GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

PFM04. PLANIFICACIÓN MÓDULOS

Sistemas de Gestión Empresarial
2º Curso // CFGS DAM // Informática y Comunicaciones
Alfredo Oltra

Cicles Formatius

ÍNDEX

Versión: 240111.2338

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

☑ Atención.

Interesante.

PFM03. IMPLANTACIÑÓN

1 INTRODUCCIÓN

Una vez tenemos el sistema base de serie (o con los aportes de módulos de terceros) es el momento de empezar a trabajar en aquellos aspectos solicitados por el cliente que necesitan ser implementados desde cero o mediante modificaciones a partir de módulos ya existentes.

Para ello vamos a empezar planificando el trabajo que vamos a realizar. Esta planificación tiene que ser entendida como una fuente de información más para alcanzar el objetivo final, pero no como algo cerrado que hay que seguir a “pies juntillas”. Los propios modelos alámbricos o bocetos que se crean en esta etapa son simplemente eso, bocetos: ideas generales de como deberían trabajar nuestros módulos, que pueden (y casi que deben) ir adaptándose en función de como vaya avanzado el desarrollo, las necesidades del proyecto e incluso las tecnologías a usar.

Como ya se comentó en la fase 1, una planificación no debe ser algo cerrado, pero tampoco tan abierto que de la idea inicial a la final no quede nada. Eso implicaría que se ha partido de una mala planificación, realizada sin pensar y que, con casi total seguridad, el tiempo invertido en el desarrollo habrá sido mayor del esperado.

2 OBJETIVO

El objetivo de esta fase es realizar una planificación de dos módulos (o uno en el caso de que la complejidad sea muy alta) que vamos a desarrollar para ser conscientes de cual es el objetivo final al que queremos llegar en cada uno de ellos. De esta fase deben salir dos documentos, uno por cada módulo, que sirvan de hoja de guía, de referencia y de apoyo para la implementación de cada uno de ellos.

3 WIREFRAMES

Los bocetos o modelos alámbricos (*wireframes*) son vistas simplificadas de lo que aparecerá en cada pantalla del desarrollo final. Su objetivo es bastante variado: ayudan a establecer funcionalidades y comportamientos, proporcionan una forma rápida de ver los conceptos de las interfaces y categorizan la importancia de los elementos, etc

En general representan una pantalla, pero pueden centrarse únicamente en un trozo. La idea que hay detrás de ellos es que, al evitar todo tipo de elemento estético (fuentes, colores, …) los desarrolladores puedan centrarse en discutir como ese elemento se tiene que comportar.

Hoy en día la mayoría de los bocetos se utilizan para describir plantillas o, lo que es lo mismo, un esquema utilizado para representar un tipo de información similar.

Figura 1. Ejemplo boceto en dos pasos

En un primer paso los bocetos se componen simplemente de rectángulos que definen las áreas de contenido. Cada una de estas áreas debe hacer referencia a una única funcionalidad (o como mucho dos). En un segundo paso es posible incluir de manera esquemática ciertos componentes de formulario, como cajas de edición, listas, botones, etc, con el objetivo de ver ciertos alineamientos o superficies. Además a veces también se suelen incluir ciertos datos a modo de ejemplo para confirmar tanto la funcionalidad o espacio disponible.

Una opción más avanzada (muy utilizada ~~el~~ desarrollo web) es la creación de *mockups* o maquetas. Su objetivo es detallar más allá de lo que hace un *wireframe* el elemento a implementar. De hecho, son recreaciones fotorrealísticas de como tiene que quedar ese elemento, atendiendo ya a detalles tan concretos como los colores o las fuentes. Durante mucho tiempo se han desarrollado con el uso de aplicaciones del ámbito del diseño como *InkJet* o *Gimp* o *Photoshop* pero hoy en día se va un paso mas allá y es posible la creación de maquetas semifuncionales o al menos funcionales a nivel de interfaz (abrir nuevas páginas, desplegar contenidos...)

4 MAPA DEL MÓDULO

Un mapa del módulo viene a ser una extrapolación de los mapas del sitio web (*site maps*) que representan como está distribuida la información en la web[1]. En este caso de manera jerárquica, es decir, elementos que son hijos de otros elementos.

Al igual que los *maps* son formas geométricas (en este caso suelen ser rectángulos) conectados entre sí. Los rectángulos representan las diferentes páginas del site, mientras que las líneas muestran como los usuarios navegan a través de la información.

Cada uno de los nodos puede representar:

- Una página. Es la unidad básica del sitio. Se etiquetan con el nombre de la página.
- Grupo de páginas. Un nodo puede representar un grupo de páginas que comparten una funcionalidad similar. Se suelen representar como un conjunto de rectángulos superpuestos.
- Recursos. Es habitual que además de las páginas HTML, los sitios permitan descarga de documentos: catálogos, fotos, etc. Se pueden representar mediante rectángulos con diferentes fondos.
- Categoría. Si la complejidad del sitio es muy alta y sólo buscamos representar el mapa de una manera general, es posible recurrir al uso de categorías. Estas consistirían en un nodo que representa una gran cantidad de páginas (muchas más de las que indica un grupo)

Los enlaces entre los nodos pueden representar dos cosas: o una simple conexión jerárquica, o una conexión de navegación. En el primer caso, se da por sentando que puede ser posible acceder a un página de varias maneras posibles y no sólo la indicada por la jerarquía.

Es posible mejorar el mapa incluyendo en posteriores versiones capas que proporcionen más información sobre cada una de las páginas. Por ejemplo, si tienen plantillas comunes, si su acceso debe aparecer en home, si pertenecen al backend o al frontend, etc.

En el caso de *Odoo*, muy encorsetado por lo que se refiere al layout de la página principal, el mapa debe reflejar cosas como las opciones de los menús superiores, las posibles conexiones entre ellas, las diferentes vistas, etc.

1 Hay que recordar que Odoo no es ni más ni menos que un servidor web.

5 DIAGRAMAS DE FLUJO

Los diagramas de flujo (*flowcharts*) van a permitir describir las tareas que han de ser soportadas por el módulo, así como ver como fluye la información a través del mismo. Se podría decir que describen la capa de negocio, lo que hace el módulo y qué pasos necesita para realizarlos. Serán implementados por los controladores.

Gráficamente hablando son parecidos a los conceptos anteriores: una serie de cajas conectadas entre sí, pero en este caso aparece un variación importante, la direccionalidad.

Cada caja indica un paso a realizar. Es la unidad básica del diagrama: se le proporcionan unos datos y responde con otro. La dirección indica cual será el siguiente paso. Se les etiqueta con un nombre que describa cual es su funcionalidad. De entre ellos hay que destacar el paso inicial y el final, donde se indican respectivamente, el entorno de partida y los resultado solicitados. Sólo debe haber un estado inicial y un final y para diferenciarlos es habitual dibujarlos como rectángulos de esquinas redondeadas (aunque también está la opción de dibujar un círculo).

Otro tipo de elemento importante son los puntos de decisión. Se representan como rombos, y lo habitual es que muestren decisiones Si/No, aunque no necesariamente.

Las líneas que unen las cajas deben tener algún tipo de flecha que marque su direccionalidad y, por claridad, conviene que no se crucen.

6 GITHUB PAGES. JEKYLL

GitHub Pages es un servicio de alojamiento web de páginas estáticas que proporciona *Github* de forma gratuita. Dicho de otra manera, es un servicio que nos va a permitir desplegar una página web en internet siempre y cuando su contenido sea fijo, es decir, no cambie en función del usuario, de los datos almacenados en un servidor, etc.

En principio lo único que hay que hacer es indicarle a Github qué parte de nuestro código es el que contiene los documento HTML que conforman la página para que cuando se ~~cuando se~~ acceda a la *url* configurada se visualice no el código sino el documento renderizado.

En principio estos documentos son ficheros HTML, pero para simplificar el trabajo del creador (generalmente desarrolladores más centrados en programar que en documentar) es habitual utilizar algún generador de sitios web como Jekyll.

De hecho, aunque el uso puede ser cualquiera, el objetivo real es el alojamiento de páginas centradas en la información sobre la aplicación con la que comparten repositorio o en información sobre los creadores: curriculums, portffolios, etc

Jekyll es una aplicación que, a partir de documentos Markdown, documentos de texto plano, genera la estructura completa de un sitio web. Es decir, que si disponemos de una documentación asociada al proyecto en formato *md*, añadiendo cierta configuración y los *layouts* de estructura de la página (cómo queremos que se vea) es capaz de generar todos los HTML necesarios para mostrarlos renderizados en el navegador.

Es posible utilizar HTML directamente o cualquier generador HTML diferente a *Jekyll* (por ejemplo, uno de uso muy habitual es Hugo), pero el que trabaja de una manera más sencilla con *Github* es él.

6.1 Instalación de Jekyll

La instalación de *Jekyll* en si es sencilla, pero su dependencia de *Ruby* (y más concretamente de versiones específicas de librerias concretas de *Ruby*) la puede complicar ~~debiado~~ a la dificultad de tener instaladas simultáneamente diferentes versiones en el mismo sistema.

En esos casos, utilizar algún sistema de gestión versiones como rvm o asdf es muy recomendable (por no decir necesario).

En nuestro caso, para evitar problemas, la solución más sencilla es arrancar un contenedor de *Jekyll* con el que poder trabajar: todo está ya configurado, con las dependencias correctas y puede ser levantado o apagado bajo demanda. Para arrancarlo simplemente:

- descomentar del fichero *.services* de *odoodock* la linea en la que aparece *jekyll*
- levantar de nuevo los contenedores con *./up.sh*

6.2 Creación de un sitio web

Una vez instalado el primer paso es la creación de la estructura del proyecto de sitio web (el scafoldding). Para ello, siguendo las instrucciones de la documentación de *odoodock*

1. Desde la carpeta *odoododck*, ejecutamos *bash* desde dentro del contenedor *jekyll*

   ```
   $ docker exec -it odoodock-jekyll-1 bash
   ```

2. Accedemos a la carpeta del módulo (cada módulo tendrá su propia documentación)

   ```
   > cd MI_MODULO
   ```

3. Creamos el nuevo sitio

   ```
   > jekyll new docs
   ```

   Para una mejor compatibilidad con github es conveniente que el nombre del sitio (de la carpeta del proyecto) sea *docs*.

4. Añadimos el paquete *webrick*

   ```
   bundler add webrick
   ```

   bundler es el instalador de paquetes de ruby, algo así como *pip* o *npm* para *python* o *javascript* respectivamente.

5. Instalar las gemas. Uno de los ficheros creados con los comandos anteriores es *Gemfile*, que define todas aquellas *gemas* que son necesarias para nuestro proyecto *jekyll*.

   ```
   bundler install
   ```

☑ Se podría decir que las *gemas* son el equivalente *ruby* a los paquetes en *python o javascript,* asi como el *gemfile es el requirements.txt* de *python* o el *package,json* en javascript.

6.3 Compilación del proyecto

Al partir de documentos *md*, es necesario que se realice un proceso de compilación para conseguir documentos *HTML; javascript y CSS* que serán los que renderizará el navegador

Lo más habitual es poder realizar la compilación en local y, una vez tenemos el resultado que deseamos, subir los documentos *md* (nuestro código fuente) a github, y configurar *Github Pages* para que detecte esa nueva subida y realice la compilación a través de una *Github Action*.

Para ver la evolución del trabajo en local *jekyll* proporciona la opción *serve*, que construye y sirve la página web. Para ello, desde un terminal:

```
docker exec -it odoodock-jekyll-1 bash
> cd MI_MODULO\docs
> jekyll server
```

Podemos acceder a este servicio desde la url http://localhost:400X/, donde X será un número del 1 al 5, generalmente 1.

6.4 Plantillas

El objetivo de usar *jekyll* es faciliar la creación de sisitos web, así que lo habitual es dejar todo lo relativo al diseño a una plantilla ya definida. Es posible encontrar por internet varias, algunas recomendadas por el propio github, o por ejemplo en jekyllthemes.

La propia plantilla indica cual es el procedimiento para su instalación, aunque, en líneas generales suele consistir en:

1. Añadir en el fichero *Gemfile* la gema de *Github Pages*

   ```
   gem "github-pages", group: :jekyll_plugins
   ```

   ☑ Es muy posible que esta línea ya se encuentre (comentada) en el fichero *Gemfile*

2. Comentar la gema de *jekyll* del fichero *Gemfile*

   ```
   # gem "jekyll", "~> 4.2.2"
   ```

3. Añadir en el fichero de configuración de jekyll *_config.yml* la opción *remote_theme* con el tema elegido y, si fueran necesarios, los plugins relacionados. Por ejemplo:

   ```
   remote_theme: temas/mi_tema
   plugins:
   - plugin_necesario
   ```

4. Lanzar la instalación de las gemas

   ```
   bundler
   ```

5. Ajustar la configuración en el fichero *_config.yml*

6.5 Despliegue el GitHub Pages

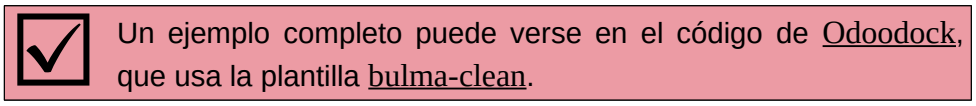
Por último, hay que configurar GitHub para que se realice la compilación. Desde la página del proyecto en GitHub:

1. Acceder a *Settings / Pages*
2. Elegir como *source: Deploy from a branch*
3. Elegir la rama y el directorio donde se encuentra el código fuente.

6.6 Creación de los documentos

La estructura donde se almacenan los documentos a generar va a depender de la plantilla, por lo que la lectura de la documentación asociada es fundamental, pero en general:

- los documentos *md* se distribuyen entre el directorio raíz y en subcarpetas que agrupan los documentos por menús.
- La carpeta *_data* incluye ficheros que contienen información asociada para la construcción del sitio, por ejemplo la estructura de menús y submenús.



7 DESARROLLO

La fase va a consistir en planificar dos (o uno) de los módulos que implementaremos en la fase 5.

☑ El hecho de que sea uno o dos dependerá del grado de complejidad que tengan los módulos. En caso de que la complejidad sea alta (varios modelos, vistas, controladores, creación de informes…) únicamente se desarrollará uno.

Será el profesor el que valore la complejidad de los módulos presentados indicando, si es el caso,la necesidad de ampliarlo o realizar otro. A partir de este momento se hará referencia a un único módulo, teniendo que duplicar todo lo indicado en el caso de que se desarrollen dos.

La documentación a realizar por cada módulo incluirá:

1. Nombre del módulo.
2. Descripción corta.
3. Descripción detallada de todas las funcionalidades a cubrir.
4. Mapa del módulo.
5. Dependencias de otros módulos.
6. *Wireframes* de las páginas a mostrar.
7. Control de accesos. ¿Qué grupos existen?¿Quién puede acceder o no al módulo? ¿ A qué modelos? ¿a qué registros? ¿Con qué permisos?
8. Diagramas de flujo funcionales de las diferentes partes del módulo.
9. Esquema relacional de las nuevas tablas en la base de datos y relación con otras existentes.
10. En caso de que el módulo tenga como función la comunicación con otros módulos, indicar las características de la comunicación: formato, estructura del mensaje, protocolo…

8 ENTREGA

La entrega de esta fase consistirá en la creación de dos webs con al información creada.

Para ello el alumno debe crear dos repositorios en *GitHub* con el nombre de los respectivos módulos y a los que haremos referencia en estos documentos como *remoto_mod1_alumno y remoto_mod2_alumno*. En cada uno de esos repositorios deberá aparecer:

- el *scaffolding* de los dos módulos, configurados con las características definidas en la documentación creada: nombre, resumen, etc
- la publicación en *alumno.github.io/nombre_mod1* y *alumno.github.io/nombre_mod2* de la documentación creada para la planificación de cada uno de los módulos. Esta documentación debe ser generada a partir de ficheros *md* con *jekyll*.

☑ La confirmación de la entrega se realizará copiando las URL de los repo remotos del alumno en la tarea *PFM-F4. Entrega*, dentro de la plataforma Aules. La fecha aconsejada es el 27 de diciembre de 2023 a las 23:55 .

☑ El sistema de entrega se basará en la descarga por parte del profesor de los proyectos alojados en *remoto_alumno* de GitHub. Aún así, para que realizar un control más sencillo de quien ha realizado la fase, será necesario entregar en la tarea *PFM-F4. Entrega* las urls de los repositorios. Sólo se corregirán aquellos que hayan entregado la tarea. .

9 CONSEJOS

- Puede parecer aburrida, pero esta fase va a ser la base para poder centrarnos en la programación en la fase 6. Es fundamental saber en cada momentos qué queremos hacer y hacia donde vamos.

- Los puntos de control son muy importante a partir de esta fase. Es muy posible que defindas módulos muy complejos o muy sencillos y en ambos casos haya que repetir la planficación. Aprovecha los puntos de control para no desciarte mucho del camino. Recuerda que debes ser tu el que los solicites.

- Para cualquier duda sobre el enunciado o sobre posible información a incluir en la empresa puedes utilizar el foro *PFM-F4. Dudas planificación módulos.*