir con las redes sociales y descargar.

- Información corporativa del cliente.

- Sección de contactos: contactos que tengan la misma aplicación y la opción de invitar a otros contactos del usuario a la aplicación.

- Perfil de usuario: se verán los datos personales y las opciones de configuración de las redes sociales.

Una vez definidas las necesidades de estructura de la *app*, el ejercicio consistirá en elaborar un árbol de contenidos que englobe todas estas funciones.

2.5.3. Iteración 3: *wireframing* general

Siguiendo el método ágil, habremos tenido validaciones del cliente antes de haber acabado todo el árbol y así podremos empezar con el prototipado (o *wireframing*) de la nueva intranet de forma paralela.

Para crear los *wireframes*, tendremos que tener en cuenta una serie de normas de usabilidad, igual que para cualquier aplicación web.

Para plantear unos *wireframes* para móvil, deberemos tener en cuenta diferencias, puesto que aquí debemos tener en cuenta que el usuario interactúa con el aparato de manera táctil y no con un ratón y un teclado como cuando el usuario está ante un ordenador.

También deberemos tener en cuenta que el usuario navegará mediante un aparato con una pantalla considerablemente más pequeña que la de un ordenador, así que tendremos que diseñar los elementos con proporciones diferentes a las que podemos estar acostumbrados cuando trabajamos para web.

Web recomendada

Un buen recurso para informarnos de todos estos detalles de usabilidad móvil es la página de Jakob Nielsen:
http://www.useit.com/alertbox/mobile-usability.html

Jakob Nielsen

Nielsen expone que lo habitual es que un usuario no lea con detalle ni siquiera una mínima parte de los textos de una página web. En su lugar, y por economía de tiempo, el usuario se limita a ojear la página. Es decir, el usuario realiza un rápido barrido visual de cada página buscando elementos que llamen su atención. Por lo tanto es fundamental la utilización de elementos como:

- Palabras resaltadas mediante negrita y cambios de color o de tamaño. En este sentido, los enlaces actúan como elementos de atracción visual pues se destacan del resto del texto.

- Listas de elementos con viñetas o numeradas.

- Títulos de sección y titulares breves intercalados (también denominados ladillos).

Debido a esta economía de lectura, según Nielsen, el contenido de un texto tiene que organizarse correctamente para ganar la atención del lector. Por ejemplo las ideas más importantes tienen que aparecer al principio, y después la argumentación de las mismas. De esta forma, nos aseguramos de que el posible lector recuerde mejor la información. Nielsen recomienda usar menos del 50% del texto usado habitualmente en una publicación escrita. Los usuarios se aburren con los textos largos. Los párrafos tienen que ser cortos, de dos o tres frases únicamente y muy directas en su estilo.

Por otro lado, asegura que los usuarios aprenden pronto a ignorar los mensajes publicitarios exagerados, incluso cuando intentan aparecer como información objetiva camuflados en el texto.

Extraído de la Wikipedia

En la primera fase de creación de *wireframes*, haremos una propuesta general de estructura. Antes de nada, plantearemos la página de registro y la *home page*.

En esta presentación, haremos dos versiones para la *home page*, para conseguir que el cliente se decida por una de ellas o una combinación de ambas.

2.5.4. Iteración 4: *wireframing* app

Una vez aprobada la propuesta de estructuración de los contenidos que habremos presentado con la *home page*, procederemos a ejecutar el resto de *wireframes* para la iteración siguiente.

Haremos un inciso para comentar que en cada iteración podemos aprovechar para hablar al cliente sobre la siguiente. En este caso, habríamos comentado nuestra idea de *wireframing* en la siguiente presentación, de forma que es imposible que nos desviemos mucho de lo que se imagina el cliente.

Así pues, en esta fase, vamos a desarrollar el resto de *wireframes*. Al ser una aplicación relativamente pequeña, no será necesario que separemos el resto de *wireframes* en diferentes iteraciones y así aceleraremos el desarrollo del proyecto.

2.5.5. Iteración 5: *Look&Feel*

Una vez aprobados los *wireframes*, la siguiente tarea será preparar una propuesta de ***Look&Feel*** para la *app*.

Así como en los *wireframes* hemos indicado hacer dos versiones (esta es una decisión más personal y dependerá del proyecto y del cliente), aquí nosotros somos partidarios de presentar solo una opción de diseño. Básicamente, lo creemos porque presentar más de una opción de diseño siempre da pie a confundir al cliente.

De este modo, tendremos que ser nosotros quienes valoremos la mejor versión a escala visual y funcional.

Aun así, en el *briefing* obtenido se nos pedía elaborar una aplicación que funcionara en tres tipos de dispositivos: iPhone, *featured phone* y *basic phone*. En cuanto al *wireframing*, no es relevante puesto que variará muy poco la estructura de los contenidos y solo plantearemos la versión iPhone, pero a partir de ahí será importante visualizar cómo quedará el diseño en las tres plataformas.

Webs recomendadas

Hay muchísimos recursos en cuanto a gráficos, iconos, menús, texturas y tipografías, entre otros, en Internet. A continuación, ponemos algunos portales de referencia que nos ayudarán a la hora de encontrar los recursos que necesitamos.

http://cssiphone.com/

http://www.mobileawesomeness.com/

http://www.refinedmobile.com/

http://tapfancy.com/

http://www.tappgala.com/

http://www.iosinspires.me/category/appinterfaces/

http://www.smashingmagazine.com/

http://www.alvit.de/handbook/

http://skout.co.za/

http://www.bluevertigo.com.ar/

Antes de empezar a trabajar con el programa, es muy importante tener en cuenta dos cosas principales:

1) La resolución de pantalla: en este caso, la resolución de pantalla en el móvil venía dada en el apartado del *briefing*. Aun así, debemos tener en cuenta que la retina *display* para el iPhone hace que la resolución cambie a 640 px de ancho.

2) La cuadrícula: una herramienta que nos ayudará mucho a la hora de diseñar es un buen sistema de cuadrículas o *grid system*. Normalmente, estos sistemas tienen posibilidades y se suelen medir con número de columnas.

Retina display

Retina display es una denominación de la pantalla *In-Plane Switching* (IPS) de 3,5 pulgadas (en diagonal) usada en los iPhone 4, iPhone 4S, en el iPod touch de cuarta generación y ahora también en la pantalla del nuevo iPad de tercera generación, a pesar de que la pantalla de este último es de solo 264 ppp.

Recibe este nombre debido a su alta densidad de píxeles, puesto que con sus 960 x 640 píxeles contiene cuatro veces más píxeles que la pantalla usada en el iPhone anterior (iPhone 3GS de 480 x 320 píxeles), manteniendo las 3,5 pulgadas de pantalla, lo que se traduce en 326 ppp. Según Steve Jobs durante su presentación en la WWDC 2010, esta resolución se encuentra sobre los 300 ppp que todavía son perceptibles por el ojo humano; por lo tanto, los píxeles de esta pantalla son tan pequeños que son indistinguibles a la vista humana y no hay diferencia con el material impreso. El único dispositivo móvil que ha superado la densidad del iPhone 4 hasta el momento es el Sony Xperia S con una pantalla de 4,3" y 1.280 x 720 píxeles, lo que proporciona un ppp de 342.

Extraído de la Wikipedia

El ejercicio aquí será elaborar un diseño de la página de registro y de la *home page* para las tres plataformas. En esta fase, no solo es importante un buen diseño, atractivo y funcional, sino también la presentación del mismo.

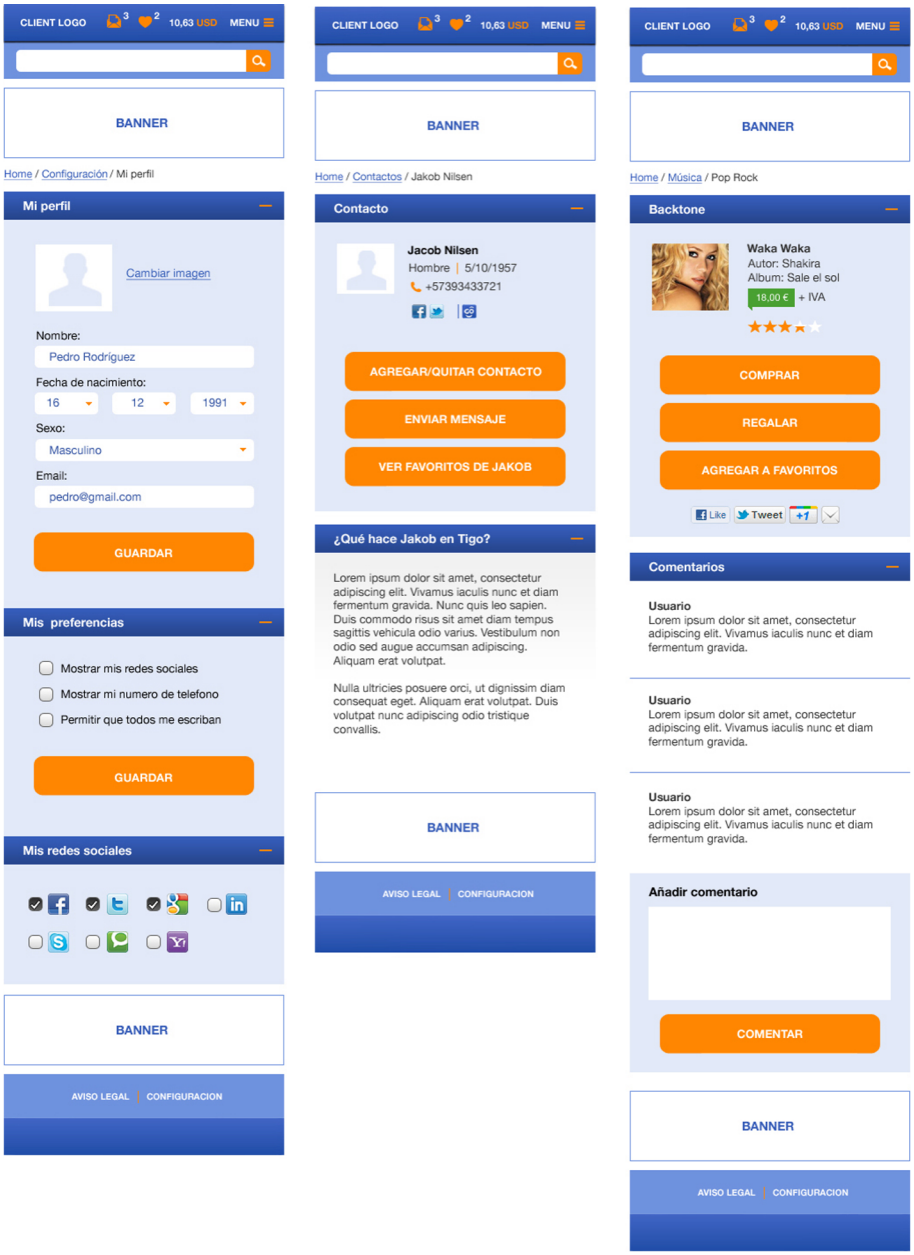
2.5.6. Iteración 6: diseño de aplicación

Cuando nos viene dado un diseño y tenemos que realizar el resto de pantallas de una aplicación, tenemos que ser muy cuidadosos con respetar la línea gráfica y funcional.

Muchas veces, el diseño del resto de pantallas de una aplicación y el del *Look&Feel* no son hechos por la misma persona, así que, si no es así, es importante poner énfasis en un buen traspaso de la información.

A partir de ahí, el mayor reto es que el diseñador que habrá asumido la responsabilidad de desarrollar las pantallas a partir de un diseño hecho ponga todo el esfuerzo y dedicación. Creemos que con metodologías ágiles podemos conseguir una mayor cohesión entre el equipo y lograr esas pequeñas cosas que pueden suceder en un proyecto.

En esta fase, vamos crear todas las pantallas necesarias para que los maquetadores puedan tener todos los elementos para montar la aplicación en las tres plataformas. Ejecutaremos las pantallas distribuidoras y especiales, como la de contactos, la de detalle o la de perfil.



2.5.7. Iteración 7: maquetación mobile

Como vamos comentando en todo el caso práctico, este es un proyecto que tiene hitos con el cliente y que se basa en las metodologías ágiles, por lo que tendremos al cliente al corriente de cómo va avanzando el proyecto.

La parte de la maquetación no será una excepción, así que también iremos mostrando la evolución de esta en los diferentes *sprints*.

La parte de maquetación la podremos ir avanzando una vez tengamos aprobado el diseño principal, puesto que nos podremos basar en los estilos generales y guiarnos con los *wireframes* desarrollados previamente.

En este caso, solo nos encargaremos de la producción parcial de la aplicación en iPhone.

En este caso, podemos producir la aplicación de varias maneras:

1) Primera opción

Desarrollarlo directamente con el lenguaje nativo de Apple, el Xcode.

Lectura complementaria

Para más información sobre el XCode:

http://developer.apple.com/xcode/

Xcode

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con el Mac OS X. Xcode trabaja conjuntamente con Interface Builder, una herencia de NeXT, una herramienta gráfica para la creación de interfaces de usuario.

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Objective-C, Objective-C++, Java y AppleScript mediante una amplia gama de modelos de programación, incluyendo, pero no limitado a Cocoa, Carbon y Java. Otras compañías han añadido soporte para GNU Pascal, 1 Free Pascal, 2 Ada y Perl.3.

Extraído de la Wikipedia

En este caso, no entraríamos en el desarrollo de la aplicación, ya que los diseños los adaptarían directamente los programadores en el lenguaje nativo de Apple.

2) Segunda opción

En la actualidad, hay muchos desarrolladores que usan algunos *frameworks* Javascript para desarrollar la aplicación en formato web (*WebApp*) y después hay programas que transforman esta *WebApp* en lenguaje nativo para móvil.

Cuando se usa un *framework* para crear una *WebApp* se hace una maquetación con HTML5 y CSS para después utilizar el *framework* para emular las funcionalidades nativas de móvil.

Framework

La palabra inglesa *framework* define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.

En el desarrollo de software, un *framework* o infraestructura digital es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, basándose en la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir apoyo de programas, bibliotecas y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Extraído de la Wikipedia

Los *frameworks* más comunes son los siguientes:

- JqueryMobile
- SensaTouch

Y las aplicaciones para transformar de *WebApp* a lenguaje nativo para cualquier dispositivo móvil más comunes son estas:

- PhoneGap
- Titanium Appcelerator

Estas aplicaciones van bien para proyectos sencillos de poca complejidad. Para desarrollar una aplicación más compleja, habría que desarrollarlo directamente con lenguaje nativo para cada una de las plataformas.

En este caso, el cliente nos pedía los HTML y CSS para poder desarrollar la aplicación con SensaTouch y PhoneGap, así que nosotros nos tuvimos que encargar de maquetar por móvil las plantillas.

El ejercicio aquí consistirá en crear la maquetación de las pantallas que tenemos diseñadas para pasar a los programadores para que implementen el *framework* y lo transformen en lenguaje nativo con las aplicaciones que hemos mostrado antes.

2.5.8. Iteración 8: entrega final del proyecto

Esta fase siempre es la más sencilla en un proyecto cuando utilizamos metodologías ágiles, puesto que el cliente sabrá exactamente qué se le entregará. Siempre habrá un margen de error, pero con este tipo de metodologías será muy pequeño.

En esta fase, nos encargaremos de corregir estos pequeños errores que puedan surgir y será cuando daremos por finalizado el proyecto.

2.6. Enlaces de interés

http://es.wikipedia.org/wiki/wikipedia:Portada

http://cssiphone.com/

http://www.mobileawesomeness.com/

http://www.refinedmobile.com/

http://tapfancy.com/

http://www.tappgala.com/

http://www.iosinspires.me/

http://www.useit.com/

http://www.smashingmagazine.com/

http://www.alvit.de/handbook/

http://skout.co.za/

http://www.bluevertigo.com.ar/

http://developer.apple.com/xcode/

http://jquerymobile.com/

http://www.sencha.com/products/touch

http://phonegap.com/

http://www.appcelerator.com/

3. Diseño de una interfaz para smart TV

3.1. Introducción

Con Apple y Google trabajando con sus *smart TV*, creemos que este caso será bastante útil para los estudiantes que os estéis planteando especializaros en diseño y desarrollo de aplicaciones para televisión interactiva.

Cuando se trata de crear un diseño de una aplicación para televisión interactiva, deberemos tener en cuenta todos los principios de interacción igual que cuando trabajamos en web o en aplicaciones móviles, pero tenemos que pensar que la experiencia del usuario con la interfaz será más limitada, debido a que el usuario interactuará con un mando de televisión.

Por eso, tendremos que plantear una interfaz donde la navegación será más arcaica, donde el usuario solo podrá ir arriba, abajo, a la derecha y a la izquierda.

Televisión inteligente

La televisión inteligente (*smart TV* o también traducido como *televisión híbrida*) describe la integración de Internet y de las características Web 2.0 a la televisión digital y al *set-top box*, así como la convergencia tecnológica entre los ordenadores y estos televisores y el STB. Estos dispositivos se centran en los medios interactivos en línea, en la televisión por Internet y en otros servicios como el vídeo a la carta.

La tecnología de las *smart tv* no solo se incorpora a los aparatos de televisión, sino también a otros dispositivos como la *set-top box*, grabador de vídeo digital, reproductores Blu-ray, consolas de videojuegos y *homecinemas*, entre otros. Estos dispositivos permiten a los espectadores buscar y encontrar vídeos, películas, fotografías y otros contenidos *online* en un canal de televisión por cable, en un canal de televisión por satélite o almacenado en un disco duro local.

Extraído de la Wikipedia

3.2. Objetivos

Como en los demás casos, el objetivo de este ejemplo será desarrollar un proyecto desde el planteamiento inicial hasta el *mockup* de la interfaz. Por lo tanto, después de leer el caso, seréis capaces de plantear un proyecto de este tipo, poniendo énfasis en las metodologías ágiles.

Así pues, veremos, a partir de un *briefing*, cómo definir la arquitectura de información, los *wireframes*, encontrar un *Look&Feel* adecuado y cómo ejecutar un *mockup*. En este caso específico, pondremos énfasis también en la interacción con el mando y la iconografía, que adoptará un papel muy importante en este tipo de interfaz.

Lecturas complementarias

Encontraréis más información sobre los *mockups* en los enlaces siguientes:

http://en.wikipedia.org/wiki/mockup

http://www.interaction-design.org/encyclopedia/mock-ups.html

3.3. Equipo

Como en la mayoría de proyectos que aparecen hoy en día (móvil, televisión, Internet), donde la tecnología nos permite evolucionar más rápido de lo que podemos, es imposible encontrar profesionales con veinte años de experiencia en desarrollo de proyectos de este tipo. Lo que sí encontramos son profesionales autodidactas con conocimientos flexibles.

A continuación, definimos el equipo de producción:

- *Product owner*
- *Scrum master*
- UX/UI designer
- IxD/visual designer
- *Flash developer*

Este proyecto acabará con un *mockup* de la interfaz, así que necesitaremos un *flash developer* para simular la aplicación.

3.4. *Briefing*

Se trata de un caso real que se desarrolló dentro de una empresa que tenía un producto ya de televisión por cable, donde ofrecía un servicio de EPG y videoclub.

Se necesitaba una evolución del producto y migrar hacia una *smart TV*, así que el *briefing* se planteó internamente viendo cuáles eran las posibilidades y qué funcionalidades se querían aplicar.

Así pues, se quiso migrar lo que teníamos hacia una interfaz donde el usuario fuera capaz de:

- crear una cuenta de usuario,
- crear un recomendador de series y películas,
- seleccionar los programas preferidos,
- tener una cartelera de películas y series, con las funcionalidades de comprar o alquilar,
- grabar programas en directo, series o películas preferidas,
- mejorar la navegación y la experiencia del usuario,
- obtener un *Look&Feel* más actual utilizando iconografía,
- acceder a Internet, e

- incorporar publicidad.

Guía electrónica de programas

La guía electrónica de programas (*electronic programming guide* o EPG) es un servicio televisivo que informa de manera rápida y sencilla de toda la programación de un canal de televisión o de radio. El servicio funciona solo en emisiones digitales como la TDT y la radio digital DAB. Representa la evolución a la era digital del tradicional servicio de programación que nos ofrece el teletexto. En el caso de la radio, representa una novedad, puesto que en la tecnología más antigua, como era el RDS, no había un servicio que desempeñara la misma función. Algunos campos típicos pueden ser, en lo referente al contenido, el título, fecha, datos PDC, género, edad mínima, sinopsis, directo o diferido, *copyright*, subtítulos de teletexto, banda sonora, idioma original. En cuanto a las características de la emisión, se puede informar sobre canal, horas de comienzo y final, PALplus, sistema de encriptación, programa repetido, formato del sonido, formato televisivo e idioma.

Extraído de la Wikipedia

3.5. Iteraciones

3.5.1. Iteración 1: análisis del proyecto

Para llevar a cabo un análisis de un proyecto de este tipo, analizaremos otras *smart TV* que haya en el mercado. Las primeras que tendríamos que mirar son las de Apple y Google, ya que es conocido por todo el mundo que son las empresas pioneras en este mercado.

En la actualidad, estas dos todavía no han salido al mercado, pero sí podremos mirar televisiones interactivas como las siguientes:

- Samsung Smart TV platform,
- Sony Bravia Internet Video smart TV platform,
- Panasonic Viera Connect/Smart Viera platform,
- LG Smart TV platform,
- Toshiba Places online TV platform,
- Philips NetTV platform, o
- Sharp AQUOS Net smart TV platform.

Una vez analizadas las características de estas aplicaciones, podremos centrarnos en el desarrollo de la nuestra, tomando las *bestpractices* que se ajusten a nuestros objetivos.

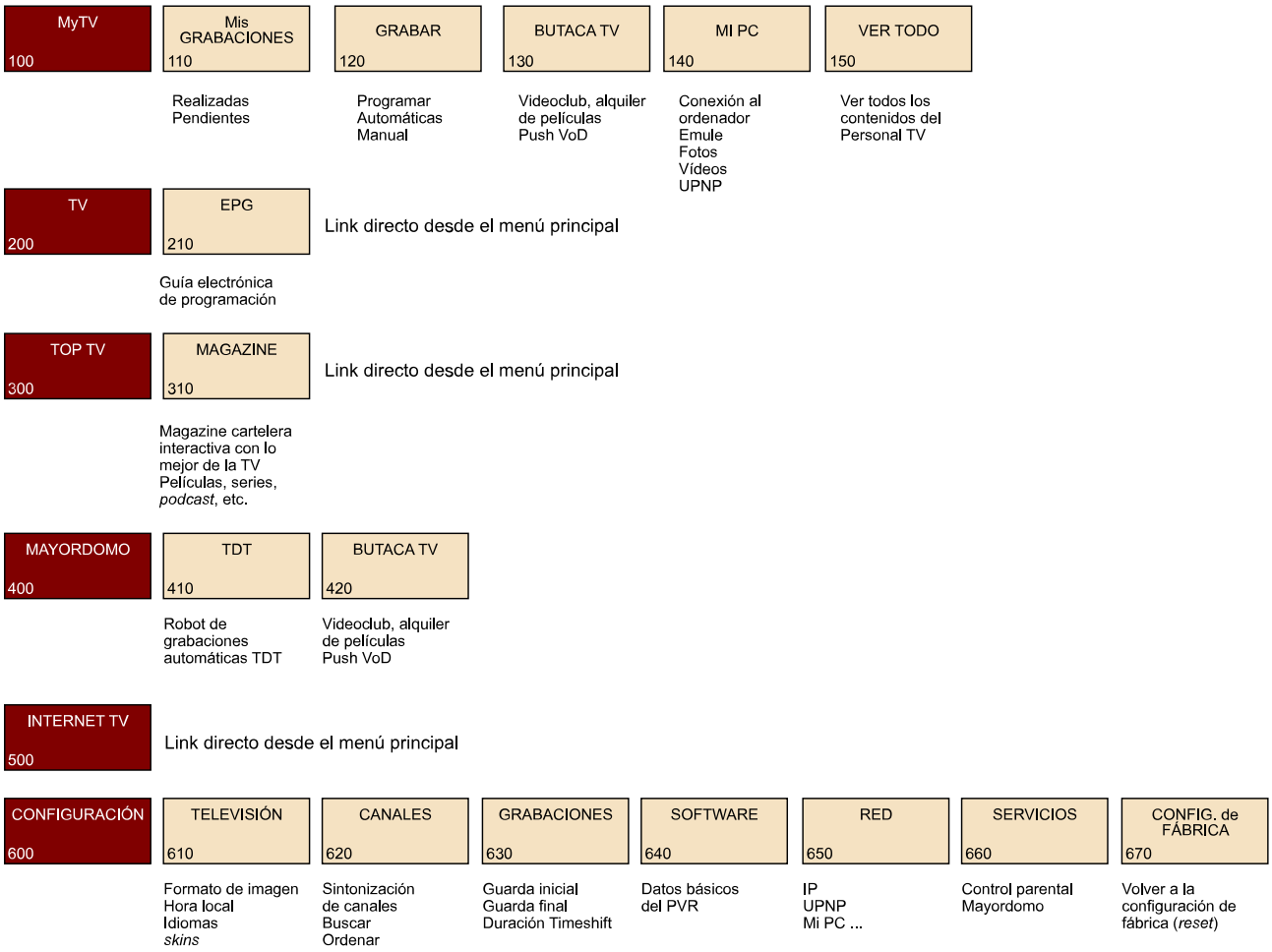
3.5.2. Iteración 2: arquitectura de la información

En este trabajo, nos tendremos que reunir con los directores de producto (los clientes) para definir exactamente cuáles queremos que sean las funcionalidades y así poder jerarquizar y estructurar la información.

Tal y como hemos dicho en la introducción, deberemos tener en cuenta la limitación del usuario para acceder a la información utilizando un mando a distancia. Así pues, una vez definidos todos los contenidos que aparecerán en la interfaz, el UX/UI *designer* se encargará de estructurar la información.

El ejercicio aquí será plantear una arquitectura siguiendo el *briefing* obtenido.

A continuación, os mostramos un ejemplo de cómo podría ser esta arquitectura.



3.5.3. Iteración 3: *wireframing*

Como ya hemos indicado, en este caso el cliente serán los directores de área, pero el procedimiento de metodologías ágiles que aplicamos será exactamente el mismo que en los demás casos. Haremos iteraciones igualmente.

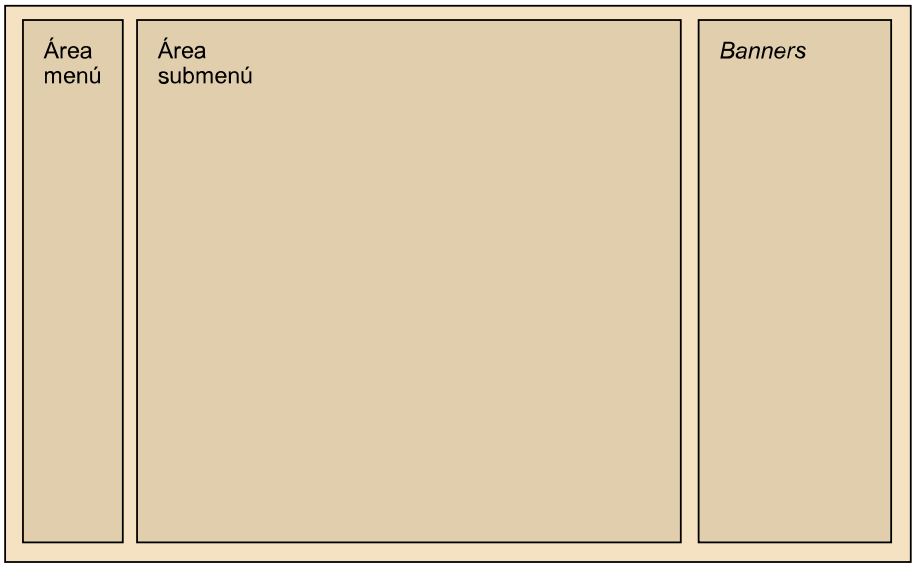
Una vez llegados a este punto, habremos aprobado la arquitectura de información de nuestra nueva *smart TV*, así que procederemos a ejecutar los *wireframes* o esqueleto para visualizar cómo irán estructuradas cada una de las pantallas.

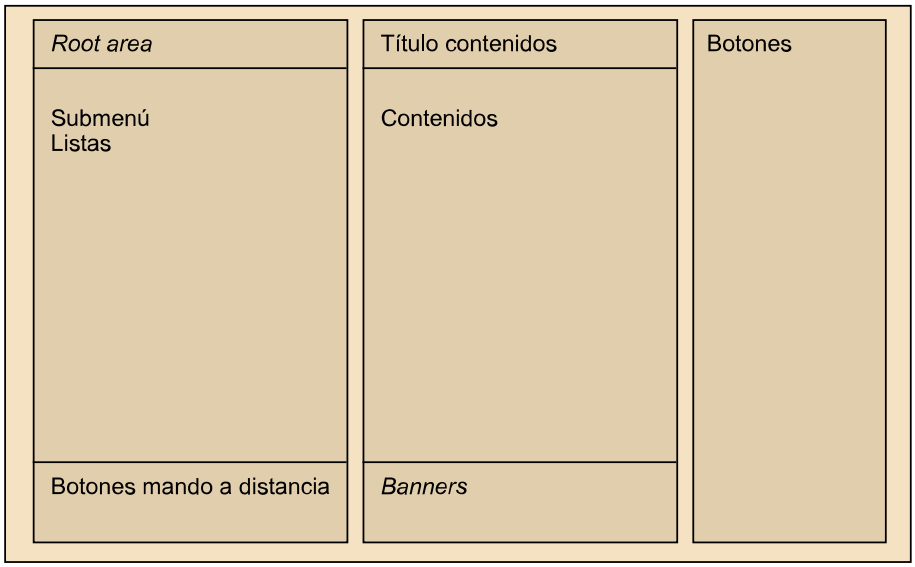
Hay un factor muy importante que debemos tener en cuenta cuando se desarrollan *wireframes* de cualquier tipo. Ser económicos. Es decir, debemos intentar que en los mínimos *wireframes* posibles pudiera caber toda la información que queremos.

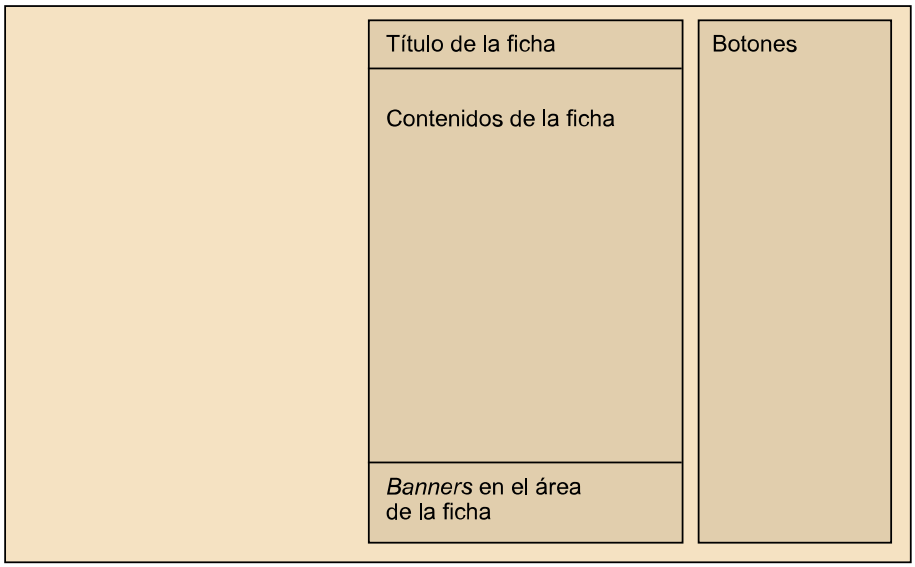
Este es un punto importantísimo para la rentabilización de un proyecto, puesto que ahorraremos recursos tanto en diseño como en desarrollo posterior.

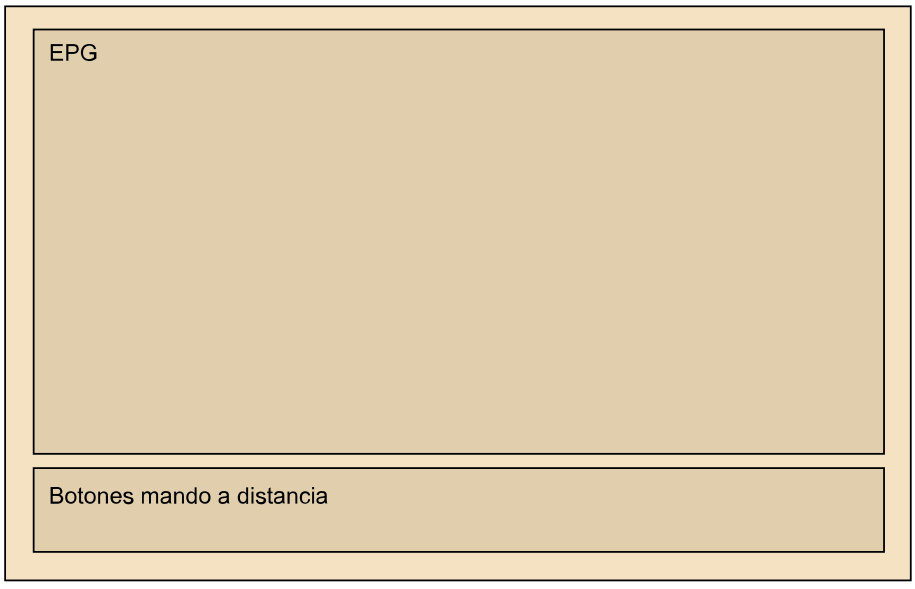
> El ejercicio aquí será, con los mínimos *wireframes* posibles, intentar plantear toda la nueva interfaz.

A continuación, os mostramos un ejemplo de cómo podrían ser estos *wireframes*.




Root area
Título contenidos
Botones
Submenú
Listas
Contenidos
Botones mando a distancia
Banners


Título de la ficha
Botones
Contenidos de la ficha
Banners en el área
de la ficha


EPG
Botones mando a distancia

3.5.4. Iteración 4: mando a distancia

Un punto básico también a la hora de crear la experiencia de usuario será el diseño del mando a distancia, por el que tendremos que dotar al mando de las funcionalidades que queremos que el usuario aplique en la interfaz.

Mando a distancia

Un **control remoto** o **mando a distancia** es un dispositivo electrónico usado para ejecutar una operación remota (o telecomando) sobre una máquina.

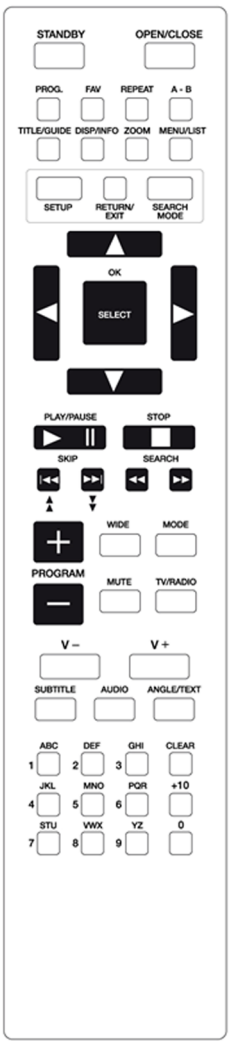
El término se emplea generalmente para referirse al mando a distancia (llamado en general simplemente *el mando* o, en Latinoamérica, *el control*) para el televisor u otro tipo de aparato electrónico propietario, como DVD, Hi-Fi, ordenadores, y para encender y apagar un interruptor, la alarma o abrir la puerta del estacionamiento. Los mandos a distancia para estos aparatos son normalmente pequeños objetos (fácilmente manipulables con una mano) con una matriz de botones para ajustar los diferentes valores, como por ejemplo, el canal de televisión, el número de canción y el volumen. De hecho, en la mayoría de dispositivos modernos el mando contiene todas las funciones de control, mientras que el propio aparato controlado solo dispone de los controles más primarios. La mayoría de estos controles remotos se comunican con sus respectivos aparatos vía señales de infrarrojo (ANAR) y solo unos pocos utilizan señales de radio. En los vehículos modernos, las clásicas claves incorporan ahora mandos a distancia con varias funciones. Su fuente de energía suele ser pequeñas pilas de tipo AA, AAA o de botón.

Extraído de la Wikipedia

En la actualidad, los mandos a distancia tienen muchas funcionalidades que de hecho no utiliza prácticamente nadie.

Muchas veces, crear un mando a distancia no estándar es un coste que una empresa no se puede permitir y se opta por tomar un mando del mercado que se ajuste a las funcionalidades que planteamos en la nueva interfaz.

El ejemplo que os mostramos a continuación es un mando estándar, que no respeta las características que buscamos para nuestra aplicación, puesto que tantas funcionalidades distraerán al usuario.



El ejercicio aquí será crear un mando que incorpore las necesidades que realmente tenemos.

3.5.5. Iteración 5: *Look&Feel*

Esta es la parte más delicada donde definiremos la gráfica y el estilo de la aplicación. Cuando trabajemos con televisión interactiva adoptaremos un poco los principios de la interacción en tabletas o dispositivos móviles.

A diferencia del uso de las tabletas y aplicativos móviles, el usuario se sitúa lejos de la pantalla, pero igualmente tendremos que crear iconografía, utilizar botones grandes, letra grande y diferenciar muy bien la jerarquía de la información.

Aparte de esto, en televisión interactiva hay otro factor que cobra importancia, y es saber diferenciar muy bien el área de navegación y de interacción del usuario con la reproducción de vídeo que haya en aquel momento. Por eso, tendremos que delimitar bien las áreas y podemos utilizar transparencias, animaciones, colores e imágenes.

Así pues, el ejercicio aquí será ejecutar un diseño atractivo y funcional para la nueva interfaz, una vez definida la arquitectura de información y los *wireframes*.

3.5.6. Iteración 6: iconografía

Para cualquier interfaz, actualmente es muy importante utilizar la iconografía, pero en este caso creemos que es vital.

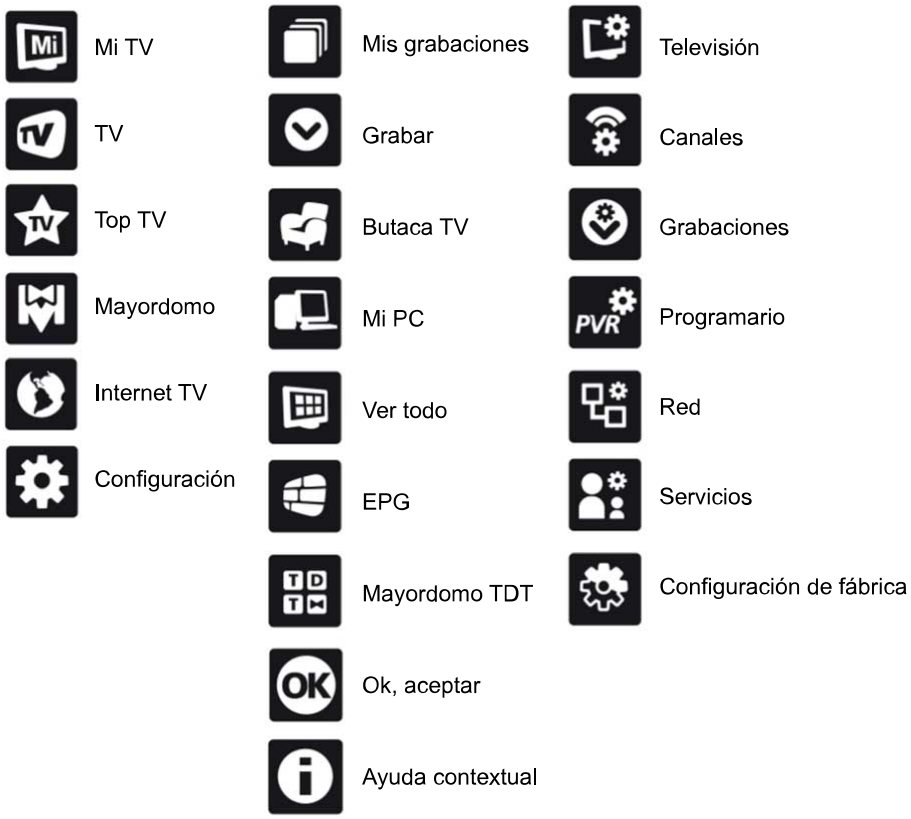
Y decimos vital porque, tal y como hemos indicado antes, el usuario se dispone lejos de la pantalla, con un mando que lo limita bastante y cualquier cosa que no sea motivar su interacción hará que rápidamente la rechace.

Así pues, la iconografía será una ayuda más y será necesario que sea muy intuitiva.

Como apunte recalcaremos que, en cuanto a la usabilidad, la iconografía, a no ser que esté normalizada, siempre irá acompañada de texto descriptivo.

El ejercicio aquí será crear una iconografía de acuerdo con el *Look&Feel* que hayamos definido antes.

A continuación, os mostramos un ejemplo de cómo podría ser esta iconografía.


Mi TV
TV
Top TV
Mayordomo
Internet TV
Configuración
Mis grabaciones
Grabar
Butaca TV
Mi PC
Ver todo
EPG
Mayordomo TDT
Ok, aceptar
Ayuda contextual
Televisión
Canales
Grabaciones
Programario
Red
Servicios
Configuración de fábrica


Película
Podcast
Programa TV
Película butaca
Grabación
¡Atención!
Información
Configuración
Control parental
Espera
Película
Tráiler
Destacado
Programa TV
Podcast
Contenedor
Grabación protegida
Grabación automática
de mayordomo TDT

3.5.7. Iteración 7: *mockup*

Como hemos dicho antes, un *mockup* es un ejemplo bastante real de la aplicación. En este caso, creemos que el mejor software para desarrollarla es Flash, ya que nos permite manipular la interacción y reproducir vídeo.

Con el Flash, no habrá la interacción con el mando a distancia, pero podremos ver ante un ordenador cómo funciona la interfaz.

El *mockup* se utilizará principalmente para poder realizar tests de usuario, donde utilizaremos voluntarios o personas de diferentes rangos sociales y edades, por ejemplo, para poder mejorar la *smart* TV.

¿Cuál es el método de test con usuarios?

Es una prueba de usabilidad que se basa en la observación y análisis de cómo un grupo de usuarios reales utiliza el sitio web, anotando los problemas de uso con los que se encuentran para poder solucionarlos con posterioridad. Se trata de una prueba llevada a cabo en **laboratorio**, es decir, no tenemos que confundirla con un estudio de campo.

Cuándo llevarlo a cabo

Cuanto más tarde, peor.

Como toda evaluación de usabilidad, cuanto más esperemos para su ejecución, más costosa resultará la reparación de los errores de diseño descubiertos.

Esto quiere decir que no solo tenemos que realizar este tipo de pruebas sobre el sitio web una vez implementado, sino también, sobre los prototipos del lugar.

Siempre después de una evaluación heurística.

Si llevamos a cabo un test con usuarios sin haber realizado previamente una evaluación heurística, probablemente prestaremos demasiada atención a problemas de uso que encuentren los participantes que podrían haber sido descubiertos con una simple evaluación heurística.

Un test con usuarios es más costoso que una evaluación heurística, por lo que sería malgastar tiempo y dinero utilizarlo para descubrir errores de diseño motivados por el incumplimiento en el diseño de principios generales de usabilidad (heurísticos).

Extraído de http://www.nosolousabilidad.com/articulos/test_usuarios.htm

Lectura complementaria

Encontraréis más información sobre la evaluación heurística en el enlace siguiente:
http://www.nosolousabilidad.com/articulos/heuristica.htm

3.5.8. Iteración 8: entrega final del proyecto

En la parte del *mockup*, donde llevaremos a cabo los tests de usuario, seguramente tendremos que hacer varias repeticiones, hasta que el producto esté optimizado.

Una vez lo habremos conseguido, trasladaremos todo el material a los desarrolladores, que se encargarán de programar el aplicativo.

3.6. Enlaces de interés

http://es.wikipedia.org/wiki/wikipedia:Portada

http://www.interaction-design.org/

http://www.nosolousabilidad.com/

4. Desarrollo de una aplicación móvil para sistemas Android

4.1. Introducción

El proyecto consistirá en desarrollar una aplicación móvil para el sistema Android. La aplicación presentará el mismo contenido que un web ya existente. Esto incluirá el contenido de las propias páginas, un sistema de búsqueda textual y un sistema de votación de contenidos.

Mediante la aplicación, se podrá acceder a los contenidos desde móviles y tabletas con sistema operativo Android.

4.2. ¿Qué es Android?

Android es el sistema operativo de Google para móviles. Es un sistema basado en Linux. Se utiliza para sistemas móviles de tipo *smartphone* y tabletas.

Android forma parte de la Open Handset Alliance, una agrupación de empresas fabricantes de hardware para dispositivos móviles.

4.3. ¿Por qué Android?

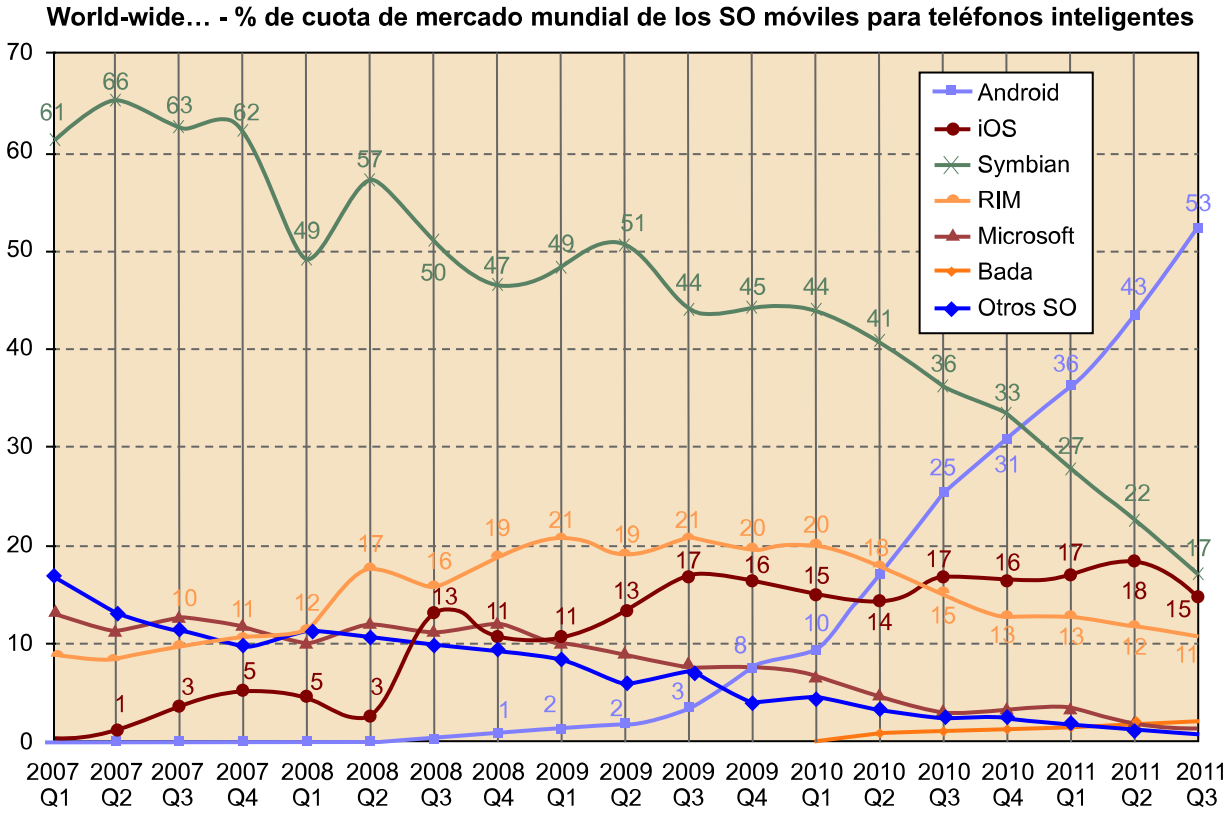
4.3.1. Ventajas

Actualmente el sistema Android es el que tiene mayor cuota de mercado en el ámbito de los *smartphones*. Existen varias empresas que producen dispositivos Android y su evolución no está determinada por una de ellas en exclusiva.

Apple

En el caso de Apple, solamente Apple fabrica y desarrolla el iPhone y todo su entorno.

La gráfica siguiente muestra la evolución de esta progresión por cuatrimestres (datos mundiales):



Fuente: Gartner. Datos cuatrimestrales. Disponible en: http://www.gartner.com/it/page.jsp?id=1924314

Lectura complementaria

Encontraréis más información sobre los sistemas operativos para móviles en:

http://en.wikipedia.org/wiki/Mobile_operating_system

Cuatrimestre	An-droid	iOS	Sym-bian	RIM	Micro-soft	Bada	Otro
2011 Q4	50,9%	23,9%	11,7%	8,8%	1,9%	2,1%	0,8%
2011 Q3	52,5%	15,0%	16,9%	11,0%	1,5%	2,2%	0,9%
2001 Q2	43,4%	18,2%	22,1%	11,7%	1,6%	1,9%	1,0%
2001 Q1	36,0%	16,8%	27,4%	12,9%	3,6%	1,7%	1,6%
2010 Q4	31,1%	16,1%	32,9%	13,1%	3,4%	1,3%	2,2%
2010 Q3	25,3%	16,6%	36,3%	15,4%	2,8%	1,1%	2,5%
2010 Q2	17,2%	14,2%	41,2%	18,2%	5,0%	0,9%	3,3%
2010 Q1	9,6%	15,3%	44,2%	19,7%	6,8%		4,4%
2009 Q4	7,6%	16,2%	44,7%	19,7%	7,9%		4,0%
2009 Q3	3,4%	17,0%	44,2%	20,5%	7,9%		7,0%
2009 Q2	1,8%	13,0%	51,0%	19,0%	9,3%		5,9%
2009 Q1	1,6%	10,5%	48,8%	20,6%	10,2%		8,2%
2008 Q4	1,1%	10,6%	46,5%	19,3%	12,2%		9,1%

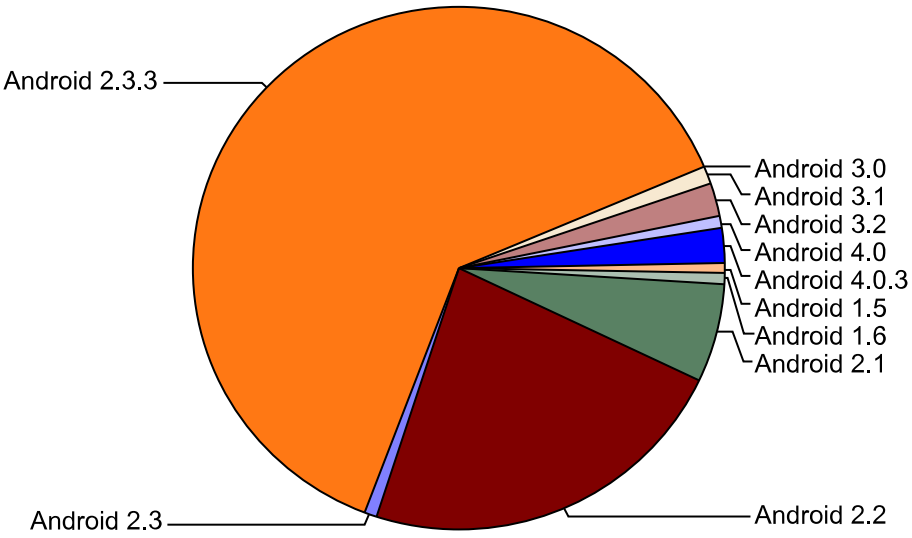
Fuente: Gartner. Datos cuatrimestrales. Disponible en http://www.gartner.com/it/page.jsp?id=1924314

Cuatrimestre	Android	iOS	Symbian	RIM	Microsoft	Bada	Otro
2008 Q3	0,6%	13,1%	50,3%	16,1%	11,2%		9,8%
2008 Q2		2,8%	57,5%	17,5%	12,1%		10,8%
2008 Q1		4,6%	49,5%	11,6%	10,4%		11,6%
2007 Q4		5,2%	62,3%	10,9%	11,9%		9,6%
2007 Q3		3,4%	63,1%	9,7%	12,8%		11,5%
2007 Q2		1,0%	65,6%	8,9%	11,5%		13,0%
2007 Q1			61,2%	8,7%	13,4%		16,8%

Fuente: Gartner. Datos cuatrimestrales. Disponible en http://www.gartner.com/it/page.jsp?id=1924314

4.3.2. Inconvenientes

Debido a la gran cantidad de fabricantes y de dispositivos que utiliza Android, existe una mayor fragmentación de este tipo de dispositivos. Cada uno de estos dispositivos puede tener diferentes resoluciones, tener o no tener cámara y muchas otras diferencias de funcionalidad.



Fragmentación entre versionesFuente: Android *developers*, disponible en: http://developer.android.com/resources/dashboard/platform-versions.html

Al existir esta diversidad de funcionalidades, será responsabilidad del desarrollador distinguir y testear si estas están disponibles y en qué medida.

En una plataforma más cerrada y con un único fabricante, podemos estar más seguros de las capacidades del dispositivo final que ejecutará la aplicación.

Videojuegos

Una analogía con lo que sucede en el mercado de *smartphones* la podemos ver en el tradicional mercado de videojuegos entre PC y videoconsolas. En gran medida esto ya ha cambiado, pero históricamente el desarrollador para el mercado de ordenador personal tiene que gestionar muchas posibles combinaciones de tarjetas gráficas y configuraciones, mientras que al desarrollar para una videoconsola específica se puede aprovechar al máximo el hardware, que es el mismo para todos los usuarios. Este ha sido siempre uno

de los atractivos de las videoconsolas para usuarios que simplemente quieren jugar, sin preocuparse de configurar sus sistemas o de enfrentarse a posibles incompatibilidades.

Otro posible inconveniente que debemos tener en cuenta es que todos los estudios apuntan a que los usuarios del iPhone compran más aplicaciones que los de Android, aunque esto puede verse compensado por la mayor base de usuarios de Android.

4.4. Opciones para desarrollar en Android

En última instancia, las aplicaciones instalables desde el Android Market están desarrolladas en código Java. No obstante, podemos generar esa aplicación desarrollando directamente el código fuente en Java o utilizando alguna otra plataforma que genere por nosotros ese código.

Presentamos a continuación dos de las opciones más populares:

Más herramientas

Algunas otras herramientas y entornos, en este caso especialmente para el desarrollo de videojuegos, son Unity y Moai.

1) Nativas. Desarrollar la aplicación en Java conseguirá el máximo control y rendimiento sobre la plataforma Android. Por otro lado, existe un gran número de desarrolladores en Java, por lo que, una vez conocida la API de Android y su entorno, es más fácil para ellos familiarizarse con el entorno de desarrollo en esta plataforma.

2) Phonegap. Permite desarrollar una aplicación en html y Javascript. Esta plataforma nos permite aprovechar los conocimientos en el desarrollo web para desarrollar aplicaciones independientes. Esta herramienta permite, además, exportar a múltiples sistemas operativos, además de Android, con una sola base de código. Para que esto funcione, habrá que tenerlo en cuenta desde el inicio del desarrollo del proyecto.

Algunas de las plataformas soportadas por Phonegap son las siguientes:

- iOS,
- Android,
- Blackberry,
- Windows Phone,
- WebOS, y
- Symbian.

4.5. Consideraciones para el desarrollo

4.5.1. Lenguaje

Las aplicaciones en Android utilizan como lenguaje de desarrollo el lenguaje de programación Java.

Lenguaje Java

Técnicamente, la máquina virtual desarrollada por Google se llama Dalvik y es diferente de la máquina virtual estándar de Java (JVM), aunque a efectos del desarrollo del proyecto podemos ignorar esto de forma segura y decir, simplemente, que utiliza el lenguaje Java.

De este modo, el modelo de desarrollo para Android es familiar para multitud de desarrolladores.

4.5.2. Instalación del entorno

Para desarrollar para un sistema Android, necesitamos instalar el ***system development kit*** correspondiente. Lo podemos encontrar en http://developer.android.com/sdk/index.html.

El SDK de Google está desarrollado en Java y requiere, por lo tanto, del JDK[1] para ejecutarse. Podemos descargar la última versión del JDK en http://www.oracle.com/technetwork/java/javase/downloads/index.html.

(1)Acrónimo de *Java development kit*.

Para el desarrollo de aplicaciones Android, utilizaremos una IDE[2](permite editar código, gestionar las versiones, compilar, introducir cambios de estructura y otras funcionalidades desde un solo programa) y se utiliza mucho en el desarrollo en Java. Se trata de **Eclipse** y podemos descargarla en www.eclipse.org/downloads/. La versión recomendada es la Classic.

(2)Acrónimo de *integrated development environment*.

Otros editores

Podemos utilizar otros editores o herramientas para gestionar un proyecto para Android. Dos editores muy conocidos para programar, pero con una dificultad de entrada muy alta, son por ejemplo:

1) Emacs:

- http://www.gnu.org/software/emacs/
- http://emacswiki.org/

2) VIM:

- http://en.wikipedia.org/wiki/Vim_%28text_editor%29
- http://www.vim.org/

El uso de editores como estos es diferente al de un IDE en varios aspectos, pero es posible incorporarlos en el flujo de desarrollo de una aplicación Android.

Aquí nos centramos en la opción basada en Eclipse por ser una de las más sencillas como introducción y la recomendada por Google. Si se quieren utilizar herramientas más independientes y sencillas y controlar cada uno de sus componentes habrá que utilizar directamente las utilidades que podemos encontrar en el SDK de Google.

Google nos proporciona utilidades de línea de comandos para gestionar todos los aspectos del desarrollo de forma propia.

Algunas buenas referencias sobre estos comandos las encontramos en:

- http://developer.android.com/guide/developing/building/building-cmdline.html
- http://developer.android.com/guide/developing/tools/android.html
- http://developer.android.com/guide/developing/devices/emulator.html
- http://developer.android.com/guide/developing/tools/adb.html

La ventaja de aprender a utilizar estas utilidades es que permiten la integración de estas funcionalidades en el flujo de trabajo. Pueden ser incorporadas en *scripts* propios y en las

propias herramientas, por lo que permiten trabajar de forma más cómoda. En cualquier caso, esto supone una inversión inicial o continua en pulir el flujo de desarrollo que no es apta para todos los equipos de desarrollo.

En este punto, tenemos ya la máquina de Java, Eclipse y el SDK de Google. Necesitamos también el *plugin* ADT[3] para Android. Este nos hará accesibles desde Eclipse las funcionalidades del SDK.

[3]Acrónimo de *Android development tools*.

ADT

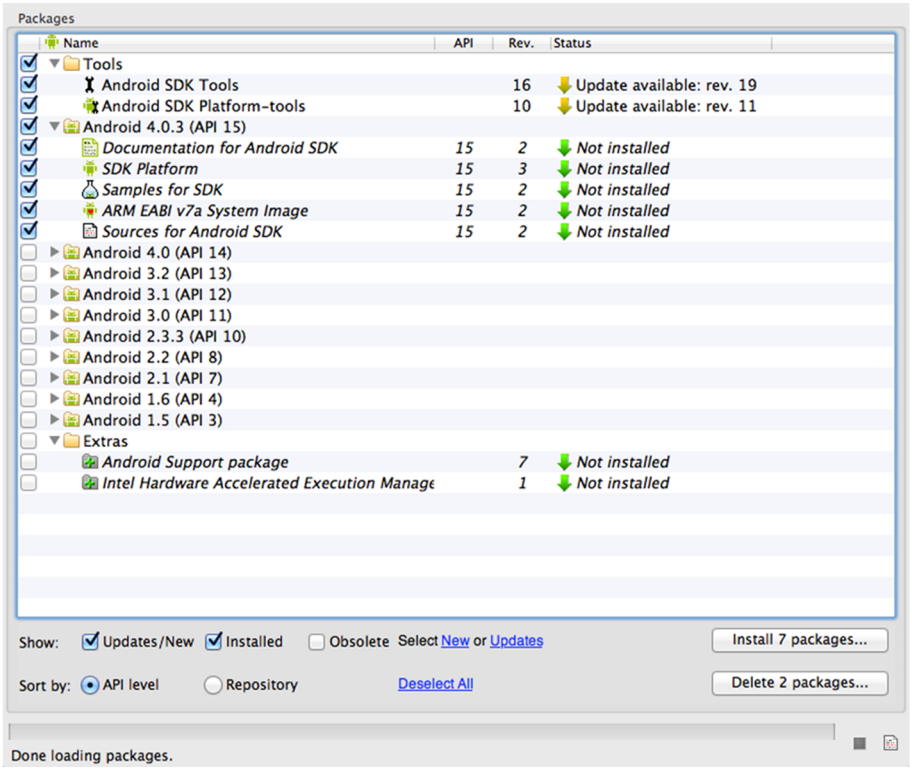
El proceso de instalación de este *plugin* es el mismo que todos los *plugins* de Eclipse. Podemos ver una guía paso a paso en http://developer.android.com/sdk/eclipse-adt.html#installing.

Con el *plugin* instalado, solamente nos falta instalar las plataformas sobre las que vayamos a trabajar y otros extras. Podemos instalar diferentes versiones del sistema operativo Android para testear la aplicación en múltiples versiones o ejecutarlas en el emulador.

Para ello, desde Eclipse accedemos a **Windows** > **Android SDK Manager**.

Si estamos utilizando las utilidades proporcionadas por el SDK sin utilizar Eclipse, simplemente ejecutamos el comando **Android**, accesible dentro de la carpeta **Tools**.

En cualquiera de los dos casos, se nos presentará un diálogo similar al siguiente, donde podremos seleccionar los paquetes que vamos a instalar.



Podemos instalar las opciones más habituales. Normalmente no son las últimas, sino una o dos versiones anteriores, ya que desde el desarrollo de la versión hasta que salen al mercado (o se actualizan) varios dispositivos con esa versión suele pasar cierto tiempo.

Con este paso ya tenemos las herramientas y el entorno preparados para crear proyectos y publicarlos desde Eclipse.

4.5.3. Testeo

Entre las herramientas que nos proporciona Google, existe un emulador con multitud de opciones para diferentes dispositivos y configuraciones. Aún con todas estas facilidades es extremadamente lento.

Anécdota

Como referencia puramente anecdótica, hemos visto en un sistema con 8 Gb de RAM un emulador con una imagen de un *smartphone* sencillo (ni siquiera tableta) tiempos de arranque de unos cinco minutos.

Recomendamos probar las aplicaciones en dispositivos reales. Idealmente deberemos testearlas en todos los teléfonos donde se puedan instalar. Debido a la dificultad de este proceso, podemos hacerlo al menos en los modelos principales.

Para instalar la aplicación en un dispositivo debemos seguir los pasos siguientes:

1) Definimos la aplicación como **de*bug*gable** en su archivo **manifest**. El archivo se llama **AndroidManifest.xml**. Tenemos que añadir el atributo **android:de*bug*gable="true**" al *tag* `<application>`.

2) En el dispositivo donde queramos probar la aplicación. En **Settings** > **Applications** > **Development** es necesario marcar el *check* **USB De*bug*ging**.

3) Seleccionar el dispositivo desde Eclipse en el diálogo que saltará al conectar el dispositivo.

Si no usamos Eclipse, usaremos el comando ADB para conectarnos al dispositivo.

Web recomendada

http://developer.android.com/guide/developing/tools/adb.html

El comando **adb devices** mostrará una lista de los dispositivos conectados.

En cualquier caso, si se quiere probar la opción del emulador podemos encontrar guías que nos indican los pasos necesarios y las opciones disponibles.

4.6. Visión general

El flujo de una aplicación Android:

Webs recomendadas

Unas guías altamente fiables con información de primera mano son las proporcionadas por Google en:

http://developer.android.com/guide/developing/devices/emulator.html

http://developer.android.com/guide/developing/tools/emulator.html

1) Se desarrolla la aplicación.

2) Testeo en el emulador.

3) Testeo en un dispositivo real.

4) Se publica la aplicación en el Android Market.

4.7. Especificaciones

¿Qué quiere el cliente?

El cliente quiere que su contenido sea accesible al máximo posible de usuarios. Ya tiene una web con sus productos e información. Quiere crear una aplicación para dispositivos móviles.

Idealmente, el desarrollo de esta aplicación móvil no debería suponer cambios estructurales en la página web ni un cambio organizativo muy grande en cuanto a cómo se genera el contenido de la página.

¿Qué se le propone?

Se plantea al cliente la posibilidad de crear una aplicación para iPhone o Android. Se opta por empezar por Android en una primera fase y más adelante desarrollar la aplicación para iPhone si la de Android tiene el éxito esperado.

La aplicación será de descarga gratuita, ya que tiene como objetivo dar visibilidad a la web ya existente y dar acceso al contenido de esta de una forma más conveniente.

4.8. Diseño

4.8.1. Esbozo general

Podemos crear los componentes por código o mediante xml. También existen muchas soluciones útiles que nos ayudan a abstraer estos pasos y nos permiten crear la UI[4].

(4)Acrónimo de *user interface.*

Una de estas utilidades es **droid draw**. Esta aplicación, disponible para varios sistemas, nos permite generar la interfaz de un proyecto Android mediante una **interfaz *drag 'n drop*** (arrastrando y soltando elementos gráficamente).

Podemos realizar unos esbozos de la tipología de páginas, a modo de *wireframe*, de una manera muy simple. En cuanto el nivel de detalle aumenta, merece la pena utilizar herramientas como estas para generar ágilmente las pantallas necesarias.

4.8.2. Diseño final

El diseño de una aplicación nativa es, en muchos casos, más delicado que el de una página web. Los usuarios de una plataforma específica, como lo es Android, esperan que los controles tengan cierto aspecto y se comporten de cierto modo. Esto se debe a que todas las aplicaciones se comportan de forma similar y nos es útil para garantizar la usabilidad de la aplicación.

Existen varios recursos que nos ayudarán a dar coherencia a la aplicación. En su conjunto, estas recomendaciones darán una sensación de familiaridad y de producto bien acabado a la aplicación:

- http://developer.android.com/guide/practices/ui_guidelines/index.html
- http://developer.android.com/design/index.html

4.9. Iteraciones

4.9.1. Iteración 1: definición del flujo de la aplicación

Tenemos una serie de contenidos disponibles en una página web y queremos hacerlos accesibles a través de una aplicación nativa de Android.

La web ya existente consta de las siguientes url:

```
es.weboriginal.com
/
/arbol_de_navegación
/quienes_somos
/servicios/
   -ilustración
   -diseño_web
   -seo
/clientes/
   -casos_de_éxito
   -showcase
      -demo1
      -demo2
      -demo3
      -demo4
/contacto/
   -donde_estamos
   -formulario_de_contacto

en.weboriginal.com
/en
/sitemap
```

```
/about_us
/services/
   -illustration
   -web_design
   -seo
/clients/
   -success_stories
   -showcase
      -demo1
      -demo2
      -demo3
      -demo4
/contact/
   -where_are_we
   -contact_form
```

Como podemos observar, existe contenido en varios idiomas y sigue un árbol de navegación habitual, con página de primer nivel y otras agrupadas en secciones mediante páginas de navegación. El contenido está bien organizado mediante *pretty urls*.

pretty urls

Las *pretty urls* o *search engine friendly urls* son url bien diseñadas y estructuradas de forma correcta.

http://en.wikipedia.org/wiki/Rewrite_engine

Podemos distinguir también las url según el idioma al que pertenecen. La distinción entre idiomas se hace utilizando el subdominio de cada idioma para tener flexibilidad total al definir los nombres de las páginas.

El sitio tiene además funcionalidad de búsqueda y un sistema de votación del contenido. Queremos mantener también estas funcionalidades.

Nota

De otro modo, no podría haber dos páginas con el mismo nombre accesibles de forma directa para ambos idiomas, como en el caso de la página seo.

Definiremos un diseño que respete esta estructura con componentes básicos.

4.9.2. Iteración 2: diseño - esbozo

Con la información que debe ser accesible clara en la iteración anterior, podemos pasar a componer los primeros esbozos del diseño.

Podemos utilizar cualquier herramienta de *wireframes*. En este caso, utilizaremos esbozos tradicionales (a lápiz o digitalmente), ya que solo nos interesa tener clara la distribución básica de los elementos en la aplicación. Más adelante, utilizaremos herramientas que nos ayudarán a generar la interfaz gráfica de forma intuitiva.

Utilizaremos una navegación por *tabs* en la parte superior para implementar el cambio entre idiomas. De este modo, el usuario verá de forma fácil los idiomas disponibles y podrá pasar del uno al otro en cualquier momento.

En la parte superior, o bien de forma oculta dependiendo del estado de la aplicación, utilizaremos un *widget* de búsqueda para permitir la funcionalidad de buscar textualmente en el contenido del sitio.

Para listar las páginas disponibles utilizaremos una *listview*.

> En el enlace podemos ver un ejemplo, si bien está hecho directamente desde Eclipse sin utilizar ninguna herramienta de ayuda, de cómo implementar una vista de tipo lista:
>
> http://developer.android.com/resources/tutorials/views/hello-listview.html

Si seleccionamos sobre una de las páginas de la lista sucederá lo siguiente:

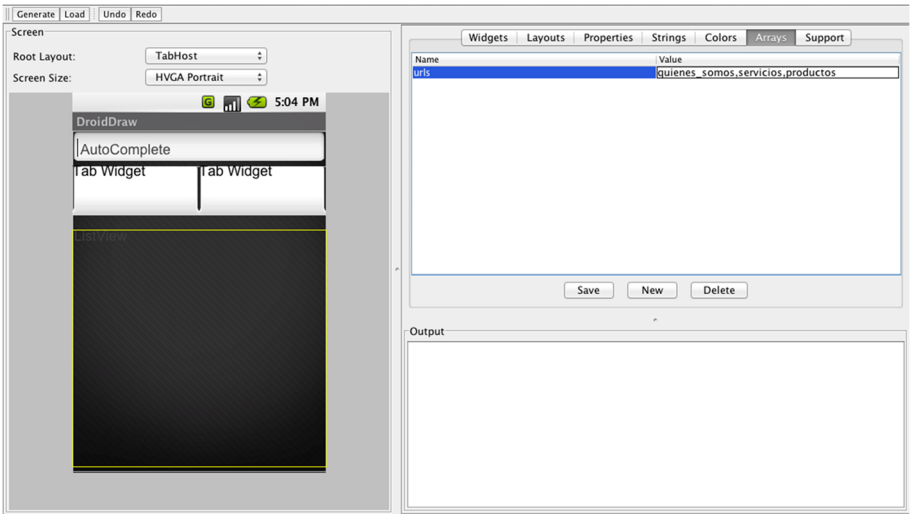
1) La página es una página con contenido. Vemos un botón para volver a la lista anterior. Debajo de ese botón se nos muestra el contenido y, al final del contenido, un sistema de estrellas para votar la calidad del mismo.

2) La página es una página distribuidora. Aparece otra lista que nos muestra el contenido de esa página y un botón para volver al paso anterior.

4.9.3. Iteración 3: diseño - diseño nativo general

Con la estructura definida en la anterior iteración nos disponemos a ejecutar el primer diseño con elementos nativos. Utilizamos la aplicación Droiddraw.

Podemos ver a continuación la disposición de elementos de la página principal:



Para distribuir el contenido como mejor nos convenga utilizamos *layouts*. Podemos anidar los *layouts* de diferentes modos para conseguir diferentes distribuciones del contenido. Aquí, por ejemplo, utilizamos *layouts* horizontales para disponer los *tabs* y *layouts* verticales para disponer, entre sí, el cuadro de búsqueda, el *layout* donde aparecen los *tabs* y la lista de url.

Lecturas complementarias

Aquí podemos ver algunos tutoriales que explican con ejemplos sencillos los *layouts*:

http://www.programacion.com/articulo/introducion_a_los_*layouts*_para_android_400

http://mobiforge.com/designing/story/understanding-user-interface-android-part-1-*layouts*

Para más información, tenemos la documentación oficial, con listados de todas las opciones disponibles:

http://developer.android.com/guide/topics/ui/*layout*-objects.html

http://developer.android.com/guide/topics/resources/*layout*-resource.html

4.9.4. Iteración 4: diseño - diseño nativo página distribuidora

La página distribuidora será prácticamente igual a la página principal. La única diferencia es que, si estamos en una página distribuidora, hemos llegado a través de otra pantalla. Por lo tanto, necesitamos una manera de volver a la pantalla anterior, a la opción superior.

Para ello, vamos a utilizar un simple botón de volver.

La funcionalidad de búsqueda está oculta en el resto de la aplicación. Podemos implementarlo de modo que aparezca a través del menú de la aplicación.

De este modo, al margen de los *tabs* de selección de idioma, no tenemos elementos en la parte superior de la pantalla que sean permanentes. Por otro lado, la mayoría de aplicaciones disponen de esta funcionalidad accesible desde el menú de Android una vez estamos dentro de la aplicación, por lo que cumple con el principio POLS[5].

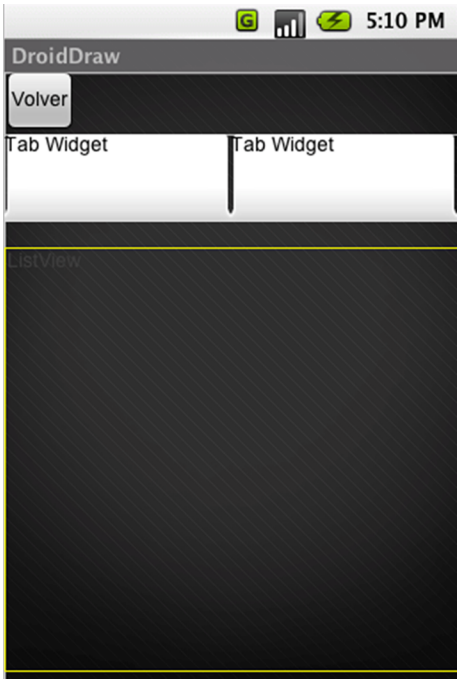
[(5)]Acrónimo de *Principle of Least Surprise*.

POLS

Consiste en que las cosas funcionen de la manera que el usuario espera, es decir, de forma parecida a como el resto de funciones del sistema.

Más información en:

http://en.wikipedia.org/wiki/Principle_of_least_astonishment

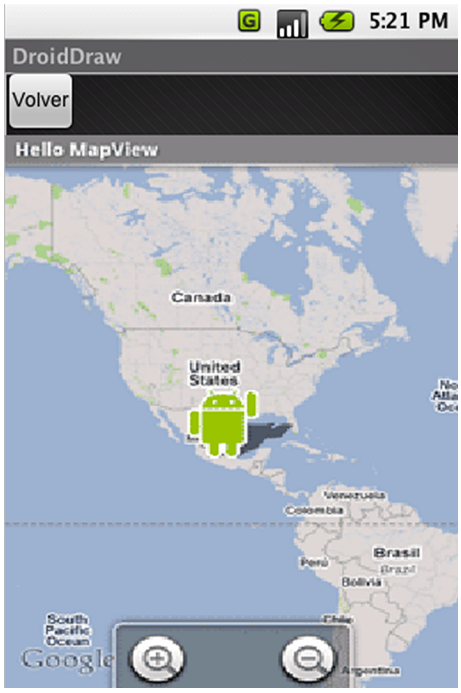


4.9.5. Iteración 5: diseño - diseño nativo contacto

En las iteraciones anteriores, hemos ejecutado el diseño de las páginas de contenido y las de distribución. En esta, diseñamos la página de contacto.

Para ello, utilizamos un *widget* muy sencillo, que dotará de mayor funcionalidad a la aplicación. Con el *widget* de Google Maps tenemos toda la funcionalidad de esta aplicación para mayor facilidad del usuario.

Tampoco nos olvidamos del botón de volver, como en el resto de páginas finales (no distribuidoras).



4.9.6. Iteración 6: contenido - contenido estático

En una última instancia queremos que la información de donde lee la aplicación venga directamente de la página web original.

De momento, introducimos información estática en esta iteración. Temporalmente, copiamos la información de la página web en el propio diseño.

Para introducir esta información podemos utilizar la facilidad que nos da Droiddraw para crear *strings* o *arrays*.

> ***Strings* y *arrays***
>
> Los *strings* se corresponden con cadenas de texto; los *arrays* son colecciones de elementos. En el primer tipo de dato, podemos guardar información, como el nombre de una sección o la descripción de un elemento. En el segundo tipo de dato podemos, por ejemplo, guardar las opciones disponibles en un apartado o los idiomas.

Podemos vincular estos tipos de datos a los diferentes componentes para que muestren el contenido de forma automática.

De este modo, la información estará disponible sin necesidad de una conexión a Internet y podremos centrarnos en que esta información se representa bien antes de cambiar su origen.

Lectura complementaria

Para una guía paso a paso de los pasos que debemos seguir, existe el siguiente tutorial:

http://www.droiddraw.org/tutorial3.html

4.9.7. Iteración 7: crear la aplicación en Android

Con la interfaz creada en la iteración anterior nos podemos disponer a generar la aplicación en el dispositivo. El programa nos generará el *layout*, que podemos importar al proyecto de Android.

Webs recomendadas

Si tenemos dificultad en exportar el diseño ejecutado en Droiddraw a una aplicación podemos consultar las siguientes referencias:

Cómo exportar desde Droiddraw:

http://www.droiddraw.org/tutorial1.html

Cómo crear una aplicación mínima con un *layout*:

http://developer.android.com/resources/tutorials/hello-world.html

Otro ejemplo de aplicación mínima con *layout*:

http://apcmag.com/building-a-simple-android-app.htm

Referencias para el concepto de *activities*:

http://developer.android.com/guide/topics/fundamentals/activities.html

4.9.8. Iteración 8: contenido - creación de webservices

Hasta este momento, hemos estado generando el contenido de manera estática. La idea inicial ha sido que el contenido de la aplicación sea el mismo que el de la página web original. Para ello, habrá que poder acceder a esta información en remoto, desde la aplicación móvil.

La solución habitual en estos escenarios es el uso de ***webservices***. Los *webservices* permiten la interacción entre distintos sistemas y facilitan el intercambio de información de forma controlada entre servicios muy dispares. Existen muchos tipos de *webservices*.

Listamos a continuación algunos:

- REST
- SOAP
- WSDL[6]
- JSON-RPC
- WPS

(6)Acrónimo de *web services description language*.

Los *webservices* se pueden desarrollar en cualquier lenguaje de tipo servidor. Para que la aplicación Android u otros servicios puedan acceder a la información de la página web, será necesario crear una capa en la aplicación web que se encargue de devolver solamente la información solicitada.

Los *webservices* se pueden desarrollar en cualquier lenguaje de tipo servidor. Para que la aplicación Android u otros servicios puedan acceder a la información de la página web, será necesario crear una capa en la aplicación web que se encargue de devolver solamente la información solicitada

Una solución habitual consiste en ofrecer una API[7]. a través de la cual pueda llamarse a los diferentes recursos. Será necesario crear este acceso para que la aplicación conecte con la información de la página web.

(7)Acrónimo de application programming interface

API

Capa de abstracción de un software utilizada como biblioteca por otro software.

Más información en:

http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

Algunos ejemplos de API los encontramos en servicios de empresas de negocio conocidas en Internet. Dos ejemplos son las API de Yahoo y del servicio de Flickr.

De este modo, cuando la información de la página web se actualice, el acceso desde la aplicación móvil también tendrá acceso a la misma información.

Si esto resultase demasiado engorroso siempre existe la posibilidad de distribuir el contenido en la propia aplicación y que se actualice únicamente de forma estática. Así, habría que actualizar la aplicación para distribuir el nuevo contenido a los usuarios.

4.9.9. Iteración 9: contenido - conexión a *webservices*

Una vez desarrollado el acceso al *webservice* en la aplicación original podemos conectarnos a este *webservice* desde nuestra aplicación.

Los recursos de Android utilizan caché cuando es posible, dependiendo de los valores en el archivo **AndroidManifest.xml**.

Existen varias aproximaciones para acceder a un *webservice* desde Android, debido a los muchos tipos de *webservices* disponibles. Elegiremos una en función del tipo implementado para acceder al contenido del web original.

Webs recomendadas

Algunos recursos que explican con suficiente detalle y de forma sencilla este proceso son los siguientes:

http://stackoverflow.com/questions/1048310/how-to-call-a-net-web-service-from-android

http://programa-con-google.blogspot.com.es/2010/12/android-como-consumir-un-servicio-web.html

http://sarangasl.blogspot.com.es/2011/10/android-web-service-access-tutorial.html

4.9.10. Iteración 10: diseño - diseño final

En esta iteración, perfilamos cualquier problema que nos hayamos encontrado con el diseño en anteriores iteraciones.

En muchas ocasiones, los *layouts* no se pueden realizar correctamente al primer intento. Debido a que se pueden anidar de formas complejas, es bueno replantear su organización y subdivisión después de un uso continuo de la aplicación para implementar posibles mejoras. El testeo en varios dispositivos también puede sacar a la luz ciertos inconvenientes de algunos tipos de *layouts* frente a otros.

4.9.11. Iteración 11: publicación en el Android Market

Publicamos la aplicación en el Android Market. Esto implica un pago de una única cantidad inicial[8].

[8] En la actualidad, esta cantidad es de 25 dólares.

Para abonar esta cantidad, es necesario crear una cuenta en **Google Checkout**, el sistema de pago de Google. Basta realizar este pago solo una vez (no es recurrente).

Al subir la aplicación a https://play.google.com/store, se nos pedirán los iconos y capturas de la aplicación, descripciones para los diferentes idiomas, el precio y rangos de recomendación por edades, entre otros. En resumen, todos los datos que aparecerán más adelante en la tienda de Google. También tendremos que aceptar las condiciones de Google y facilitar un web y datos de contacto.

Una vez publicada, la aplicación aparecerá en las categorías correspondientes del Market y los usuarios podrán descargarla, hacer valoraciones y enviarnos feedback sobre su uso.

4.9.12. Iteración 12: lanzamiento y puesta en sociedad

Al margen de la publicación en la tienda de Google, es interesante plantearse llevar a cabo algún tipo de promoción. Podemos utilizar las redes sociales o campañas de diferentes tipos para incentivar el uso de la aplicación. Un *banner* en la página web original puede ser suficiente para informar a los usuarios habituales de la disponibilidad de esta nueva manera de acceder a la misma información.

Hay que tener en cuenta que en el Android Market existe gran cantidad de aplicaciones y, a menos que el usuario busque activamente nuestra aplicación, es difícil que dé con ella de modo accidental.

4.9.13. Iteración *n*: ¿...?

Siguiendo este modelo de desarrollo podemos ir incorporando cualquier cambio necesario a la aplicación. Podemos revisar el *layout* en cualquier momento o añadir funcionalidad extra accediendo al contenido necesario a través del *webservice*. Así, el sitio web original y la aplicación Android pueden compartir parte del desarrollo y evolucionar de forma conjunta. Para funcionalidades

más avanzadas, siempre tenemos como referencia la documentación oficial de Android, con las recomendaciones y buenas prácticas para nuestros proyectos Android.

Recomendamos descargar ejemplos con las funcionalidades deseadas para investigar en profundidad cómo están desarrollados.

5. Desarrollo de un web con contenidos administrables en Drupal

5.1. Introducción

El proyecto consistirá en desarrollar un portal web ficticio con las siguientes características:

- contenido administrable,
- backoffice de uso sencillo para administrar los contenidos,
- multiidioma,
- menús y navegación de varios niveles,
- elementos y noticias de diferentes tipos,
- listado de noticias y detalle de esas noticias, y
- funcionalidad de búsqueda.

5.2. ¿Por qué Drupal?

Drupal es un gestor de contenidos o CMS[9]. muy utilizado. Puede ayudarnos a acelerar el desarrollo de un sitio web debido a que gran parte de la funcionalidad ya está escrita y testeada por la comunidad.

[(9)]Acrónimo de content management system

Código aprovechado

"El mejor código es el que no es necesario escribir." Utilizando software de otros podemos recortar considerablemente el tiempo de desarrollo.

A continuación, presentamos una simple tabla comparativa con algunos de los CMS más habituales, según nuestra experiencia:

Tabla comparativa de las principales CMS

Nombre	Plataforma	Bases de datos	Licencia	Última fecha
Drupal	PHP	MySQL, Oracle, PostgreSQL, SQ-Lite, Microsoft SQL Server	GPL	01/02/12
Joomla!	PHP	MySQL	GPL	15/03/2012
Wordpress	PHP	MySQL	GPL	03/01/12
TYPO3	PHP	MySQL, Oracle, PostgreSQL	GPL	24/01/2012
Django-cms	Python/Django	PostgreSQL, MySQL, SQLite 3, Oracle	BSD	27/01/2012
MoinMoin	Python	Archivo plano	BSD	21/02/2012
Prestashop	PHP	MySQL	GPL	08/02/12

Webs recomendadas

Aquí se pueden ver otras tablas comparativas:

http://socialcompare.com/en/comparison/popular-content-management-system-cms-comparison-table

http://en.wikipedia.org/wiki/List_of_content_management_systems

5.2.1. Ventajas

Las ventajas que presenta Drupal son las siguientes:

1) Toda la parte de gestión de contenidos está desarrollada desde un inicio.

2) El código está revisado por un mayor número de gente debido a ser código libre.

3) Existe una multitud de módulos para todo tipo de tareas y suelen ser fáciles de utilizar.

5.2.2. Inconvenientes

Las inconvenientes que presenta Drupal son los siguientes:

1) Los estándares de calidad de Drupal no son tan altos como los de otros proyectos *open source*. A veces, la implementación de algunos módulos[10] deja bastante que desear.

[(10)]No citaremos ninguno en concreto.

2) Algunas funcionalidades muy específicas pueden ser más difíciles de implementar que en un desarrollo a medida.

5.3. Consideraciones generales de desarrollo

Drupal, como muchos otros sistemas, consta de una parte de archivos (código y recursos gráficos) y una base de datos de respaldo. Durante el desarrollo, ambas partes se van transformando. Existen dos maneras de gestionar estos cambios especialmente útiles para la gestión de un proyecto:

1) control de versiones y

2) *deployment*

5.3.1. Control de versiones

El uso de sistemas de control de versiones durante el desarrollo facilita:

- volver a versiones anteriores de archivos;
- gestionar los cambios de multitud de desarrolladores de forma ágil; y
- sirve como documentación del propio proyecto.

Control de versiones

Algunos ejemplos de sistemas de control de versiones serían Git y Subversion

5.3.2. *Deployment*

El *deployment* consiste en automatizar la subida a producción. Esto nos ayuda a:

- evitar errores manuales cuando son más críticos (subida a producción de un nuevo desarrollo);
- poder desarrollar con iteraciones rápidas;
- solucionar imprevistos de forma rápida; y
- como documentación posterior de los parámetros de configuración.

Normalmente existen entornos de desarrollo, de test y de producción:

- **Desarrollo**: donde se introducen los cambios.
- **Test**: se comprueba que todo funciona de forma correcta. Es recomendable probar la aplicación en test con datos ya presentes en producción para testear el *site* con datos reales.
- **Producción**: es donde está la aplicación real. Ese es el único web con el que interactuarán los usuarios finales de la aplicación.

Estos tres entornos deben ser lo más parecidos el uno del otro (en especial test y producción).

El sistema de *deployment* suele ser muy personalizado, por lo que podéis utilizar *scripts* que automaticen los comandos básicos (como copiar a una carpeta, subir a una dirección FTP o copiar la base de datos).

Para escenarios más complejos (varios servidores o bases de datos o entornos replicados, por ejemplo) existen sistemas de *deploy* más complejos.

Webs complementarias

Deploy más complejos como Chef y Puppet.

5.3.3. Seguridad

La seguridad es muy importante en cualquier tipo de desarrollo web.

No es extremadamente raro que una persona o proceso automatizado consiga introducirse en un servidor aprovechándose de alguna vulnerabilidad.

1) Básicos

El software tiene *bugs* (errores en un programa informático).

Lectura complementaria

Para conocer el origen del término *bugs* ved:
http://en.wikipedia.org/wiki/Software_bug#Etymology

Se trata de minimizar estos *bugs* siguiendo unas prácticas adecuadas. Programas que pueden ser correctos hoy pueden ser inseguros simplemente con el paso del tiempo. Se descubren vulnerabilidades que no se solucionan y pueden ser aprovechadas por un atacante.

2) Seguridad y SEO[11].

(11)Acrónimo de search engine optimization

Hay que recordar que una de las consecuencias, aunque quizás de las menos importantes, de una intrusión de seguridad es que los principales buscadores harán perder muchos puestos en los resultados a la página. Pueden llegar incluso a eliminarla de los resultados. Este proceso puede alertar a los usuarios de la página, que pueden perder la confianza en el sitio web.

Mantener la confianza

Tanto la confianza de los usuarios como la de los motores de búsqueda es muy fácil de perder y extremadamente difícil de recuperar, por lo que conviene invertir en seguridad antes de que se produzca una intrusión.

3) Medidas de seguridad

Conviene seguir las buenas prácticas del desarrollo en el lenguaje y *framework* que utilicemos.

Webs recomendadas

Algunas buenas referencias sobre medidas de seguridad al desarrollar una aplicación son las siguientes:

http://www.owasp.org/index.php/Main_Page

http://stackoverflow.com/questions/2621706/good-guide-for-web-app-security

http://stackoverflow.com/questions/6166833/hacking-how-do-i-find-security-holes-in-my-own-web-application-did-i-do-a-good

http://stackoverflow.com/questions/3891349/web-applications-security

http://stackoverflow.com/questions/47323/top-tips-for-secure-web-applications

Algunos consejos pueden ayudar a recuperarnos más rápidamente en caso de una intrusión:

- Mantener *backups* regulares y probar el proceso de recuperación de esos *backups*.

- Automatizar el *deploy* y la recuperación de *backups*. Prácticamente todos los servicios de *hosting* tienen un servicio de *backups* que, por poco dinero más, nos proporcionará la restauración de *backups* de una fecha determinada. Conviene tenerlo en cuenta antes de elegir un *hosting*.

- Actualizar regularmente la versión de los CMS, *framework*s y librerías que utilizamos.

Muchos atacantes quieren mejorar la eficiencia de los ataques. Por eso, eligen productos de software que utilice mucha gente y buscan vulnerabilidades en ellos. De este modo, se utiliza una misma vulnerabilidad para introducirse en muchos servidores.

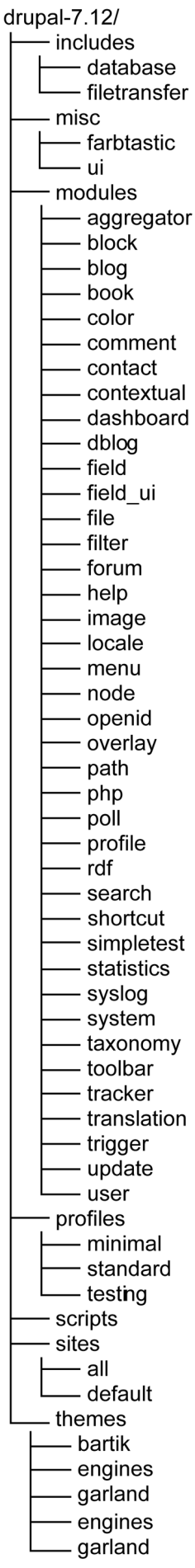
Conviene usar la última versión estable, pero ir actualizando esa versión. La versión más reciente suele estar aún en desarrollo y presenta problemas de seguridad que aún no han sido encontrados.

La versión demasiado antigua tiene problemas de seguridad que nadie soluciona, ya que es obsoleta.

El compromiso entre las dos suele ser la solución óptima para la mayoría de casos de uso.

5.4. Estructura de un proyecto Drupal

Drupal utiliza una estructura de carpetas convencional que organiza las diferentes partes de la aplicación. Podemos ver los dos primeros niveles de directorios en el gráfico siguiente:

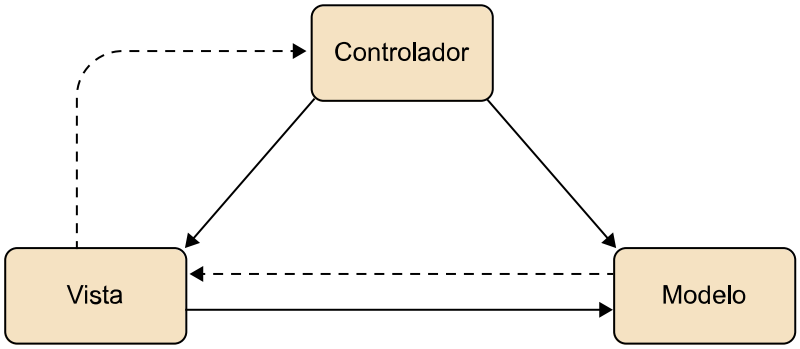

drupal-7.12/
includes
database
filetransfer
misc
farbtastic
ui
modules
aggregator
block
blog
book
color
comment
contact
contextual
dashboard
dblog
field
field_ui
file
filter
forum
help
image
locale
menu
node
openid
overlay
path
php
poll
profile
rdf
search
shortcut
simpletest
statistics
syslog
system
taxonomy
toolbar
tracker
translation
trigger
update
user
profiles
minimal
standard
testing
scripts
sites
all
default
themes
bartik
engines
garland
engines
garland

Podemos ver cómo guarda cada módulo base en la carpeta **modules**.

En **Sites/Sll** están las carpetas **Modules/** y **Themes/**, donde se sitúan los módulos y temas personalizados.

5.5. Modelo vista controlador y Drupal

El **modelo vista controlador** es un patrón de arquitectura de software útil para mantener diferentes secciones de la aplicación separadas y desarrollarse de forma más clara. Fomenta la **separación de responsabilidades**, un principio del desarrollo de software.



Fuente: http://craftyman.net/mvc-en-php/

Drupal no usa exactamente este modelo, pero puede ser útil ver qué abstracciones usa Drupal y qué equivalentes tendrían en el modelo vista controlador para entender mejor su función.

1) Modelo. El modelo especifica las reglas de negocio y el acceso directo a los datos. Es el cerebro de la aplicación.

2) Vista. Drupal tiene varios sistemas equivalentes a la funcionalidad de vistas. Por un lado, tiene los temas (*themes*), que serían conjuntos de vistas con una coherencia visual.

Cada una de estas vistas es estilable por código de manera independiente. Estos archivos siguen un sistema de convención para saber de qué elemento van a ser la vista en función del nombre que tiene el archivo. Cada uno de estos archivos sería lo más parecido a una vista clásica. Son *templates* que llaman a las funcionalidades necesarias.

Por último, tenemos los **nodos**, elementos individuales que podemos estilar como queramos.

3) Controlador. No existe un controlador como tal, aunque podríamos entender que la definición a través de los paneles de administración actúa realmente de controlador: vincula cierta página a ciertas vistas.

5.6. Visión general

5.6.1. Instalación de Drupal en *server*

Para instalar, simplemente vamos al sitio web de Drupal y descargamos la última versión estable disponible.

Descomprimimos el resultado y lo situamos en un directorio que pueda servir un servidor web (por ejemplo, Apache) y visitamos la página donde veamos ese directorio. Normalmente, será **localhost:número_de_puerto** si lo estamos instalando en la propia máquina utilizada para el desarrollo. A partir de ahí, podemos seguir los pasos para configurar la base de datos, el usuario y contraseña por defecto y otras características básicas del proyecto.

Tendremos que crear una base de datos con las mismas credenciales especificadas en estos pasos para permitir el correcto funcionamiento de la aplicación.

Bases de datos y servidor

Podemos utilizar una gran variedad de programas para tener una base de datos y servidor en local y para gestionarla. Para hacerla funcionar, algunos de los más sencillos son WAMP para Windows y MAMP para Mac OS. Para sistemas UNIX/Linux, lo más habitual sería instalar el servidor (por ejemplo, Apache) y la base de datos (por ejemplo, MySQL) desde el sistema de gestión de paquetes de la distribución (YUM, apt-get, pacman, ports).

Desde la versión 5.4 PHP, incorpora también su propio servidor de test.

Para conectarse a la base de datos recomendamos MySQL Workbench, de Oracle. Está disponible para varios sistemas.

Después de este proceso, ya tenemos una aplicación funcional y podemos empezar el desarrollo. Podemos ver la aplicación en **localhost:número_de_puerto** con el texto "Welcome to localhost".

5.6.2. El panel de administración

Todas las tareas de administración de Drupal se desarrollan a través del panel de administración de la propia web, accesible en la sección privada del *site*. Excepto en casos en los que será necesaria la edición del código o la manipulación de archivos, la mayor parte de cambios se introducen a través de esta interfaz web.

5.6.3. Funcionalidades

Nos interesa desarrollar la aplicación de un modo ágil. Por lo tanto, intentaremos desarrollarla de forma incremental y en pequeñas iteraciones. Para ello, nos centraremos en las funcionalidades que queremos obtener.

En Drupal, las diversas funcionalidades básicas de un *site* son accesibles a través de módulos. Iremos instalando módulos a medida que vayamos necesitando las funcionalidades correspondientes.

5.7. Especificaciones

¿Qué quiere el cliente?

El cliente quiere crear un sitio web de contenidos administrables. No tiene un departamento propio de desarrollo web y quiere que sus empleados sean capaces de administrar los contenidos necesarios de forma fácil e intuitiva. Tiene una idea clara sobre la navegación y los tipos de contenido que espera encontrar. También pide consejo en otros contenidos que no sabe exactamente cómo distribuir en el portal.

¿Qué se le propone?

El cliente no requiere de unas funcionalidades muy avanzadas de ámbito tecnológico. Básicamente el sitio web, aunque tiene un diseño y unas funcionalidades no triviales, no va al límite de lo que se puede hacer con la tecnología y utiliza muchas funcionalidades presentes en gran cantidad de aplicaciones web.

Debido a esto y al coste, descartamos crear una aplicación a medida. En este caso, deberíamos desarrollar también la parte de gestión de la aplicación. El cliente no requiere de una administración a medida y está dispuesto a encontrar algunas limitaciones en este aspecto a cambio de agilizar el proceso de desarrollo.

5.8. Iteraciones

5.8.1. Iteración 1: la primera página

Si entramos en la dirección del *server*[12], podemos ver una página por defecto con su contenido. Vamos a crear a continuación una página propia con un nombre que escogeremos nosotros.

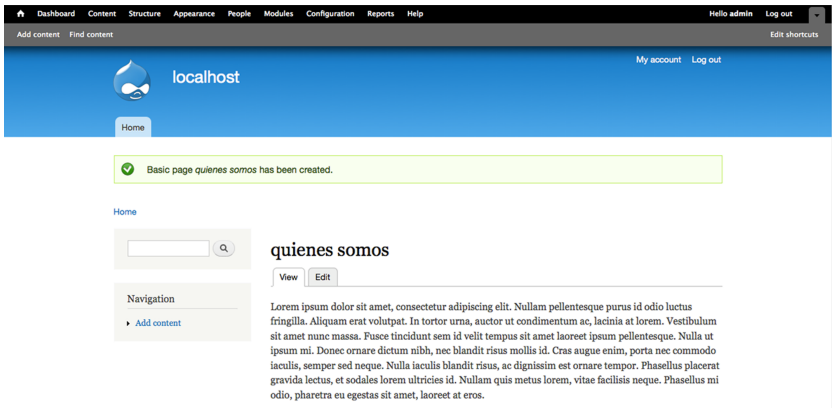
[12]De ahora en adelante *localhost*.

En **Content > Add content** creamos una nueva página de tipo **Basic page**. Le ponemos un título y un texto ficticio.

Texto ficticio

Para el texto suele utilizarse el conocido *Lorem Ipsum*. Existen varios generadores, uno de los cuales es http://www.lipsum.com.

Nos indicará que la página se ha creado correctamente.



Podemos ver, además de la página con el contenido introducido, las diferentes secciones de la página. Estas secciones serán las mismas para todas las páginas. Vemos el menú de administración en la parte superior y, debajo de él, la página que verán los visitantes sin registrar de la página. Además de un menú de navegación, que de momento solo muestra un elemento, vemos unas opciones laterales y el contenido en sí de la página.

5.8.2. Iteración 2: tipos de contenido - noticias

Hasta ahora, tenemos un sitio básico en funcionamiento. También hemos creado una página de contenido sencillo. Para la creación de páginas más complejas necesitaremos crear diferentes tipos de contenido.

Definimos a continuación los tipos de contenido para la página que mostrará las noticias.

La *home* consistirá en una serie de noticias con imagen pequeña, imagen grande, título y un contenido textual de la noticia.

Para ello, necesitaremos instalar varios módulos:

- El módulo Views nos permitirá crear listas de elementos con un criterio y, en general, organizar el contenido. A su vez, este módulo requiere el módulo Ctools para su funcionamiento.

- También necesitaremos el módulo Entityreference para enlazar diferentes tipos de contenidos en estructuras más complejas. Este módulo, a su vez, depende del módulo Entity.

En general, veremos que las páginas de los módulos tienen una descripción de los módulos de los que dependen, las versiones de Drupal con las que funcionan correctamente, referencias y otras recomendaciones.

Para instalar módulos de Drupal descargamos los correspondientes a nuestra versión de Drupal y los situamos en la carpeta **Modules** de la estructura de archivos del proyecto.

Con los archivos en el lugar indicado, activamos los módulos desde el panel de administración: en la opción del menú **Módulos** activamos los módulos y sus dependencias.

Existen varias aproximaciones para trabajar con distintos tipos de contenido. Una opción interesante y muy flexible es crear tipos de contenido básico (como imagen, noticia o *banner*) y enlazarlos entre sí en tipos más complejos que representen una página o sección compleja. Por ejemplo, un tipo "página noticia" puede estar formado por *n* noticias y dos *banners*. Esta es la aproximación que usamos aquí.

Creamos una nueva definición de tipo de contenido desde **Structure > Content types > add content type**. Lo llamamos noticia. Existen multitud de opciones extra disponibles en este punto. A continuación, definimos los campos que va a tener este tipo de contenido:

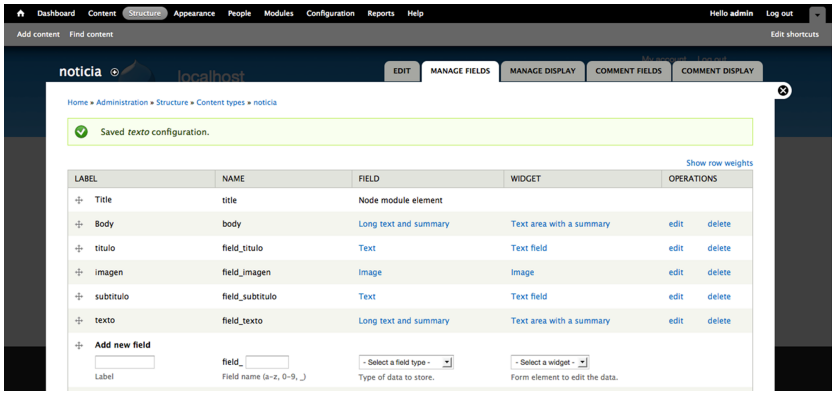
1) Título: el título de la noticia. Es de tipo texto.

2) Imagen: la foto que puede incluir la noticia. Podemos incluir imágenes por defecto, tamaños mínimos y máximos y multitud de opciones. Es de tipo imagen.

3) Subtítulo: el subtítulo (si lo tiene) de la noticia. Es de tipo texto.

4) Texto: el propio contenido de la noticia. Elegimos la opción **Long text with summary** para que nos genere el resumen de la noticia.

Podemos ver una imagen de la pantalla de definición de campos del tipo de contenido noticia:



En este punto, tenemos definido el tipo de contenido noticia. Necesitamos un tipo de contenido página noticia, que contendrá, mediante referencias del módulo de referencia de entidades, contenidos de tipo noticias. También podría incluir otros tipos de contenidos como *banners*, encuestas o cualquier otro tipo que definamos.

Al final de esta iteración, acabamos con la definición completa de tipos referentes a la página que muestra las noticias.

5.8.3. Iteración 3: tipos de contenido - *banners*

Un tipo interesante para un portal web es el de diferentes tamaños de *banners*. Se tendrán que poder situar en las otras páginas.

De igual modo que hicimos antes, crearemos un tipo de contenido específico para cada tipo de *banner* que tengamos. En este caso, implementamos un tipo al que llamamos *banner* sencillo.

La funcionalidad de incluir un campo con un enlace no existe por defecto en Drupal. Para utilizarla, necesitaremos instalar el módulo Link.

La definición de un *banner* sencillo sería la siguiente:

1) Imagen: imagen que se mostrará como contenido del *banner*. De tipo imagen.

2) Texto: texto opcional para mostrar por encima de la imagen. De tipo texto.

3) Enlace: enlace adonde apunta el *banner*. Es de tipo *link*.

5.8.4. Iteración 4: tipos de contenido - formulario de contacto

Para crear los formularios de contacto, necesitamos instalar el módulo Webform.

Una vez instalado y activado, lo utilizamos desde **Content > Add content > webform**. Le ponemos como nombre contacto. Pasamos a definir sus campos, del mismo modo que definimos los del tipo de contenido.

Además de la definición de campos, podemos definir a qué dirección de email se enviarán los resultados. Cada vez que alguien rellene este formulario se enviará a la dirección de correo electrónico configurada. También quedarán los registros accesibles desde el administrador, desde la pestaña **results**. De este modo, podemos comprobar fácilmente el *input* de los usuarios de manera muy conveniente.

5.8.5. Iteración 5: páginas - noticias

Hasta ahora, hemos creado todas las definiciones de tipos que necesitábamos para las páginas.

A continuación, vamos a crear páginas con los tipos de contenido que hemos comentado. Utilizaremos las vistas en lugar de la creación de páginas básicas para tener mayor flexibilidad. Las vistas permiten poner restricciones en los tipos de contenido, paginar los resultados y multitud de funcionalidades de gran utilidad.

Vamos a **Estructure > Views** en el menú de administración y creamos una nueva vista. Podemos elegir ahí el nombre de esa vista, que será el nombre de la página que contenga la vista. Creamos una vista de nombre página-noticias y podremos ver esa vista en la página **localhost/pagina-noticias**.

Nombre de la página

Si el nombre de la página contiene caracteres prohibidos, Drupal sustituirá los caracteres por otros permitidos o los eliminará de la url. Este suele ser motivo de confusión al buscar los *templates* correspondientes a esta vista o al buscar la página que se ha creado.

En la vista de página-noticias, accesible desde **Estructure > Views**, filtraremos para que solamente muestre el contenido de tipo página-noticia. De este modo, la página tendrá acceso a toda la información definida con este tipo, incluidos los datos vinculados a este tipo a través de referencias.

Con esta vista definida, ya podemos ver el resultado visitando la página. Podemos observar que la página está vacía.

¿A qué se debe que la página esté vacía?

En realidad, solo hemos creado la definición de relaciones entre tipos y lo que debe mostrar la vista, pero no hemos creado contenido como tal.

Con el fin de que haya contenido real en la página, vamos a añadirlo desde el panel de administración. Aquí veremos uno de los principales beneficios de Drupal.

Desde **Content > Add content** seleccionamos el tipo de dato correspondiente y vamos añadiendo los valores para los diferentes campos.

Panel de administración

Automáticamente, Drupal tiene este panel de administración, con restricciones para los diferentes tipos y facilidades como la subida de imágenes y su reescalado. Todo esto solamente a partir de la definición de tipos de contenido.

5.8.6. Iteración 6: páginas - *banners*

En esta iteración, vamos a crear el contenido de los *banners*. Para ello, simplemente agregamos el contenido desde **Content > Add content**.

Para añadir los *banners* a las diferentes páginas, podemos incorporarlos en la definición de tipos de páginas, como en la anterior página-noticias. Cualquier página que muestre *banners* tendrá en la definición de campos de su tipo una referencia al tipo *banner*.

La ventaja de esta aproximación al problema es que tenemos un tipo único de datos para cada página y, al mismo tiempo, podemos reaprovechar tipos de contenido más específicos como noticia, *banner* o imagen.

5.8.7. Iteración 7: páginas - formulario de contacto

Crearemos una página para poder recibir *inputs* de los usuarios. Así, podrán enviarnos sus opiniones o sugerencias. En el futuro, podría ampliarse la funcionalidad para solicitar información o adquirir servicios que ofrezcamos.

El tipo *webform* es un tipo más de contenido, como cualquier otro. Podemos incorporarlo a la definición de las páginas que lo utilicen, utilizando una referencia de entidad. Luego, desde el *template* de la vista correspondiente, lo estilamos como mejor nos convenga.

5.8.8. Iteración 8: diseño

En este punto, tenemos ya el contenido y las páginas funcionales. Hemos optado por desarrollar una web funcional sobre la que después aplicaríamos el diseño. Ha llegado el momento de definir el diseño de estas páginas, teniendo en cuenta de qué modo las pinta Drupal, para aprovechar al máximo las funcionalidades que nos ofrece sin rehacerlas por completo.

En algunas ocasiones, será necesario modificar el modo en como Drupal pinta algunos de los elementos. También puede ser necesario tener que añadir CSS o Javascript para modificar el aspecto o el comportamiento del lado cliente de esos comportamientos.

Teniendo en cuenta estos recursos, ejecutamos un diseño aproximado de las páginas. Podremos ir iterando sobre este diseño a medida que lo vayamos implementando.

Para saber qué opciones tenemos para implementarlo, veamos la siguiente iteración.

5.8.9. Iteración 9: maquetación - estilos comunes

En este punto, contamos ya prácticamente con la funcionalidad de toda la página web. Tenemos las diferentes páginas y sus elementos mostrándose.

En esta iteración, empezaremos a personalizar el modo como Drupal muestra estos elementos, principalmente el aspecto que afecta a todas las páginas.

Elegiremos un tema similar al que nos interese o uno mínimo para tener una base sobre la que comenzar. El tema por defecto (*bartik*) es bastante sencillo y es una buena opción.

Tema

Para cambiar el tema activo, simplemente vamos a la opción de menú **Appearance**. Allí podemos seleccionar entre los diferentes temas.

Accediendo a los archivos de *template*, podemos ver los *templates* genéricos. Los podemos encontrar en **/themes/bartik/templates/**.

También existe un archivo *template*.php, que define funciones genéricas para el proceso de los *templates*. Este archivo permite modificar muy a fondo ciertas funcionalidades, pero también es más complejo.

Webs recomendadas

Para profundizar más en las opciones que permite *template*.php:

http://api.drupal.org/api/drupal/includes!theme.inc/function/theme/7

http://api.drupal.org/api/drupal/themes!garland!template.php/7

Para este paso, elegimos el *template* común a todas las páginas: **page.tpl**. Incluimos las rutas de CSS y Javascript necesarias para el desarrollo en el mismo archivo.

Nota

Incluiremos los archivos CSS y Javascript de la forma estándar:
http://www.w3schools.com/tags/tag_link.asp

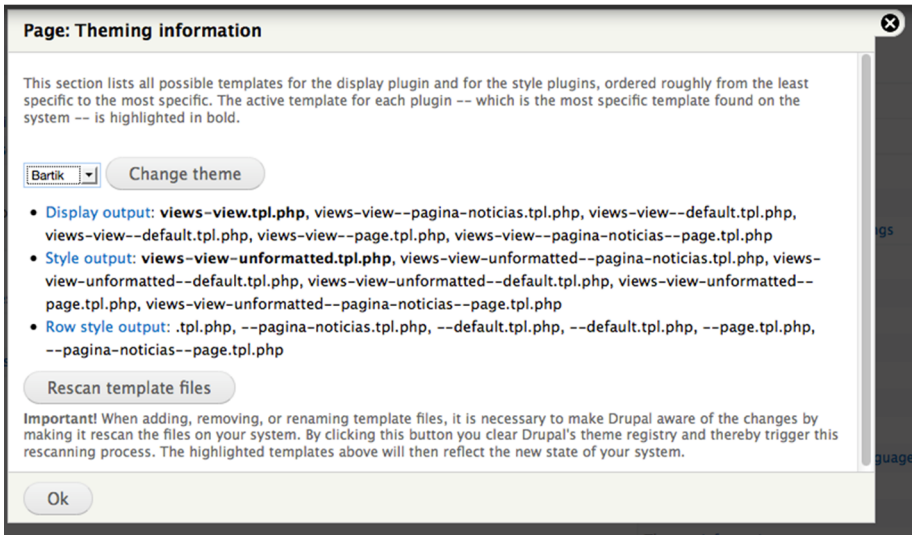
Incluyendo estos archivos, podemos maquetar con CSS y desarrollar la programación cliente en Javascript del mismo modo que en cualquier otro desarrollo web.

5.8.10. Iteración 10: maquetación - *templates* específicos

Drupal crea diferentes *templates* para las vistas, páginas, bloques. Prácticamente cualquier elemento presente en el gestor tiene un *template* equivalente. Modificando estos elementos podemos cambiar totalmente cómo se muestran todos los elementos de un sitio web desarrollado en Drupal.

Uno de los recursos más flexibles para gestionar el contenido, y sobre el que basamos gran parte de la organización de los elementos, son las vistas. A veces, puede resultar complicado encontrar el nombre del *template* adecuado para una vista. Una buena manera de encontrar los *templates* disponibles para una vista es ir a la página con toda su información.

Desde **Structure > Views** editamos la vista de la que queremos obtener la información. En la edición, podemos ver diferentes apartados. En el apartado de **Opciones avanzadas** seleccionamos la opción **Themes**. Allí se muestra la ubicación de todos los archivos de *templates* disponibles.



Modificando o creando esos archivos podemos personalizar el html que Drupal pintará para el elemento (ya sea una vista o cualquier otro tipo de elemento).

5.8.11. Iteración 11: menú y navegación

Ahora que ya tenemos páginas con contenido, podemos establecer un mecanismo para navegar entre ellas. Situaremos las diferentes rutas de las páginas en un menú.

En **Structure > Menu** podemos editar los menús y, desde la opción **List links**, es posible ver los enlaces de los que consta un menú y añadir o quitar otros.

Con estos cambios, podremos modificar los tipos de menú que Drupal nos ofrece por defecto:

- **Main Menu**: normalmente es un menú horizontal en la parte superior de la pantalla. Muestra las secciones principales del sitio.

- **Management**: este tipo de menú contiene opciones para las funcionalidades administrativas del sitio web.

- **Navigation**: esta tipología de menús contiene enlaces dirigidos a los visitantes del sitio. Algunos módulos añaden nuevos menús a este tipo cuando se utilizan.

- **User Menu**: menú vinculado al usuario. Contiene los ítems de *Log out* y los propios de la cuenta del usuario activo.

Cualquiera de estos menús puede ser estilado con CSS desde hojas de estilo incluidas en Drupal desde *templates* genéricos. Lo mismo sucede con el Javascript que utilicen.

En algunos casos, sobre todo si queremos un control total sobre cómo se pinta el menú en HTML, será necesario buscar los *templates* del menú y modificarlos. Puede ocurrir que, si el comportamiento que queremos modificar es muy específico, necesitemos modificar comportamiento del core de Drupal. Para ello, podemos utilizar las funciones de *template*.php.

Webs recomendadas

Además de otras páginas de referencia con las opciones disponibles, algunos buenos recursos de casos prácticos para este tipo de modificaciones son los siguientes:

http://drupal.org/node/11811

http://answers.oreilly.com/topic/1512-using-drupals-templatephp-for-overrides/

http://www.nerdliness.com/article/2008/01/01/power-template-php

http://stackoverflow.com/questions/5347154/drupal-template-php

http://drupal.org/node/223430

Existen multitud de módulos que nos ayudan con la gestión y la personalización de menús y su aspecto. Merece la pena investigar cuales nos pueden ser de mayor ayuda.

Módulos para la gestión y la personalización de menús

Algunos de los más útiles son los siguientes:

- **Jump**. Crean una opción de navegación rápida para cualquier vocabulario definido.
- **Menu block**. Crea listas de bloques personalizables y muy adaptables en los menús.
- **Menu minipanels**. Permite añadir contenido rico (*rich content*) en las opciones de menú.
- **Nice Menus**. Crea menús desplegables atractivos que funcionan correctamente en todos los navegadores.
- **Nice Primary Menus**. Divide los menús en una lista de menús primarios y secundarios y facilita la navegación entre ellos.
- **Site Menu**. Crea un árbol de navegación del *site* partiendo de las categorías definidas en él.
- **Submenu Tree**. Este módulo permite estructurar el contenido de forma jerárquica y navegar en función de esta jerarquía.
- **Superfish**. Permite utilizar la librería Superfish automáticamente en un menú de Drupal. Esta librería genera menús atractivos de forma muy sencilla.

5.8.12. Iteración 12: buscador

En esta iteración, investigamos la funcionalidad de buscador de la aplicación.

Drupal incorpora esta funcionalidad por defecto. Gracias a esto no va a ser necesario desarrollarla. Podemos personalizar las opciones de búsqueda desde **Configuration > Search settings**. Desde este panel de opciones podemos, entre otros:

- forzar el reindexado de todo el sitio web;

- seleccionar en qué módulos se efectúa la búsqueda y en cuáles no; y

- definir el orden de importancia de cada tipo de resultados, de modo que se destaquen unos por encima de otros.

5.8.13. Iteración 13: internacionalización

En esta iteración, vamos a incorporar funcionalidad multiidioma a todos los contenidos que aparezcan en el sitio. Existen varios módulos de Drupal para esta funcionalidad.

1) Módulos útiles

Instalaremos los siguientes módulos:

- **Internationalization**
- **Pathauto**
- **Token**
- **Transliteration**
- **Variable**
- **Chaos Tools**
- **Views**
- **Internationalization Views**

Por defecto, Drupal viene solamente con el idioma inglés. Añadiremos el nuestro en **localhost/admin/config/regional/language en Add language**.

En la opción **Configure** podremos también determinar cuál de los idiomas es el idioma por defecto y otros parámetros.

2) Selector de idioma

Activamos la opción de cambiar de idioma en **Structure > Block** para poder ver el selector de idiomas en la página.

3) Traduciendo contenido

Para que aparezca el contenido traducido, tendremos que activar en **Publishing options > Multilingual support** la opción **Enabled, with translation**.

Una vez activada, observaremos que aparecen pestañas para traducir al crear los contenidos.

Con estas opciones, podremos traducir fácilmente el contenido del sitio.

Webs recomendadas

Otros módulos de traducción que nos pueden ser de utilidad son los siguientes:

Administration Language

Administration Menu

Demonstration site (Sandbox / Snapshot)

Module Filter

5.8.14. Iteración *n*: ¿...?

Cualquier cambio que se produzca en el desarrollo del *site* en Drupal se puede introducir siguiendo este proceso incremental. La mayoría de nuevas funcionalidades se facilitan encontrando el módulo de Drupal adecuado. Los cambios de maquetación o de cliente siguen el desarrollo normal de CSS o Javascript. En las ocasiones en las que es necesario modificar el html generado, podemos acceder al *template* correspondiente y modificar el modo como se pintan los elementos.

6. Desarrollo de una aplicación para Facebook

6.1. Introducción

El proyecto consistirá en desarrollar una aplicación para Facebook.

La aplicación consiste en un concurso ficticio con las siguientes características:

- con diferente contenido para usuarios que han hecho "Me gusta";
- aceptación de las bases legales;
- diseño de las diferentes pantallas que el usuario va a utilizar;
- recogida de datos del usuario y almacenamiento en una base de datos; y
- opciones de compartir con amigos.

6.2. ¿Por qué Facebook?

6.2.1. Ventajas

Las ventajas que presenta Facebook son las siguientes:

1) dispone de una gran cantidad de base de usuarios;

2) facilita conectar con usuarios reales;

3) gran parte de personas pasa mucho tiempo en Facebook y nos da la posibilidad de acercar nuestra marca o producto a ese público.

6.2.2. Inconvenientes

Hay que tener en cuenta que, al desarrollar una aplicación para Facebook, dependemos de esta empresa. La API puede cambiar en cualquier momento (y de hecho cambia), de igual modo que los requerimientos de las aplicaciones. Estos cambios pueden hacer que ciertas aplicaciones dejen de funcionar o funcionen de forma incorrecta cuando antes lo hacían de forma normal.

Reflexionar antes de comenzar

Antes de comenzar el desarrollo de una aplicación en Facebook, conviene plantearse realmente si es lo más conveniente para los objetivos que el cliente pretende alcanzar. En ocasiones, una aplicación web estándar ofrece mayores posibilidades para el cliente.

6.3. Opciones para desarrollar en Facebook

Facebook nos permite desarrollar con un sistema de *markup* propio llamado **FBML** y con un modelo de desarrollo igual que una página web normal. Debido a que no hay ninguna funcionalidad de FBML que no esté ya disponible desde un desarrollo clásico y que habría una curva de aprendizaje, optaremos siempre por desarrollar prescindiendo de FBML.

Para este desarrollo, podemos usar cualquier lenguaje del lado del servidor. En este caso, utilizaremos PHP con el *framework***Codeigniter**.

Lectura complementaria

Referencia de documentación del *framework* **Codeigniter**:
http://codeigniter.com/user_guide/

6.4. Consideraciones para desarrollar en Facebook

Temas de política de privacidad. En el caso de una aplicación como la de este caso, un concurso, conviene mostrar unas bases legales que el usuario debe aceptar antes de entrar a la aplicación.

6.4.1. Certificados seguros

Desde hace un tiempo, Facebook exige que las aplicaciones tengan el protocolo seguro. Habrá que tener en cuenta este previo desarrollo para contratar un certificado seguro.

Lectura complementaria

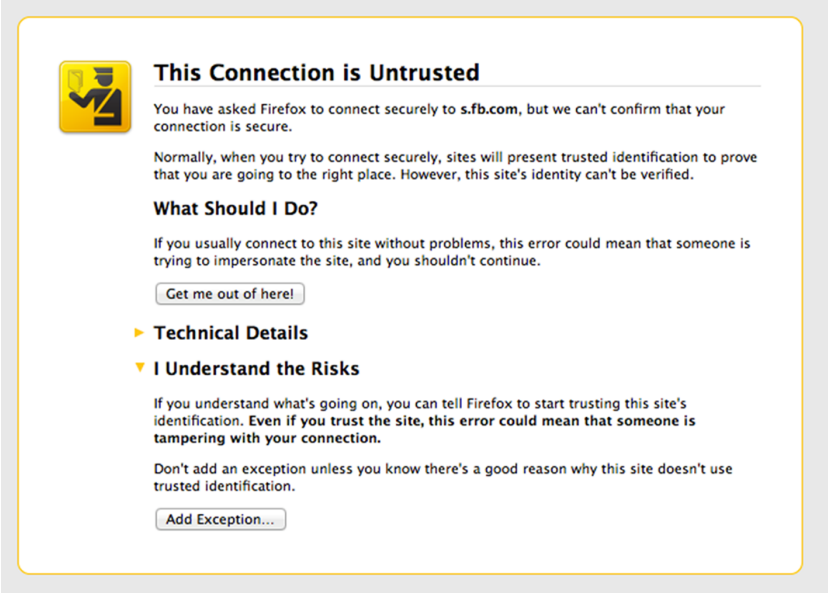
Para más información sobre el protocolo seguro:
http://es.wikipedia.org/wiki/Hypertext_Transfer Protocol_Secure

Certificados seguros

La gran mayoría de empresas de *hosting* ofrecen servicios de contratación de certificados seguros. Nos interesa un certificado seguro emitido por una entidad importante, ya que los navegadores muestran mensajes de advertencia cuando los certificados no son de entidades muy importantes, mientras que no los muestran para las principales entidades certificadoras. Estos mensajes, aunque son puramente informativos, pueden provocar que el usuario abandone la página.

Certificado de entidad

La imagen que se expone a continuación es con la que se encontrarán los usuarios si el certificado no es de una entidad certificadora (CA) importante. Aunque el mensaje es aceptable, produce más alarma de la que muchos usuarios pueden tolerar (abandonarán la página).



6.4.2. Seguridad

En las aplicaciones de Facebook, deberemos seguir las mismas medidas de seguridad que en el desarrollo de una página web. Requieren especial atención las aplicaciones que tratan información de carácter personal o de concursos.

Webs recomendadas

Algunos buenos recursos sobre la seguridad en el desarrollo de páginas web son los siguientes:

http://programmers.stackexchange.com/questions/46716/what-should-every-programmer-know-about-web-development

http://www.owasp.org/index.php/Main_Page

6.4.3. Limitaciones del contenido

Al diseñar y maquetar un aplicación para Facebook, debemos tener en cuenta que no disponemos de todo el espacio de pantalla disponible en una página web. La integración con Facebook implica una serie de elementos de interfaz de Facebook que engloban el contenido.

En estos momentos, el espacio que Facebook permite a la aplicación es de 760 px de ancho. En vertical, no existe una limitación concreta, ya que podemos hacer uso del *scroll*. Habrá que considerar qué parte del espacio vertical también está ocupada por la interfaz de Facebook, de modo que cabe ser especialmente cuidadoso para controlar qué ve el usuario en una primera visita antes de hacer *scroll*.

6.5. Visión general

Para maquetar una aplicación de Facebook, se utilizan las mismas herramientas y lenguajes que para el desarrollo de un web estándar. El aspecto se define mediante html y CSS, mientras que el comportamiento se desarrolla en Javascript (para programación del lado del cliente) y en un lenguaje servidor (en este caso PHP) para el lado del servidor. Este lenguaje servidor suele conectar típicamente con una base de datos (en este caso MySQL) para persistir los datos.

El desarrollo en PHP suele hacerse basándonos en algún *framework*. Algunos de los *framework*s más utilizados en PHP son los siguientes:

- **Codeigniter**. Es un *framework* ligero con un alto rendimiento y las funcionalidades más habituales, sin forzar excesivamente a un desarrollo concreto por parte de los desarrolladores.

- **Kohana**. Está basado inicialmente en **Codeginiter**. Incorpora múltiples mejoras y utiliza funcionalidades de versiones más recientes de PHP.

- **Cakephp**. Este *framework* se inspira en Ruby on Rails para tener una funcionalidad similar, bajo la filosofía de *convention over configuration*.

Lectura complementaria

Más información sobre *convention over configuration* en:

http://en.wikipedia.org/wiki/Convention_over_configuration

En el enlace siguiente, se puede ver una comparativa entre diferentes *framework*s:

http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks#PHP_2

En este caso, vamos a utilizar el *framework* **Codeigniter**.

6.6. Especificaciones

¿Qué quiere el cliente?

El cliente nos solicita una aplicación en Facebook. Siente que eso puede dinamizar la imagen del producto. Le gustaría tener más información sobre los potenciales clientes de su producto, pero no sabe exactamente cómo relacionarlo con Facebook.

¿Qué se le propone?

Le proponemos crear un concurso en Facebook. La gente se hará socio para participar. Se obtendrán datos sobre los usuarios y se dinamizará la marca.

6.7. Diseño

La fase de diseño nos ayudará a comunicarnos con el cliente. Es importante que el cliente no perciba el diseño como el producto.

El diseño

El diseño es un artefacto del proceso de desarrollo. Sirve para comunicarse con el cliente. **No es el producto final**. El producto final es la aplicación en funcionamiento. Lo mismo sucede con los documentos de especificaciones, presentaciones y similares.

Si esto sucede una vez finalizado el diseño el cliente percibirá que la aplicación ya debe ser funcional. Es responsabilidad nuestra comunicarnos con el cliente de forma adecuada.

6.7.1. Esbozo general

Nos interesa concretar cuanto antes la funcionalidad con el cliente y garantizar que vamos en la misma dirección. Podemos esbozar el diseño de los elementos necesarios y su interacción mediante *wireframes*.

La ventaja de los *wireframes* es que nos permiten comunicar el diseño al cliente sin perder la perspectiva por los detalles.

Decisiones sencillas

Las decisiones aparentemente sencillas pueden, debido a su aparente sencillez, ser mucho más difíciles de pactar (todo el mundo se ve con capacidad de decidir respecto a ellas). Más información en:

http://en.wikipedia.org/wiki/Parkinson%27s_Law_of_Triviality

Si hay incoherencias o discrepancias en el funcionamiento, conviene enfrentarlas lo antes posible en el proceso de desarrollo.

Aunque puede haber cambios en el diseño, los *wireframes* nos servirán para tener un marco donde desarrollar esos cambios. De este modo, no se replanteará el rediseño de todos los elementos posibles, sino que la funcionalidad y navegación será básicamente la misma.

Esto nos permite empezar la implementación antes de que el diseño final esté cerrado. Lo que sí tenemos que asegurar es que el cliente comprende correctamente el papel de los *wireframes* y que se compromete con la funcionalidad básica. Esto no abre la puerta a cambios irracionales e ilimitados.

Existen gran cantidad de aplicaciones y servicios web para el desarrollo de *wireframes*. En este proyecto, vamos a utilizar **Lumzy**.

Algunas otras opciones son las siguientes:

1) Como servicio web

- **Lovely Charts**
- **Cacoo**
- **Mocking Bird**
- **Mockflow**

2) Como aplicación nativa

- **Balsamiq**
- **Omnigraffe**
- **Lovely Charts**

También podemos utilizar un programa habitual de diseño o ilustración (como Fireworks, Adobe Photoshop o Adobe Illustrator), aunque recomendamos utilizar un programa específico de producción de *wireframes*.

Una última opción es utilizar la propia maquetación html con CSS, con unos estilos simplificados para realizar los propios *wireframes*. Esto es factible cuando los encargados de la maquetación son el mismo equipo encargado del diseño.

6.7.2. Diseño final

Dentro del marco de los *wireframes*, podemos variar en cierto grado el diseño. Es importante matizar que esto no da libertad total para cambiar cualquier aspecto del proyecto. Existen partes del diseño de una aplicación o página web que están íntimamente ligadas a su funcionalidad. Introducir cambios de funcionalidad graves a última hora es una receta para el desastre.

Es importante que se mantenga una coherencia global en el proyecto, y los cambios demasiado locales en el diseño sin tener en cuenta la aplicación como un todo suelen romper esa coherencia.

6.8. Iteraciones

Conforme a la metodología ágil, el proyecto se estructurará en torno a una serie de iteraciones de corta duración que irán incrementando paulatinamente las funcionalidades de la aplicación. Al final de cada operación, tendremos algo tangible.

6.8.1. Iteración 1: definición del flujo de la aplicación e implementación mínima

Una de las primeras funcionalidades que necesitamos es la navegación por las diferentes páginas del proyecto según el funcional.

En este caso, el flujo de navegación será el siguiente:

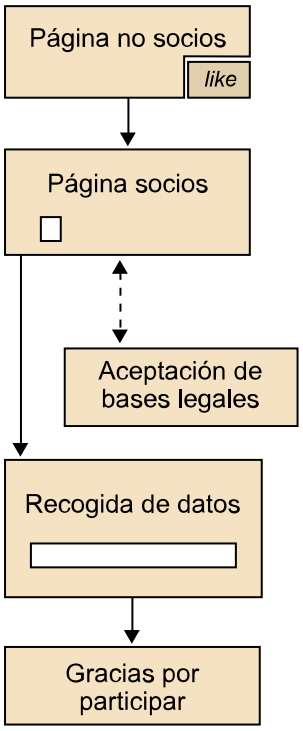
1) Accedemos a la página de presentación de la aplicación. Para acceder a ella necesitamos hacer clic en “Me gusta” y dar los permisos adecuados.

2) Vemos una página especial solo para socios. En ella, constan las bases legales y hay que aceptarlas antes de poder seguir adelante en la aplicación.

3) Al aceptar las bases legales, procedemos a rellenar los datos que nos solicita la aplicación.

4) Una vez rellenados los datos, se da las gracias al usuario y se nos informa de los siguientes pasos, si los hubiera.

Instalamos **Codeigniter** y procedemos a crear los controladores y métodos necesarios.



Definimos las url que aceptará la aplicación para crear los métodos correspondientes en el controlador principal de la aplicación.

Lectura complementaria

En el enlace siguiente, encontraréis una guía de usuario sobre el controlador:

http://codeigniter.com/user_guide/general/controllers.html

Utilizamos *pretty urls* para los diferentes pasos:

- `/ /`. La ruta base de la aplicación. Al entrar nos dirige aquí.
- / `socio`. La ruta para los usuarios que ya han hecho clic en "Me gusta".
- / `formulario`. Esta página nos presenta los datos por rellenar.
- / `gracias`. Pantalla final de la aplicación. Se nos agradece la participación y se nos informa de cualquier dato relevante.

En cada una de las url definidas en el controlador, cargamos una vista correspondiente, que simplemente tendrá una página vacía con el nombre de la url. Cargamos cada vista desde el método correspondiente del controlador.

Lectura complementaria

En el enlace siguiente, encontraréis una guía de usuario sobre la vista:

http://codeigniter.com/user_guide/general/views.html

Al final de esta iteración, contamos con el flujo real de toda la aplicación funcional, si bien el contenido de cada uno de los pasos no es el real (ni siquiera están los elementos más básicos). En las siguientes iteraciones, iremos desarrollando la funcionalidad correspondiente a cada uno de estos pasos.

Podemos navegar por las diferentes páginas accediendo a **www.nuestroservidor.com/el_nombre_de_la_página (/, socio, formulario, gracias)**.

6.8.2. Iteración 2: crear la aplicación en Facebook

Hasta ahora, hemos accedido a la aplicación, que es exactamente igual a una aplicación web. En esta iteración, vamos a integrar nuestra página para poder acceder desde dentro de Facebook y empezar a verla como realmente la verán los usuarios finales.

Nos interesa ejecutar este paso cuanto antes para detectar posibles inconsistencias o limitaciones de Facebook y poder solucionarlas lo antes posible.

Creamos una cuenta de Facebook de test, con la que crearemos la aplicación. Es interesante que los amigos de esta cuenta sean pocos y de prueba, ya que verán notificaciones sobre el uso de la aplicación a medida que la vayamos desarrollando.

Accedemos con la cuenta de test a www.facebook.com.

Creamos la aplicación en https://developers.facebook.com/apps y creamos en **Crear nueva app**.

Configuramos la aplicación. Necesitamos un nombre para la aplicación y un *namespace*, que determinarán la url final de la aplicación.

En este proceso, tenemos que solucionar algún CAPTCHA[13] y proporcionar un número de móvil o de tarjeta de crédito para que Facebook compruebe que los datos son reales.

(13)Acrónimo de completely automated public turing test to tell computers and humans apart.

CAPTCHA

Es un sistema automatizado para reconocer que a una página va a acceder una persona y no un programa. Es muy útil para minimizar el *spam* en comentarios o para optimizar recursos para los usuarios reales frente a *crawlers* u otro tipo de agentes de software.

El proceso puede llevar su tiempo y aquí estamos a merced de Facebook. Según nuestra experiencia, puede tardar desde unas pocas horas hasta algunos días. Recomendamos validar la cuenta de todas las formas de las que dispongamos si con el paso de los días no recibimos el mensaje en el móvil.

Configuramos la aplicación para que apunte adonde se aloje nuestra página.

Recordemos que recientemente Facebook solo acepta aplicaciones con el protocolo https://.

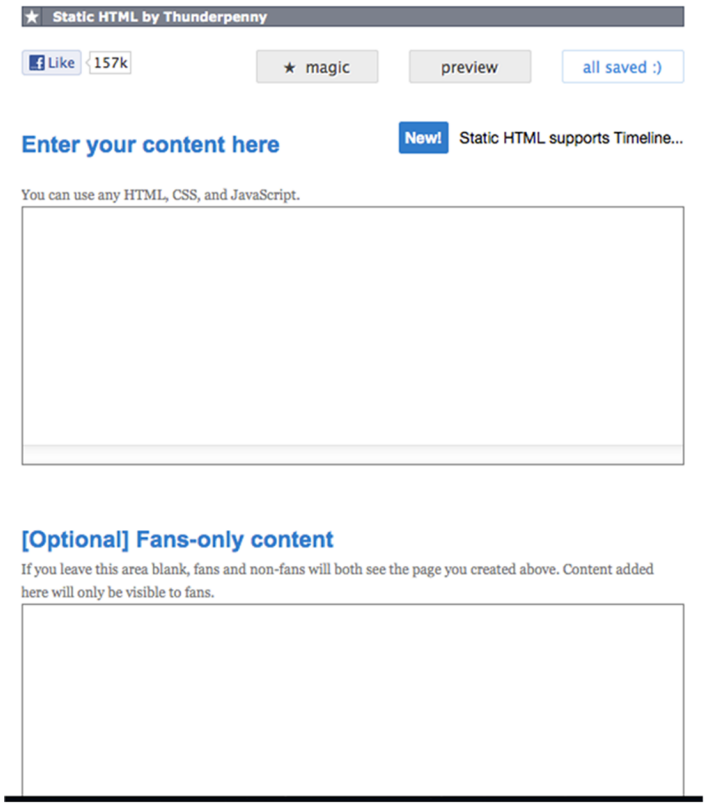
Webs recomendadas

Algunas buenas referencias para dudas durante el desarrollo de una aplicación en Facebook son las siguientes:

https://developers.facebook.com/docs/

http://facebook.stackoverflow.com/

Una vez la aplicación ha sido creada, podemos instalar alguna aplicación más que nos ayudará a gestionarla. Una de estas aplicaciones es Static HTML by Thunderpenny. Esta aplicación nos permite configurar dos pantallas de código html, una para los usuarios que no han hecho clic en "Me gusta" y otra para ser vista cuando ya han hecho clic en "Me gusta".



En la página de la *app*, veremos un mensaje en el lateral "Welcome". Al hacer clic en él podremos empezar a utilizar esta aplicación.

Con esta iteración, ya tenemos las rutas básicas funcionando con contenido no real dentro de una aplicación de Facebook visible.

En las próximas iteraciones, vamos a utilizar esta aplicación para poner el código correspondiente para no socios y para socios.

6.8.3. Iteración 3: *wireframes* - pantalla para no socios

En esta y en las siguientes iteraciones, vamos a ir definiendo a grandes rasgos los elementos presentes en cada una de las pantallas de la aplicación, basándonos en el flujo de uso de la aplicación.

En esta iteración, queremos mostrar la primera pantalla que verán los usuarios de la aplicación. Esta debe de ser atractiva y tener una llamada a la acción clara, ya que es lo que debe convencer a los visitantes para que hagan clic en "Me gusta".

Ponemos un texto que indique al usuario que ha de hacer clic en "Me gusta" para entrar a la aplicación.

En el siguiente *wireframe*, podemos ver lo simple que es esta página. La columna de la derecha representa el UI[14] de Facebook.

(14)Acrónimo de user interface.

Con este *wireframe* definido, podemos hacer una implementación muy básica. En este caso, nos bastará simplemente con una imagen. Implementamos en html los elementos mínimos que imiten el *wireframe*. Editamos la vista para incluir en el html una imagen y el código html del botón "Me gusta".

A modo de ejemplo, el html podría tener un aspecto similar al siguiente:

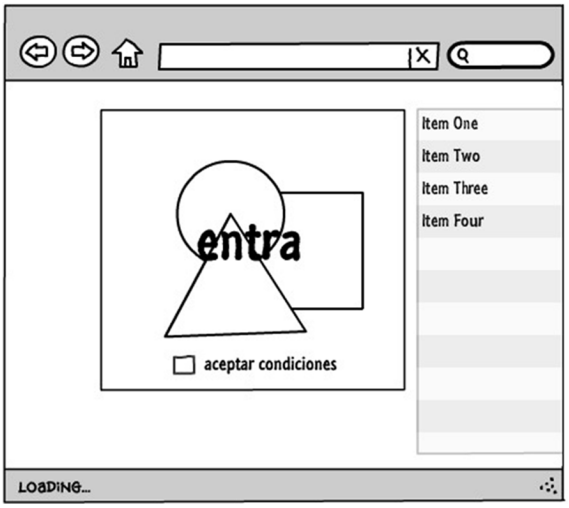
```
<! DOCTYPE html>
<html>
<head>
</head>
<body>
<div>
<img src="https://nuestrservidor.co/imgs/nosocio.jpg" />
</div>
</body>
</html>
```

Una vez tengamos la página implementada, podemos hacer **Guardar como** desde el navegador para tener la página como html estático.

Hacemos **Copiar Pegar** de este html estático y, en la página de **static_html_plus**, lo pegamos en la casilla correspondiente a los usuarios no fans (la primera).

Al entrar en la aplicación, veremos ahora este código estático hasta que pulsemos en "Me gusta".

6.8.4. Iteración 4: *wireframes* - pantalla para socios



La pantalla para usuarios que ya han hecho clic en "Me gusta" puede contener más información. Incluiremos la aceptación de las bases legales como condición para pasar a la aplicación en sí misma.

Para llevar a cabo una implementación básica de este *wireframe* necesitaremos:

1) Un enlace o botón que vaya a la siguiente página (formulario) en el flujo definido.

2) Un código Javascript que valide que el *checkbox* de aceptación de condiciones está marcado.

3) Un enlace al lado del *checkbox* de aceptación de condiciones que nos llevará (aunque aún no existe) a las bases del concurso o aplicación.

4) Una imagen de fondo o la maquetación correspondiente en CSS para una implementación mínima de la parte gráfica de este *wireframe*.

Una vez ejecutada esta implementación, guardamos la página como código estático y lo pegamos en la casilla correspondiente de **static_html_plus** al contenido de solo fans (el segundo apartado).

Si ya somos fans de la aplicación, veremos este contenido al entrar en ella. Con esto, acaba el uso de la aplicación **static_html_plus**. Si más adelante queremos retocar alguna de estas dos páginas, deberemos volver a copiar y pegar el html estático en su configuración.

6.8.5. Iteración 5: *wireframes* - formulario de recogida de datos

Definimos ahora los datos que queremos recoger y cómo serán mostrados al usuario de modo simplificado con la ayuda de un *wireframe*.

Para hacer una implementación mínima de la funcionalidad del formulario necesitaremos:

1) Programar los formularios desde PHP para incluir su validación, el orden en el que mostrarlos y cuáles son obligatorios o no.

Formularios

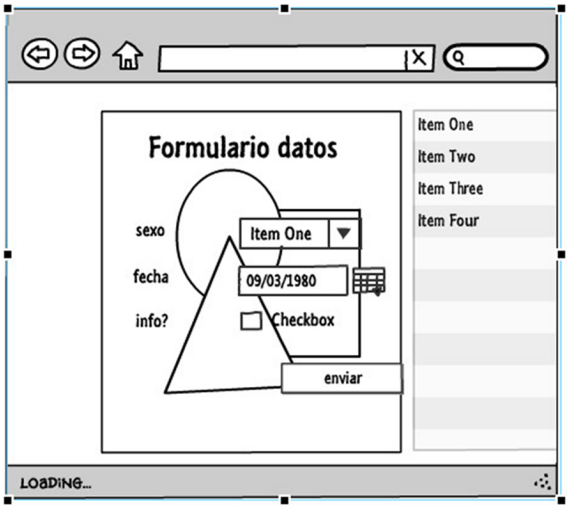
En el caso de **Codeigniter**, disponemos de facilidades para realizar formularios y validaciones complejas de estos.

En los enlaces siguientes, encontraréis guías del usuario referentes a formularios y validación:

http://codeigniter.com/user_guide/helpers/form_helper.html

http://codeigniter.com/user_guide/libraries/form_validation.html

Otros *frameworks* tienen utilidades similares. Si no disponemos de alguno de ellos, deberemos implementar esta funcionalidad nosotros mismos.



2) Validar desde Javascript que los formularios cumplen las mismas validaciones para evitar enviarlos para hacer la comprobación.

6.8.6. Iteración 6: respaldo en base de datos

Hasta ahora, las páginas no guardaban ningún tipo de información del usuario; simplemente eran navegables. En esta página, recogemos datos del usuario.

Para eso, tendremos que recoger los datos del formulario y guardarlos en la base de datos de la aplicación.

Al instalar **Codeigniter**, ya hemos introducido los datos de una base de datos. Ahora necesitaremos lo siguiente: crear una tabla para guardar la información y acceder a esa tabla para guardar los datos desde el método de nuestra página.

Crear la estructura de la base de datos

Accedemos con una aplicación cliente de gestión base de datos a la base de datos.

Clientes de base de datos

Existen muchos clientes de base de datos. Algunos de uso habitual son los siguientes:

- http://dev.mysql.com/downloads/gui-tools/5.0.html: de los creadores de Mysql, por lo tanto muy recomendable para gestionar y trabajar con esta base de datos. Son una serie de utilidades disponibles para múltiples sistemas operativos que se instalan como aplicaciones nativas.
- http://www.phpmyadmin.net/home_page/index.php: una aplicación web que se instala en el servidor y permite gestionar allí, con un *login* y una contraseña, las bases de datos presentes en el servidor.

Recomendamos siempre que sea posible usar aplicaciones nativas para gestionar la base de datos. Suelen ser más eficientes, lo que se traducirá en una mayor eficiencia al desarrollar y ejecutar tareas de mantenimiento.

Allí creamos una tabla de nombre "registros" con los campos necesarios y sus tipos. En este caso, según el diseño básico del que disponemos, los campos son los siguientes:

- **sexo** (hombre/mujer)
- **fecha** (YYYY-MM-DD)
- **recibir más info** (sí/no)

La aplicación generará por nosotros el código SQL[15] necesario para crear las tablas. Un desarrollador puede crear la tabla mediante scripts o algún otro sistema de abstracción que acabe generando este código del mismo modo.

En general, suele ser más práctico e intuitivo generar la estructura inicial de tablas mediante herramientas gráficas, pero no es la única opción.

(15)Acrónimo de Structured Query Language.

SQL

Es un lenguaje de programación empleado para tratar con bases de datos relacionales.

Guardar los registros en la base de datos

Las aplicaciones desarrolladas en **Codeigniter** utilizan, como ya hemos visto antes, el modelo vista controlador. Hasta ahora, hemos utilizado los controladores para distribuir el flujo de la aplicación.

Recordemos que hicimos un solo controlador con un método para cada una de las páginas disponibles en la aplicación.

También hemos utilizado las vistas para maquetar cada una de las páginas de las que consta la aplicación. A su vez, estas vistas eran cargadas en los controladores para que aparecieran en las páginas correspondientes.

En este caso, vamos a utilizar el último elemento de esta manera de estructurar código: los modelos.

Los modelos abstraen el modo como accedemos a los recursos en una aplicación. Una forma práctica de pensar en los modelos es pensar que son actores que toman parte en la acción. Así, no pensaremos en un modelo base de datos, sino en un modelo usuario, que dispondrá de los métodos "Insertar" o "guardar_datos".

Lecturas complementarias

En los enlaces siguientes, vais a encontrar una guía del usuario sobre el modelo vista controlador y modelos:

http://codeigniter.com/user_guide/overview/mvc.html

http://codeigniter.com/user_guide/general/models.html

Creamos un modelo "usuario" y un método "guardar_datos", que aceptará los datos recogidos en el formulario y los guardará en la base de datos.

Lectura complementaria

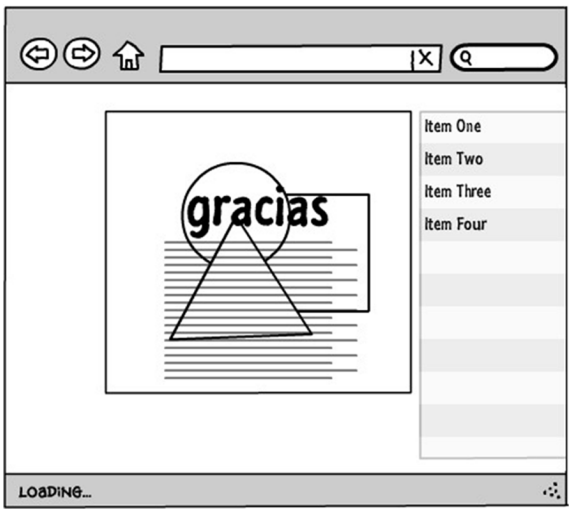
En el enlace siguiente, vais a encontrar una guía del usuario sobre base de datos:
http://codeigniter.com/user_guide/database/index.html

Para acceder a la base de datos desde un modelo **Codeigniter**, nos proporciona algunas utilidades que nos facilitan la tarea. Son totalmente opcionales, pero pueden agilizar el proceso de desarrollo si tenemos familiaridad con ellas.

6.8.7. Iteración 7: *wireframes* - pantalla final

En esta pantalla, simplemente informamos al usuario de que sus datos se han enviado correctamente. Es también el lugar indicado para avisarle de futuras acciones o eventos que esperar. Le damos las gracias por haber participado en el concurso o haber utilizado la aplicación.

El *wireframe* correspondiente sería el siguiente:



6.8.8. Iteración 8: bases del concurso

Añadiremos una página más a las páginas definidas hasta ahora para incluir un enlace a las bases legales de la aplicación. Una forma sencilla de implementarlo es enlazar directamente a un archivo PDF[16].

(16)Acrónimo de portable document format.

PDF

Es un formato de Adobe, considerado uno de los estándares de facto para el intercambio de archivos. Por lo tanto, es un buen formato, ya que se puede abrir en cualquier sistema y entorno y los usuarios están familiarizados con él.

De este modo, no habrá que modificar el acceso a la página. Simplemente, pondremos el enlace en la vista correspondiente.

Este archivo puede ser proporcionado por el propio cliente. Dependiendo del tamaño del cliente y del ámbito de su negocio, podrá disponer de un departamento específico encargado de tratar cuestiones legales. En este caso, es muy recomendable que sea este departamento el encargado de elaborar este documento. Nosotros simplemente lo enlazaremos para que el usuario pueda leerlo antes de aceptar las condiciones.

Diseño final

Hasta este punto, tenemos una implementación funcional completa. Seguimos el flujo de la aplicación que definimos y hemos hecho una implementación básica pero totalmente funcional partiendo de los *wireframes* en continua comunicación con el cliente. En este momento, el diseño final debería estar cerrado para poder maquetar las páginas mediante CSS y pulirlas para que estén visibles como las va a experimentar el usuario final.

6.8.9. Iteración 9: diseño final - pantalla para no socios

A partir de los *wireframes*, que ya son funcionales, solamente nos falta maquetar de forma específica el diseño final de la pantalla previa para los no socios.

Según el *wireframe* anterior, en este caso se trata simplemente de una imagen, por lo que prestamos especial atención a los tamaños de esta.

Una vez exportada la imagen, guardamos el html y lo actualizamos entrando en la pestaña de Facebook de **Static HTML**. En este caso, irá en el código que no es exclusivo para fans (el primer recuadro).

6.8.10. Iteración 10: diseño final - pantalla para socios

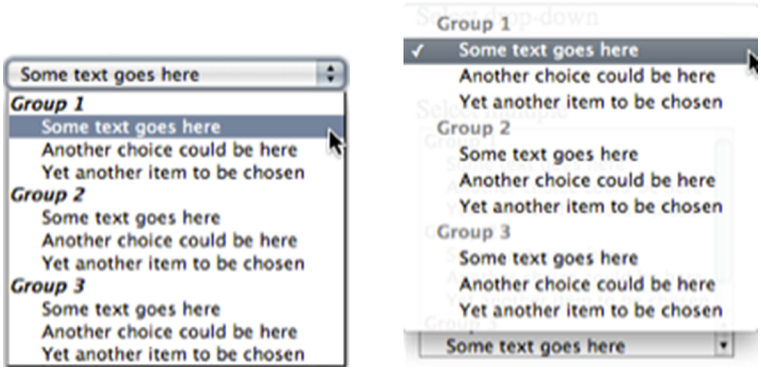
La pantalla para gente que ya ha hecho clic en "Me gusta" consta de una imagen más grande que la de no socios. Al margen del *check* obligatorio y del enlace a las bases legales, se ha podido maquetar prácticamente con una sola imagen. También consta de un texto que invita a entrar en la aplicación que, en esta iteración, será sustituido por el texto definitivo.

Debido a que es básicamente una imagen, será muy sencilla de maquetar. Simplemente, exportamos la imagen en los tamaños adecuados.

6.8.11. Iteración 11: diseño final - formulario de recogida de datos

En esta iteración, actualizamos el diseño del formulario de recogida de datos de la aplicación. Esta es una de las pantallas que probablemente consta de más detalles. Es importante que los campos sean atractivos y fáciles de rellenar y que quede claro el formato que se espera en cada uno de ellos.

Debemos tener en cuenta también que los controles de formularios (*inputs*, *selects*, *radiobuttons*, entre otros) utilizan controles nativos en cada sistema operativo. Esto puede hacer que un mismo formulario tenga aspectos ligeramente diferentes, por ejemplo, en un entorno Mac Os o en uno Windows. Los elementos pueden ser diferentes incluso entre navegadores.



Mismo componente select nativo en distintas plataformas

Dependiendo de la aplicación, esto puede no ser un problema. Al fin y al cabo, la funcionalidad es exactamente la misma, pero es importante comunicarle estas opciones al cliente.

Si se desea exactamente el mismo diseño por motivos de *Look&Feel* de la aplicación, podemos usar alguna librería para ayudarnos a tratar las diferencias entre los formularios de los diferentes entornos.

Lectura complementaria

Más información sobre *Look&Feel* en:

http://en.wikipedia.org/wiki/Look_and_feel

Un diseño exacto de estos componentes de formulario puede no ser necesario y supone un tiempo extra de desarrollo, según la familiaridad de los desarrolladores con las librerías empleadas.

Jnice

Una buena opción para maquetadores y programadores cliente es Jnice.

Permite maquetar componentes a medida. De esta forma, conseguimos que se vea y se simule el comportamiento en el componente propio mientras que por debajo se utiliza el componente nativo de cada plataforma.

6.8.12. Iteración 12: diseño final - pantalla final

La pantalla final es el último paso para acabar la maquetación. Es una pantalla simple, con unos pocos elementos. Damos las gracias al usuario e incluimos un texto donde le avisamos de la fecha del resultado del concurso y futuros eventos.

6.8.13. Iteración 13: protección de datos

Es importante que guardemos solamente los datos necesarios para el concurso y que sigamos la Ley Orgánica de Protección de Datos, LOPD. Las multas por incumplir esta ley son increíblemente altas.

También es importante que la seguridad bajo la que está la aplicación y restos de datos sean las adecuadas.

Antes de lanzar la aplicación, conviene dar un último repaso a la aplicación para detectar algún error de seguridad que pueda haber pasado inadvertido durante el desarrollo de la misma.

Desde las primeras iteraciones funcionales ya hemos podido probar el funcionamiento de la aplicación. En este punto, podemos realizar las últimas pruebas o ejecutar los tests unitarios una última vez antes de lanzar la aplicación.

6.8.14. Iteración 14: compartir en el muro o con los amigos

En la última pantalla de la aplicación, puede ser útil proponer al usuario compartir estos datos con algún amigo o en su propio perfil de Facebook. Si el concurso es atractivo para él probablemente lo sea también para sus amigos.

Para incluir esta opción, utilizamos uno de los *widgets* de Facebook.

Ejemplo de botón de compartir de Facebook

Web recomendada

Aquí podemos ver las diferentes opciones disponibles para colocar el botón en la página:

http://developers.facebook.com/docs/share/

Copiamos el código que Facebook nos proporciona y lo añadimos en el html de la última pantalla. Probablemente, serán necesarios algunos pequeños retoques en la maquetación para situar el botón justamente donde lo queremos.

6.8.15. Iteración 15: lanzamiento y puesta en sociedad

Hacemos pública la aplicación y la viralizamos. Incluimos la dirección en material promocional de la empresa o *mailing* o contactamos con una empresa de medios que viralice la campaña.

A partir de aquí, tendremos que recopilar diferentes métricas para ir viendo el éxito de la aplicación.

6.8.16. Iteración *n*: ¿...?

Si fuera necesario modificar o ampliar funcionalidad para un mejor funcionamiento de la aplicación podemos seguir este mismo modelo de desarrollo, desarrollando iteraciones rápidas que mejoren incrementalmente la aplicación.