Juny 2014

[FORMULARIS DE GESTIÓ EN LÍNIA:

DE PROGRAMACIÓ ESTRUCTURADA A POO]

Universitat Oberta de Catalunya

José Manuel Cortés Martínez
Enginyeria Tècnica en Informàtica de Sistemes

Consultor: Ignasi Lorente Puchades

A Jèssica, la meva dona, pel seu recolzament constant.

A Sara, la meva filla, per portar llum nova a la meva vida.

Us estimo molt.

ÍNDEX

1. INTRODUCCIÓ 4
2. DESCRIPCIÓ DE L'ESTAT ACTUAL DEL PROJECTE 5
3. OBJECTIUS DEL TREBALL DE FINAL DE CARRERA 8
3.1 OBJECTIUS PRINCIPALS 8
3.2 OBJECTIUS SECUNDARIS 9
4. AVANTATGES DE LA PROGRAMACIÓ ORIENTADA A L'OBJECTE SOBRE LA PROGRAMACIÓ ESTRUCTURADA EN AQUEST PROJECTE 10
5. METODOLOGIA 11
6. ARQUITECTURA DE L'APLICACIÓ 12
7. PLANIFICACIÓ 13
7.1 CALENDARI 13
7.2 FITES DEL PROJECTE 14
7.3 ANÀLISI DE RISCOS 15
8. DIAGRAMES UML 16
8.1 DIAGRAMA DE CLASSES 16
9. IMPLEMENTACIÓ 17
10. API 19
10.1 TIPUS DE DADES 19
10.2 CLASSE FORMULARI 20
10.2.1 Variables de la classe 20
10.2.2 Constructor 20
10.2.3 Mètodes 20
10.3 CLASSE DADA 26
10.3.1 Variables de la classe 26
10.3.2 Constructor 26
10.3.3 Mètodes 26
11. REFACTORITZACIÓ DE FORMULARIS 32
11.1 REFACTORITZAR UN FORMULARI DE RECOLECCIÓ DE DADES AMB LES CLASSES 32
11.2 REFACTORITZAR UN FORMULARI DE MODIFICACIÓ DE DADES AMB LES CLASSES 36
11.3 FER NOUS FORMULARIS FENT SERVIR LES CLASSES 37
11.4 PRIMERES CONCLUSIONS DESPRÉS DE FER LES REFACTORITZACIONS 37
12. EXEMPLE DE FORMULARI 38
13. REQUISITS D'INSTAL·LACIÓ 40
13.1 SOFTWARE 40
13.2 HARDWARE 40
13.3 FORMACIÓ/CONEIXEMENTS 40
14. INSTRUCCIONS D'INSTAL·LACIÓ 41
15. CONCLUSIONS DEL TREBALL DE FINAL DE CARRERA 42
RENDIMENT 42

MIDA DELS FITXERS 43
TEMPS DE TREBALL 44
16. PROJECCIÓ A FUTUR 45
17. CONCLUSIONS PERSONALS 46
ANNEXOS 47
ANNEX 1. LLIURABLES DEL PROJECTE 47
ANNEX 2. LLIBRERIES/CODI EXTERN UTILITZAT 47

1. INTRODUCCIÓ

Quan parlem d'Administració electrònica o e-administració ens referim no només a la incorporació de les tecnologies de la informació i de la comunicació a les oficines de l'administració si no també a una nova forma de relacionar l'usuari i l'administració.

A la Universitat Autònoma de Barcelona (UAB) s'està fent un esforç per incorporar les noves tecnologies a les tasques administratives reduint la burocràcia en paper i alhora redefinir les relacions entre els usuaris i l'administració.

Fruit d'aquesta dinàmica el Director del Departament de Física, Santiago Surinyach, i la Gestora Departamental, Celia Martínez, van redefinir el paper de la secretaria del departament:

Fins llavors el Departament de Física, que es troba dispers per diferents parts de la Facultat de Ciències, disposava d'una petita secretaria a cada lloc on estigués alguna part del Departament. Aquesta presència física, d'una sola persona, havia de donar resposta a tots el protocols i demandes burocràtiques d'aquella part del Departament. Era un sistema clarament ineficient. Amb la reorganització les cinc persones que treballaven per separat es van reunir en un únic despatx. Es van redefinir tasques, processos i protocols per tal d'optimitzar les tasques administratives del departament. Aviat es va veure que calia també oferir una eina de comunicació als membres del departament que, fins ara, tenien una persona de referència de secretaria en el seu entorn físic que ara desapareixia.

Van contactar amb el Servei d'Informàtica de la facultat, lloc on treballo, i van explicar el seu problema. La solució que els vam oferir van ser formularis en línia que fessin recull de les peticions dels membres del Departament.

Aquest projecte, petit i senzill en origen, va començar a incorporar noves funcionalitats seguint les noves necessitats sorgides pel seu ús. Aquest naixement, senzill, i aquesta forma de creixement, mitjançant demandes puntuals, van fer que el disseny de l'aplicació fos del tot ineficient. Simplement no es va pensar pels requeriments que ara se li demanaven.

Un cop feta la reorganització de la secretaria del departament es va fer una petita presentació pública explicant com havia anat l'experiència i si es podia repetir a altres departaments de la universitat. Una petita part de l'exposició es destinava a mostrar breument l'eina de formularis en línia. Tant el llavors gerent, Santiago Guerrero, com la llavors Cap de l'Àrea d'Organització, Maria Genescà, van creure que aquesta eina s'hauria de fer servir a tots els departaments de la universitat i així ho van manifestar.

En una primera fase es va modificar el projecte per poder ser replicat a altres departaments de la Universitat.

En aquest Treball de Final de Carrera es durà a terme una segona fase.

2. DESCRIPCIÓ DE L'ESTAT ACTUAL DEL PROJECTE

Qualsevol membre d'una gran empresa o d'una organització s'enfronta a la necessitat d'haver de realitzar un conjunt de tasques burocràtiques. A la Universitat Autònoma de Barcelona els exemples més clars son:

- Fer una comanda: encarregar la compra d'un material de laboratori, un llibre, etc a un proveïdor.
- Fer el pagament d'una inscripció a un congrés.
- Fer una nota de despesa de viatge: demanar el reintegrament de despeses produïdes durant un viatge de feina. Tiquets de taxi, benzina, restaurants, etc.

L'usuari (professorat i personal tècnic) té la necessitat de fer una tasca burocràtica i fa la petició al seu suport administratiu. Fins ara la comunicació entre l'usuari i l'administració per demanar aquestes tasques s'havia fet verbalment o en el millor dels casos fent servir el correu electrònic.

Això implicava una ineficiència en el servei. El personal d'administració es veia interromput constantment per trucades o visites que feien que fos menys productiu. A més es donava el cas que molts cops les dades subministrades eren insuficients o incomplertes.

Els formularis de gestió en línia funcionen amb dos nivells d'usuaris: usuaris bàsics i usuaris administradors. Els usuaris bàsics son aquells que tenen la necessitat de demanar una tasca, d'omplir un formulari. Un cop omplert el formulari aquest apareix a la pantalla dels usuaris administradors com un formulari pendent de revisar. Un cop han revisat el formulari poden omplir camps que no es mostraven als usuaris bàsics i el poden marcar com formulari revisat. En tot moment l'usuari bàsic pot veure l'estat de la seva petició i en cas que sigui necessari el pot imprimir en format PDF.

Fent servir uns formularis en línia s'han aconseguit les següents millores per als usuaris:

- L'usuari pot fer la petició en qualsevol horari. No cal esperar a que la secretaria estigui en horari laboral.
- L'usuari pot fer la petició des de qualsevol lloc fent servir un navegador web. Ja no cal que es desplaci fins el departament.
- Existeix un registre que emmagatzema els moviments que es fan sobre una sol·licitud concreta. Tant l'usuari com l'administració poden veure quan es va fer la petició i quan s'ha validat per part de l'administració.
- Els formularis incorporen un conjunt de camps obligatoris. Els camps mínims per que es pugui fer la tasca. D'aquesta forma l'usuari sempre proporciona tota la informació d'un sol cop i s'evita haver de fer trucades o consultes posteriors.
- Cada persona té un registre de totes les peticions que ha fet al llarg del temps. Les pot tornar a consultar en qualsevol moment.
- Cada formulari es pot exportar en format PDF. D'aquesta forma els responsables poden signar autoritzant qualsevol petició i aquesta es pot presentar en paper a qui ho demani en aquest format.

Des de el punt de vista tècnic l'aplicatiu de formularis disposa de les següents característiques:

- Programació: s'ha fet servir Programació Estructurada en llenguatge PHP. Per diverses funcionalitats es fa servir Javascript, de forma directa o fent servir la llibreria jQuery.
- Base de dades: es fa servir MySQL. La connexió es fa fent servir les comandes pròpies de PHP.
- Maquetació: la web fa servir HTML i CSS3.
- Validació: hi ha una doble validació. Per una banda es fa servir la validació estàndard a la Universitat Autònoma de Barcelona, CAS , un projecte de Jasig .
  Aquesta eina ens proporciona la validació de tots els membres de la universitat per això ha calgut fer una taula a la base de dades on guardar els usuaris vàlids per entrar a l'eina, per fer una doble discriminació i que només les persones desitjades puguin accedir a la web. En aquesta taula de la base de dades també s'informa del rol de l'usuari ja que hi ha persones que fan formularis i persones que els validen i fan la tasca burocràtica demanada.
  Un cop l'usuari s'ha validat com a membre de la Universitat i com a usuari vàlid de l'aplicació es generen unes variables de sessió amb el seu nom d'usuari i el seu rol, usuari bàsic o administrador. D'aquesta forma podem saber en tot moment qui està fent servir el formulari i quin rol té.
- Multillenguatge: totes les pàgines web per als usuaris del projecte es poden consultar tant en català com en anglès. Les pàgines web d'administració només es poden consultar en català. La traducció ha estat encarregada al Servei de Traducció de la universitat. Per fer-ho es fan servir constants de PHP.
- Replicació: la web ha estat modificada per poder ser replicada per altres departaments. La solució emprada és fer servir una carpeta diferent al servidor web i una base de dades diferent per cada departament . Dintre d'aquesta carpeta es troben dues carpetes, "app" i "config". La carpeta "app" conté el codi PHP i javascript propi de la web i es comuna a totes les repliques. La carpeta "config" conté un fitxer de configuració que és diferent a cada departament. Entre altres dades hi figura el nom del departament i dades de connexió a la base de dades.
  Quan cal fer un desplegament d'una nova versió de la web només cal copiar la carpeta "app" del lloc web de prova a cada carpeta de cada departament. Una tasca que es pot automatitzar.
- Allotjament: el projecte estarà allotjat a un servei d'allotjament web que ofereix la Universitat Autònoma de Barcelona.
- Fitxers: cada tipus de formulari consta de 6 fitxers:
  1. **nomFormulari.php:** genera una pàgina web on l'usuari bàsic pot veure un llistat de tots els formularis que ha omplert d'aquest tipus de formulari. Pot fer cerques, consultar formularis antics i omplir un nou formulari.
  2. **nomFormulari1.php:** formulari que farà servir l'usuari bàsic. Recull les dades i les incorpora a la base de dades. Això genera un avís visible pel usuaris administrador per que el validin. Un cop omplert i guardat el pot imprimir en PDF.
  3. **admin_nomFormulari.php:** genera una pàgina web on l'usuari administrador pot veure tot el llistat de formularis omplerts d'aquest tipus, tant històrics com nous. Pot fer cerques i consultar els formularis.

4. **nomFormulari2.php:** el formulari on l'usuari administrador veurà les dades que ha omplert l'usuari bàsic. Incorpora els camps exclusius dels usuaris administradors, en cas de ser necessaris. Si tot es correcte el pot validar. En tot moment el pot imprimir en PDF.
5. **nomFormulari_vista.php:** l'usuari pot veure però no modificar un formulari que ha omplert anteriorment. Aquí també veurà els camps exclusius per usuaris administrador si existeixen. Té l'opció d'imprimir-lo en PDF.
6. **nomFormulari_pdf.php:** genera un arxiu PDF amb les dades del formulari.

- Per fer les bateries de proves i avaluació de la qualitat primer es fa servir un departament fictici. Un cop provades les funcionalitats, s'incorpora a la web del Departament de Física, on tant els usuaris com el personal de la Gestió tenen més experiència en el projecte, per recollir els petits errors que trobin o els canvis que suggereixin un cop facin servir cada nova característica. Un cop estigui testada la nova versió es fa la còpia de la carpeta "app" a la resta de carpetes dels departaments.

L'aplicatiu de formularis es troba actualment en producció en vuit departaments amb un total de més de tres-cents usuaris que tenen a la seva disposició onze formularis diferents. L'any 2013 es van omplir més de vuit-cents formularis fent servir aquesta eina.

3. OBJECTIUS DEL TREBALL DE FINAL DE CARRERA

3.1 OBJECTIUS PRINCIPALS

Aquests son els principals objectius d'aquest treball:

- Traspassar totes les característiques que tenia fins ara el projecte en Programació Estructurada en PHP a Programació Orientada a l'Objecte també en PHP, mantenint les funcionalitats i l'aspecte de la web, però millorant el reaprofitament de codi i facilitant la tasca de fer nous formularis. Les característiques principals són:
    - Poder definir de forma senzilla cada tipus de dada del formulari amb totes les seves característiques.
    - Els tipus de dades han de poder ser data, text, select, text area, radio button o checkbox.
    - Cal poder fer de forma senzilla una inserció i una actualització de les dades a la base de dades.
    - Cal poder omplir un formulari amb dades recuperades d'una base de dades donat un identificador.
    - Cal que funcioni totes les funcionalitats fetes en llenguatge Javascript que hi ha ara.
- Fer servir Objectes de Dades de PHP (PDO) a l'hora de fer la connexió amb la base de dades. PDO fa servir les mateixes funcions per realitzar consultes independentment de la base de dades que es faci servir i a més funciona amb orientació a objectes. D'aquesta forma no només s'integra amb l'esperit del projecte de deixar de banda la Programació Estructurada si no que a més prepara el projecte davant del probable cas que la versió *community* de MySQL sigui discontinuada per part d'Oracle.
- Refer els formularis actuals aprofitant les classes que es desenvoluparan. Això ajudarà a confirmar que es pot mantenir l'aspecte i la funcionalitat reduint no només el temps de programació si no també la complexitat i l'espai que ocupa cada fitxer.
- Fer nous formularis amb les classes desenvolupades per avaluar la seva funcionalitat al començar de zero un formulari.

3.2 OBJECTIUS SECUNDARIS

A continuació els objectius secundaris d'aquest treball:

- Modularització: Implementar la possibilitat que cada departament pugui fer servir els formularis que triï. D'aquesta forma si un departament decideix no fer servir algun formulari els seus usuaris no el veuran.
- Flux controlat per rols: Implementar tota una nova part per que cada Investigador Principal (IP, cap responsable del projecte) pugui veure i consultar tots els formularis omplerts que tinguin relació amb algun dels projectes del que es responsable. D'aquesta forma podria tenir una visió gairebé complerta dels viatges, materials, conferencies, etc que han suposat un càrrec al seu/s projecte/s.

4. AVANTATGES DE LA PROGRAMACIÓ ORIENTADA A L'OBJECTE SOBRE LA PROGRAMACIÓ ESTRUCTURADA EN AQUEST TREBALL

Fent servir Programació Estructurada podem resoldre un problema de principi a fi fent servir una sola estructura de codi. És perfectament vàlida per resoldre petits problemes i el codi és fàcilment llegible i si es comenta bé resulta un tipus de codi més fàcil d'entendre.

Malgrat això l'estat actual del projecte convida a fer servir Programació Orientada a l'Objecte per aprofitar les següents característiques:

- Foment de la reutilització del codi:
  Tot i que fins ara ja s'han fet servir funcions i procediments el reaprofitament del codi serà clarament superior al fer servir els mètodes de cada objecte. Això ens permetrà ser més eficients i reduir el temps necessari per desenvolupar nous formularis. Serà una de les principals avantatges aportades pel canvi de paradigma.
- Aporta senzillesa a un sistema complex:
  A l'identificar als actors que participaran en el problema així com les seves accions s'ofereix al programador una visió més clara i neta de la complexitat del problema i permetrà estalviar temps d'adaptació en cas que més persones s'afegeixin al projecte a posteriori.
- Facilita la feina en equip:
  Al tenir unes classes i uns mètodes definits es fomenta que el codi sigui més homogeni malgrat hi treballin diferents persones. A més es definiran unes classes i uns mètodes senzills juntament amb una documentació amb abundants exemples de forma que qualsevol persona que comenci a treballar en aquest projecte passi el menys temps possible d'adaptació.
- Facilita el manteniment del codi:
  En general no caldrà modificar cada formulari quan hi hagi una modificació en algun mètode de les classes o quan es generin nous mètodes. D'aquesta forma el manteniment serà molt més ràpid i suposarà un gran estalvi de temps i feina. Això també implica que els fitxers a actualitzar entre diferents versions del projecte siguin menys que amb la Programació Estructurada: en general només caldrà actualitzar les classes i no tots els fitxers de formularis afectats pels canvis.
- Facilita les modificacions i el disseny en codi HTML:
  Com que al fer servir Programació Orientada a l'Objecte tindrem menys codi PHP entre el codi HTML serà més senzill modificar l'aspecte de formularis existents així com fer nous dissenys de formularis.

Com es pot veure al fer el canvi de paradigma no busquem un increment en la eficiència del codi del projecte ja que ara mateix el fitxers de formularis son prou petits i sense elements que alenteixin la càrrega que el marge de millora és mínim. Per una altre banda si busquem la eficiència en el temps de desenvolupar nous formularis, que aprofitin els mètodes que es crearan, així com les modificacions constants que s'han de fer en els actuals formularis. Busquem que la programació en aquest projecte sigui més ràpida, eficient i senzilla.

5. METODOLOGIA

La metodologia escollida per desenvolupar les classes i provar-les és la següent:

1. Es farà una primera versió de les classes necessàries tenint en compte que cal implementar totes les característiques detallades al punt 3.1 Objectius Principals.
2. S'aprofitarà un dels formularis actuals i es modificarà fent servir les classes desenvolupades i la nova forma de connectar amb la base de dades. Es recolliran dades de com es redueix el codi i la seva complexitat.
3. Es farà una segona versió de les classes, en cas que sigui necessari, per corregir o millorar amb l'experiència obtinguda al fer la modificació d'un formulari.
4. Es farà dos formularis de nou fent servir les classes desenvolupades. Es calcularà el cost comparat amb el sistema anterior de programació.
5. Si es necessari es farà una tercera versió de les classes aprofitant l'experiència de fer el punt anterior.

Un cop les classes estiguin acabades i quedi clar que compleixen amb la seva funció sense errors es refactoritzaran tots els formularis actuals, unificant la programació i els comentaris. Això serà molt útil per si el projecte es traspassa a unes altres persones o algú més entri a desenvolupar.

Id	Nombre de tarea	Duración
1	Primera versió de les classes	5 días
2	Prova de les classes modificant un formulari	2 días
3	Segona versió de les classes	1 día
4	Formularis nous	4 días
5	Tercera versió de les classes (si es necessària)	1 día

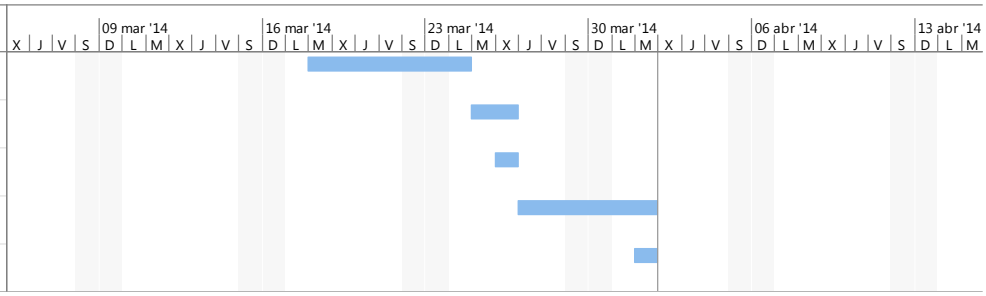


Figura 1

6. ARQUITECTURA DE L'APLICACIÓ

- **Client:**
  Al tractar-se d'una aplicació web l'únic que es necessita és un navegador web. L'aplicació s'està fent servir amb Mozilla Firefox, Google Chrome i Internet Explorer indistintament segons les preferències dels usuaris.
- **Servidor:**
  Per executar l'aplicació es necessita un servidor web amb PHP versió 5. En aquest cas l'aplicació està allotjada a un servei d'allotjament del Servei d'Informàtica de la Universitat Autònoma de Barcelona, que ofereix aquest servei mitjançant el programa Parallels Plesk Panel.
- **Base de dades:**
  L'aplicació fa servir el SGBD MySQL 5 també oferta pel mateix servei d'hostatge.

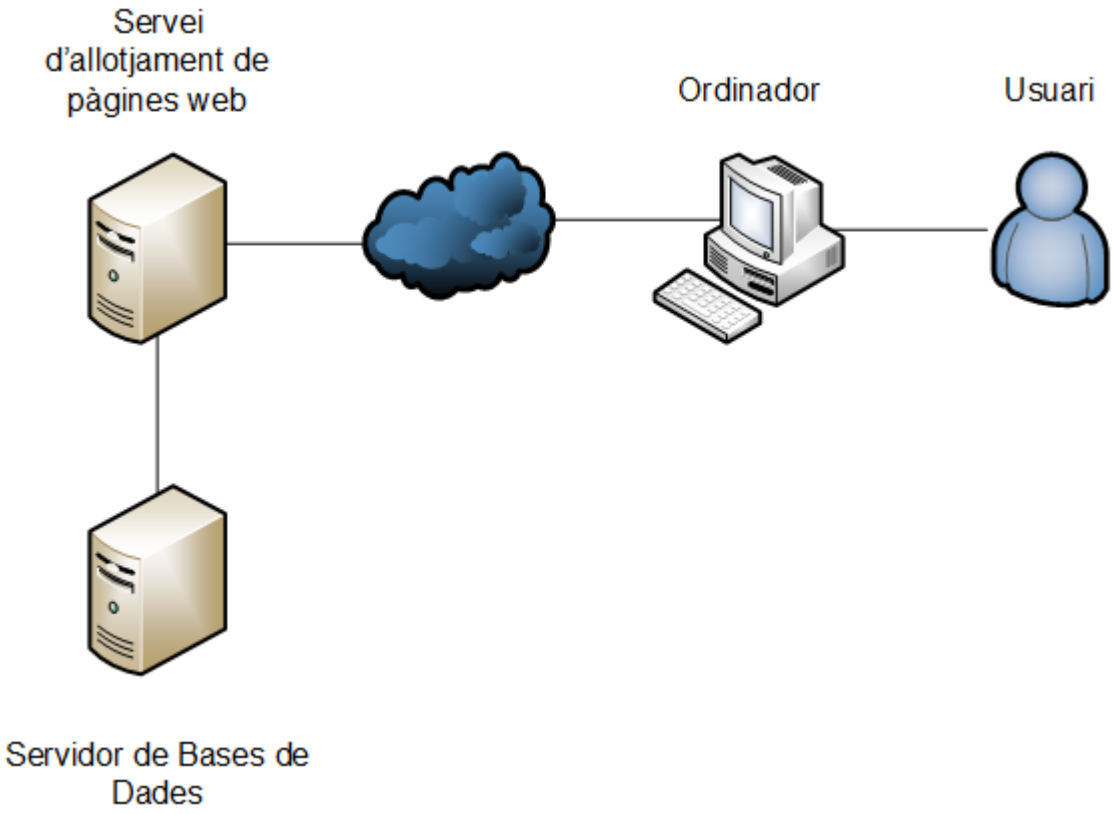


Figura 2

7. PLANIFICACIÓ

7.1 CALENDARI

Al següent calendari es pot veure la planificació inicial de projecte, tenint en compte:

- Considero dia laborable de dilluns a divendres.
- Cada dia laborable puc dedicar 3 hores al projecte.
- En cas de necessitat, imprevistos o altres puc dedicar més hores i treballar en cap de setmana.

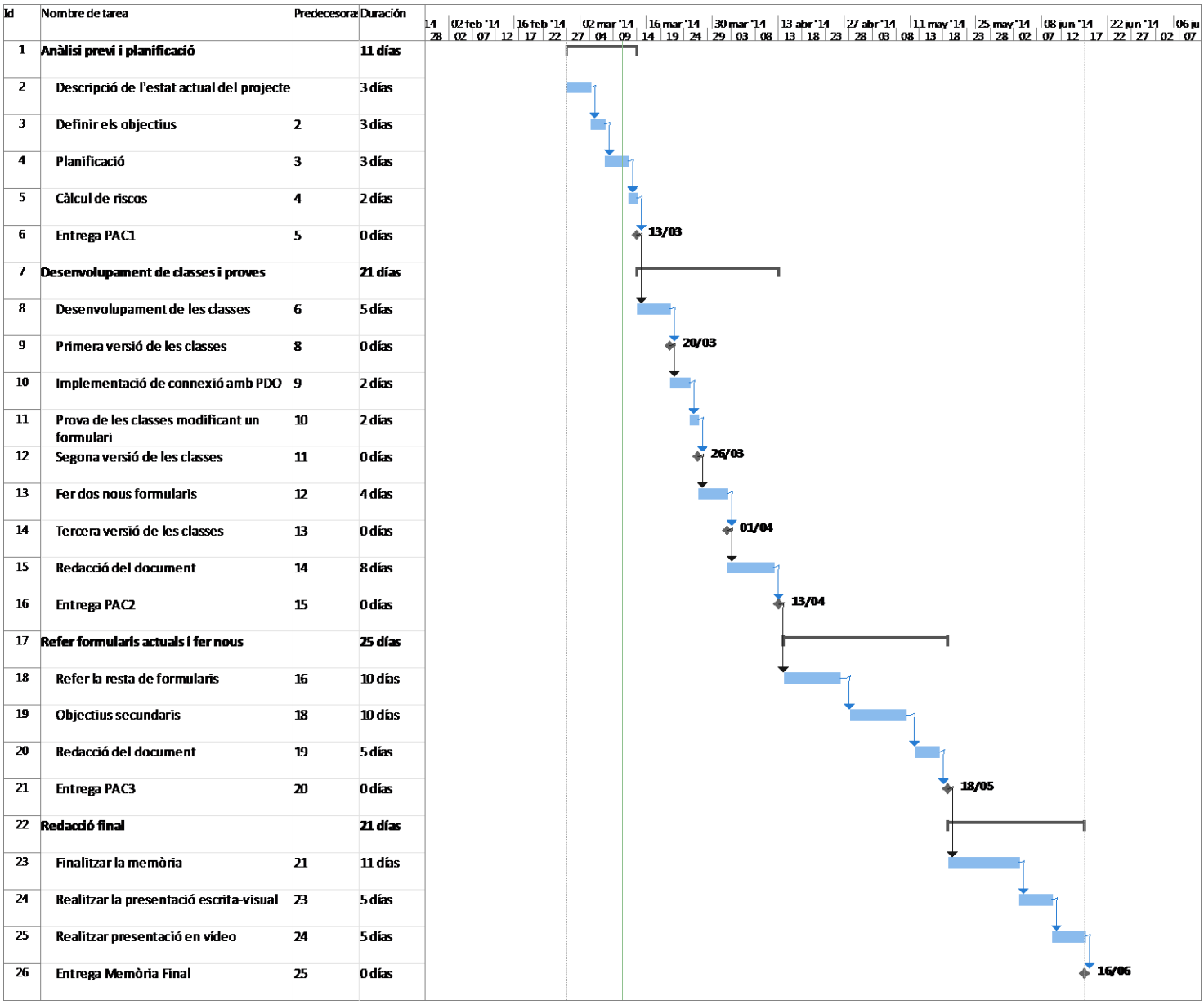
A la **figura 3** es pot veure el diagrama de Gantt corresponent.

Anàlisi previ i planificació	**11 dies**	**27/02/2014**	**13/03/2014**
Descripció de l'estat actual del projecte	3	27/02/2014	03/03/2014
Definir els objectius	3	04/03/2014	06/03/2014
Planificació	3	07/03/2014	11/03/2014
Càlcul de riscos	2	12/03/2014	13/03/2014
Entrega PAC1		**13/03/2014**	

Desenvolupament de classes i proves	**21 dies**	**14/03/2014**	**13/04/2014**
Desenvolupament de les classes	5	14/03/2014	20/03/2014
Implementació de connexió amb PDO	2	21/03/2014	24/03/2014
Prova de les classes modificant un formulari	2	25/03/2014	26/03/2014
Fer dos nous formularis	4	27/03/2014	01/04/2014
Redacció del document	8	02/04/2014	11/04/2014
Entrega PAC2		**13/04/2014**	

Refer formularis actuals i fer nous	**25 dies**	**14/04/2014**	**18/05/2014**
Refer la resta de formularis	10	14/04/2014	25/04/2014
Objectius secundaris	10	28/04/2014	09/05/2014
Redacció del document	5	12/05/2014	16/05/2014
Entrega PAC3		**18/05/2014**	

Redacció final	**21 dies**	**19/05/2014**	**16/06/2014**
Finalitzar la memòria	11	19/05/2014	02/06/2014
Realitzar presentació escrita-visual	5	03/06/2014	09/06/2014
Realitzar presentació en vídeo	5	10/06/2014	16/06/2014
Entrega Memòria Final		**16/06/2014**	



Id	Nombre de tarea	Predecesora	Duración
1	**Anàlisi previ i planificació**		11 días
2	Descripció de l'estat actual del projecte		3 días
3	Definir els objectius	2	3 días
4	Planificació	3	3 días
5	Càlcul de riscos	4	2 días
6	Entrega PAC1	5	0 días
7	**Desenvolupament de classes i proves**		21 días
8	Desenvolupament de les classes	6	5 días
9	Primera versió de les classes	8	0 días
10	Implementació de connexió amb PDO	9	2 días
11	Prova de les classes modificant un formulari	10	2 días
12	Segona versió de les classes	11	0 días
13	Fer dos nous formularis	12	4 días
14	Tercera versió de les classes	13	0 días
15	Redacció del document	14	8 días
16	Entrega PAC2	15	0 días
17	**Refer formularis actuals i fer nous**		25 días
18	Refer la resta de formularis	16	10 días
19	Objectius secundaris	18	10 días
20	Redacció del document	19	5 días
21	Entrega PAC3	20	0 días
22	**Redacció final**		21 días
23	Finalitzar la memòria	21	11 días
24	Realitzar la presentació escrita-visual	23	5 días
25	Realitzar presentació en vídeo	24	5 días
26	Entrega Memòria Final	25	0 días

Figura 3

7.2 FITES DEL PROJECTE

Data	Descripció
13/03/2014	Entrega de la PAC1
20/03/2014	Finalització de la primera versió de les classes
26/03/2014	Segona versió classes incorporant suggeriments al refer un formulari
01/04/2014	Tercera versió classes incorporant suggeriments al fer un formulari de nou
13/04/2014	Entrega de la PAC2
18/05/2014	Entrega de la PAC3
16/06/2014	Entrega de la memòria final i de les presentacions

7.3 ANÀLISI DE RISCOS

Risc	Naixement de la meva filla
Descripció	Estem esperant el naixement d'una nena per la setmana del 26/03/2014. Es possible que pugui dedicar menys hores al projecte durant les primeres setmanes
Impacte	Alt. Afectació a les dates d'acabament d'activitats
Probabilitat	Alta
Acció correctora	Considero que la dedicació exposada a la planificació ha estat conservadora. Intentaré avançar feina abans que arribi el naixement aprofitant més hores al dia i incloent caps de setmana (treballant remotament)

Risc	Malaltia
Descripció	Baixa temporal degut a malaltia
Impacte	Baix. Afectació a les dates d'acabament d'activitats
Probabilitat	Molt baixa
Acció correctora	Dedicar més hores un cop recuperat per complir les fites

Risc	Avaria de l'ordinador de desenvolupament
Descripció	Avaria de l'ordinador on desenvolupo el projecte
Impacte	Molt baix. Afectació a les dates d'acabament d'activitats
Probabilitat	Baixa
Acció correctora	Al tractar-se d'un ordinador estàndard al servei d'informàtica on treballo puc disposar d'un exactament igual minuts després de l'avaria. En el pitjor dels casos caldria fer una reinstal·lació del software.

8. DIAGRAMES UML

8.1 DIAGRAMA DE CLASSES

A la figura 4 es pot veure el diagrama de classes.

Tal i com s'ha dissenyat la classe *Formulari* és una composició d'objectes *Dada*. Cada component, una dada, constitueix una part de l'objecte *Formulari* i la seva vida coincideix amb la vida de l'objecte contenidor.

Aquest disseny permet treballar sobre cada dada concreta i crear-la amb de forma molt precisa, amb totes les seves característiques i alhora poder treballar amb totes les dades del formulari i per exemple guardar-les a la base de dades, controlar si totes les dades obligatòries estan emplenades, etc, i suposa un gran avantatge amb el model de Programació Estructurada ja que independentment de la quantitat de dades que tingui un formulari el cost de treballar amb ell sempre és el mateix enfront a per exemple construir sentències SQL específiques segons els nombre i el tipus de les dades.

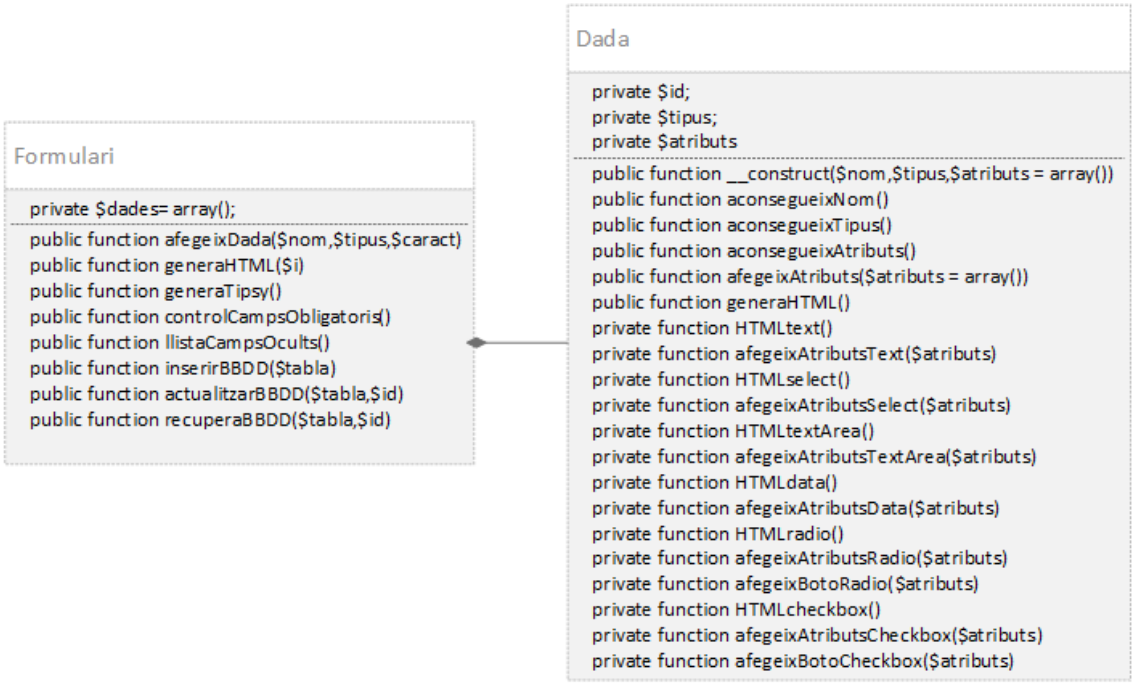


Figura 4

9. IMPLEMENTACIÓ

El primer pas per fer el canvi de Programació Estructurada a Programació Orientada a l'Objecte era identificar els objectes. Donat que necessitàvem mètodes per operar amb tot el formulari, com per exemple inserir les dades a la base de dades, recuperar-les, modificar-les, validar els camps, etc, vam decidir que el mateix formulari havia de ser un objecte.

El següent pas va ser veure que un formulari està compost per dades. Cada dada ha de ser un objecte en si mateix.

Cada dada d'un formulari s'interpreta en html com a un element de formulari. Cada element té un nom, un tipus i uns atributs específics per aquell tipus i no es pot generalitzar i per tant calia fer un objecte molt versàtil. Cada tipus d'element de formulari té atributs diferents i de vegades encara que tinguin el mateix nom s'han de tractar de forma diferent. Això va donar la idea de fer un array d'atributs. I cada atribut havia de tenir un valor. D'aquesta forma s'han pogut fer implementacions com per exemple:

El tipus *Data* té un atribut anomenat *value* però pot tenir valors diversos:

- 'value'=>'post': el valor de la dada ve donada per post.
- 'value'=>'avui': el valor de la dada és la data d'avui.

Això permet una gran flexibilitat: a cada necessitat la classe Dada s'ha anat adaptant sense gaire esforç.

Una característica bàsica és que aquest array d'atributs no té una mida tancada. Certs elements de formulari tenen més possibilitats que altres.

I com a conseqüència no buscada de fer servir aquest array en forma *atribut=>valor* i tractar-lo amb la funció *switch* de PHP ha sorgit la possibilitat de poder afegir html sense que estigui definit a la classe. A cada *switch* s'ha definit la opció per defecte (*default*) de forma que si es troba un atribut no llistat a la llista de casos del switch passa a la opció per defecte i s'ha fet que el resultat html sigui de la següent forma:

atribut="valor"

Per exemple, podem afegir atributs a un element de tipus *text* com per exemple:

```
"title"=>"Aquest camp s'ha d'omplir amb números"
```

que al entrar en l'opció "*default*" del *switch* generarà el següent codi html:

```
title="Aquest camp s'ha d'omplir amb números"
```

I el que és encara més interessant, podem afegir funcionalitat Javascript sense haver d'afegir programació en PHP per tractar els atributs.

A continuació es pot veure un exemple més complet de la definició d'una dada de tipus text:

```
$formulari->afegeixDada("import","text", array('size'=>9, 'maxlength'=>10,
'value'=>'post', "justificat"=>"dreta",
"onChange"=>"actTotalInscripcions('');", "onkeypress"=>"return
validaNaN(event);"));
```

I el resultat html que retorna la classe:

```
<input id="import" type="text" name="import1" size="9" maxlength="10"
onChange="actTotalInscripcions('');" onkeypress="return validaNaN(event);"
style=" text-align: right;" />
```

Cal puntualitzar que el nom del camp on s'emmagatzema la dada a la base de dades no es part de l'array d'atributs ja que per evitar equivocs fem servir el mateix nom per l'element de formulari que per el camp de la taula on emmagatzemava el formulari i per tant no cal especificar-ho.

10. API

Una API (*Application Programming Interface*) en la Programació Orientada a l'Objecte és el conjunt de mètodes que té a la seva disposició la persona que fa servir la classe. Ens proporciona una certa abstracció en la programació i facilita al programador/a la seva tasca.

És una part molt important d'aquest TFC ja que servirà de consulta per la gent que ja fa servir les classes i permetrà que les persones que les facin servir a partir d'ara en facin un ús més eficient.

10.1 TIPUS DE DADES

- **Text:** L'objecte dada generarà un element de formulari html de tipus INPUT. Es pot deixar buit o prefixar el contingut amb el nom de l'usuari actual o qualsevol altre valor.
- **Data:** L'objecte dada generarà un element de formulari html de tipus INPUT però per contenir específicament una data. Pot ser una data prefixada o incorporar funcionalitat Javascript per seleccionar-la mitjançant un calendari.
- **Textarea:** L'objecte generarà un element de formulari html de tipus TEXTAREA.
- **Select:** L'objecte generarà un element de formulari html de tipus SELECT. Cal fer un array amb les opcions que volem que apareguin al desplegable en la forma `$arrayExemple= array ('valor1'=>'etiqueta1','valorN'=>'etiquetaN');` Aquest array s'ha de fer servir com el valor de l'atribut 'option'.
- **Radio:** L'objecte generarà un element de formulari html de tipus RADIO BUTTON. Cal fer un array amb les opcions que volem en la forma $arrayExemple= array ('valor1','valor2','valorN');
  Aquest array s'ha de fer servir com el valor de l'atribut 'option'. Cal afegir una dada per cada opció amb el nom de cada opció. D'aquesta forma podrem generar l'html de cada RADIO BUTTON per separat i col·locar el codi HTML allà on ens calgui.
- **Checkbox:** La dada ha de generar un element de formulari html de tipus CHECKBOX.

10.2 CLASSE FORMULARI

Representació abstracta d'un formulari. Està format per un conjunt de dades.

10.2.1 Variables de la classe

Nom	Privacitat	Descripció
dades	private	Array de dades

10.2.2 Constructor

`formulari();`
Crea una nova instància Formulari.

10.2.3 Mètodes

afegeixDada	
`public function afegeixDada (string $nombre, string $tipus, array $atributs)`	
Descripció: El mètode més important del projecte. Afegeix una dada al formulari. Hi han 6 tipus diferents: text, data, textarea, select, radio i checkbox. Cada tipus té atributs diferents que li ofereixen versatilitat.	
Tipus: TEXT	
Exemple:	
`$formulari->afegeixDada("nom", "text", array( 'size'=>50, 'maxlength'=>100, 'value'=>'post', 'obligatori'=>'obligatori'));`	
Atributs:	
'value'=>'post'	El valor de la dada es passa per sistema POST.
'value'=>'usuari'	El valor de la dada ha de ser el nom de l'usuari actual.
'value'=>'un valor'	Opció per defecte. Es pot fixar quin valor es vol posar a la dada.
'obligatori'=>'obligatori'	La dada ha de ser omplerta obligatòriament. Si no s'omple es marca el camp en vermell i impedeix que el

	formulari es guardi a la base de dades.
'obligatori'=>'projecte' **'obligatori'=>'centreCost'**	Casos especials: en aquest cas és obligatori que UN dels dos camps, projecte o centre de cost, estiguin omplerts.
'obligatori'=>'nom d'una dada'	Condicional: la dada és obligatòria si la dada introduïda com a paràmetre està plena. Ex: l'import és obligatori si em omplert el camp *descripció* a una comanda.
'importObligatori'=>''	Cas només per camp numèric. Indica que és un número obligatori. Afegeix els paràmetres per cridar a funcions javascript que asseguren que no es posa cap lletra i que el separador dels decimals és un punt (.).
'readonly'=>''	Aquesta dada ha de generar un html on no es pugui editar el camp.
'justificat'='dreta'	Al generar l'html aquesta dada s'ha de mostrar justificada a la dreta. Normalment es fa servir per camps numèrics.
'gris'=>''	Fa el fons del camp de text html de color gris. Acostuma a estar acompanyada de la opció 'readonly'
'etiqueta html'=>'valor per l'etiqueta'	Opció genèrica. Serveix per fixar paràmetres html. Per exemple 'title'=>'hola' farà que al generar l'html aquest tingui un camp TITLE amb valor 'hola'. També es fa servir per fixar els paràmetres html SIZE i MAXLENGHT o per afegir funcionalitat JAVASCRIPT.
Tipus: DATA	
Exemple:	
`$formulari->afegeixDada("data", "data", array('value'=>'avui', 'readonly'=>'', "gris"=>''));`	
Atributs:	
'calendari'=>''	Cal afegir la crida a javascript per fer servir un calendari per triar la data.
'value'=>'post'	El valor de la dada es passa per sistema POST.
'value'=>'avui'	El valor de la dada ha de ser la data d'avui.
'value'=>'una data'	Fixa el valor de la dada a un valor introduït.
'obligatori'=>'obligatori'	La dada ha de ser omplerta obligatòriament. Si no s'omple es marca el camp en vermell i impedeix que el formulari es guardi a la base de dades.
'readonly'=>''	Aquesta dada ha de generar un html on no es pugui editar el camp.
'gris'=>''	Fa el fons del camp de text html de color gris. Acostuma a estar acompanyada de la opció 'readonly'.

Tipus: TEXTAREA

Exemple:

```
$formulari->afegeixDada("observacions", "textArea", array('cols'=>'',
'rows'=>'5', 'ample'=>'', 'value'=>'post'));
```

Atributs:

'ample=>''	La dada ha de generar un element TEXTAREA html de tota l'amplada del formulari.
'value'=>'post'	El valor de la dada es passa per sistema POST.
'value'=>'un valor'	Fixa el valor de la dada a un valor introduït.
'readonly'=>''	Aquesta dada ha de generar un html on no es pugui editar el camp.
'gris'=>''	Fa el fons del camp de text html de color gris. Acostuma a estar acompanyada de la opció 'readonly'
'etiqueta html'=>'valor per l'etiqueta'	Opció genèrica. Serveix per fixar paràmetres html. Per exemple 'title'=>'hola' farà que al generar l'html aquest tingui un camp TITLE amb valor 'hola'. També es fa servir per fixar els paràmetres html COLS i ROWS o per afegir funcionalitat JAVASCRIPT.

Tipus: SELECT

Exemple:

```
$arrayTarifa=array('0.19'=>'0.19€','0.30'=>'0.30€');

$formulari->afegeixDada("tarifa",   "select",
array('option'=>$arrayTarifa,'value'=>'post'));
```

Atributs:

'option'=>'nom de l'array d'opcions'	Defineix les opcions de l'element SELECT. Cal que sigui un array en la forma `$arrayExemple= array ('valor1'=>'etiqueta1', 'valorN'=>'etiquetaN');`
'value'=>'post'	El valor de la dada es passa per sistema POST.
'value'=>'un valor'	Fixa el valor de la dada a un valor introduït.
'readonly'=>''	Aquesta dada ha de generar un html on no es pugui editar el camp.
'gris'=>''	Fa el fons del camp de text html de color gris. Acostuma a estar acompanyada de la opció 'readonly'
'etiqueta html'=>'valor per l'etiqueta'	Opció genèrica. Serveix per fixar paràmetres html. Per exemple 'title'=>'hola' farà que al generar l'html aquest tingui un camp TITLE amb valor 'hola' o per afegir funcionalitat JAVASCRIPT.

Tipus: RADIO

Exemple:

```
$arrayRadio=array('becari','uab','alie','vinculat');
$formulari->afegeixDada("becari", "radio", array('option'=>$arrayRadio,
'value'=>'post',  'obligatori'=>'obligatori'));
$formulari->afegeixDada("uab", "radio", array('option'=>$arrayRadio,
'value'=>'post',  'obligatori'=>'obligatori'));
$formulari->afegeixDada("alie", "radio", array('option'=>$arrayRadio,
'value'=>'post','title'=>Marcar_Quatre, 'obligatori'=>'obligatori'));
$formulari->afegeixDada("vinculat", "radio", array('option'=>$arrayRadio,
'value'=>'post','title'=>Marcar_Quatre, 'obligatori'=>'obligatori'));
```

Atributs:

'option'=>'nom de l'array d'opcions'	Defineix les opcions de l'element RADIO BUTTON. Cal que sigui un array en la forma `$arrayExemple= array ('valor1','valor2','valorN');`
'value'=>'post'	El valor de la dada es passa per sistema POST.
'value'=>'un valor'	Fixa el valor de la dada a un valor introduït.
'readonly'=>''	Aquesta dada ha de generar un html on no es pugui editar el camp.
'gris'=>''	Fa el fons del camp de text html de color gris. Acostuma a estar acompanyada de la opció 'readonly'
'etiqueta html'=>'valor per l'etiqueta'	Opció genèrica. Serveix per fixar paràmetres html. Per exemple 'title'=>'hola' farà que al generar l'html aquest tingui un camp TITLE amb valor 'hola' o per afegir funcionalitat JAVASCRIPT.

Tipus: CHECKBOX

Exemple:

```
$formulari->afegeixDada("proposta_feta","checkbox",
array('value'=>'post','modificable'=>''));
```

Atributs:

'value'=>'post'	El valor de la dada es passa per sistema POST.
'value'=>'un valor'	Fixa el valor de la dada a un valor introduït.
'modificable'=>''	Marca si el CHECKBOX es podrà prémer o no.
'etiqueta html'=>'valor per l'etiqueta'	Opció genèrica. Serveix per fixar paràmetres html. Per exemple 'title'=>'hola' farà que al generar l'html aquest tingui un camp TITLE amb valor 'hola' o per afegir funcionalitat JAVASCRIPT.

generaHTML
`public function generaHTML(string $nom)`
Descripció: Genera l'html de la dada que volem. Cal passar com a paràmetre el nom de la dada.
Exemple: `$formulari->generaHTML("titular_compte");`

generaTipsy
`public function generaTipsy()`
Descripció: Genera el codi javascript per fer que la llibreria Tipsy doni format a les etiquetes html TITLE. Vegeu Annex 3 per més informació.
Exemple: `$formulari->generaTipsy();`

controlCampsObligatoris
`public function controlCampsObligatoris()`
Descripció: Comprova si el camps marcats com obligatoris estan emplenats.
Retorna: 1 si tots el camps son omplerts 0 en cas contrari.
Exemple: `$control=$formulari->controlCampsObligatoris();`

llistaCampsOcults
`public function llistaCampsOcults()`
Descripció: Genera el codi html amb totes les dades passades per POST. Genera un INPUT ocult per cada dada amb el seu valor. Es fa servir per passar les dades al generador de PDF's ja que necessita que li passin per POST.
Exemple: `$formulari->llistaCampsOcults();`

inserirBBDD
`public function inserirBBDD(string $taula)`
Descripció: Genera i executa l'INSERT en SQL per tal que totes les dades es guardin a la BBDD. Cal determinar com a paràmetre el nom de la taula.
Retorna: Identificador del registre inserit a la BBDD si la inserció ha estat . Útil en cas que calgui enviar un correu electrònic automàtic a l'usuari.
Exemple: `$idFormulari=$formulari->inserirBBDD($taula);`

actualitzarBBDD
`public function actualitzarBBDD(string $taula,int $id)`
Descripció: Genera i executa l'UPDATE en SQL per tal que totes les dades s'actulitzin a la BBDD. Cal determinar com a paràmetre el nom de la taula i l'identificador del formulari.
Retorna: Identificador del registre actualitzat a la BBDD. Útil en cas que calgui enviar un correu electrònic automàtic a l'usuari.
Exemple: `$idFormulari=$formulari->inserirBBDD($taula);`

recuperaBBDD
`public function recuperaBBDD(string  $taula,int $id)`
Descripció: Recupera les dades d'un formulari concret i assigna un atribut VALUE a cada objecte de l'array de dades.
Exemple: `$formulari->recuperaBBDD($taula,$id);`

10.3 CLASSE DADA

Representació abstracta d'una dada d'un formulari.

10.3.1 Variables de la classe

Nom	Privacitat	Descripció
id	private	Nom de la dada
tipus	private	Tipus de dada
atributs	private	Array d'atributs

10.3.2 Constructor

```
$dada=new dada(string id, string tipus,array atributs);
```

Crea una nova instància Dada.

10.3.3 Mètodes

aconsegueixNom
public function aconsegueixNom()
Descripció: Retorna el nom de l'objecte.
Retorna: Variable de la classe id (string).
Exemple: `$this->aconsegueixNom();`

aconsegueixTipus
`public function aconsegueixTipus()`
Descripció: Retorna el tipus de l'objecte.
Retorna: Variable de la classe tipus (string).
Exemple: `$this->aconsegueixTipus();`

aconsegueixAtributs
`public function aconsegueixAtributs()`
Descripció: Retorna l'array d'atributs de l'objecte.
Retorna: Variable de la classe atributs (array).
Exemple: `$this->aconsegueixAtributs();`

afegeixAtributs
`public function afegeixAtributs($atributs = array())`
Descripció: Afegeix un array d'atributs a l'objecte.
Exemple: `$dada->afegeixAtributs($atributs);`

generaHTML
`public function generaHTML()`
Descripció: Genera l'html de l'objecte actual segons el tipus de dada que sigui.
Retorna: Codi html generat (string).
Exemple: `$dada->generaHTML();`

HTMLtext
`private function HTMLtext()`
Descripció: Genera l'html d'una dada de tipus TEXT.
Retorna: Codi html generat (string).
Exemple: `$str=$this->HTMLtext();`

afegeixAtributsText
`private function afegeixAtributsText(array $atributs)`
Descripció: Afegeix codi html segons la llista d'atributs de l'objecte tipus TEXT.
Retorna: Codi html generat (string).
Exemple: `$str.=$this->afegeixAtributsText($this->atributs);`

HTMLselect
`private function HTMLselect()`
Descripció: Genera l'html d'una dada de tipus SELECT.
Retorna: Codi html generat (string).
Exemple: `$str=$this->HTMLselect();`

afegeixAtributsSelect
`private function afegeixAtributsSelect(array $atributs)`
Descripció: Afegeix codi html segons la llista d'atributs de l'objecte tipus SELECT.
Retorna: Codi html generat (string).
Exemple: `$str.=$this->afegeixAtributsSelect($this->atributs);`

HTMLtextArea
`private function HTMLtextArea()`
Descripció: Genera l'html d'una dada de tipus TEXTAREA.
Retorna: Codi html generat (string).
Exemple: `$str=$this->HTMLtextArea();`

afegeixAtributsTextArea
`private function afegeixAtributsTextArea(array $atributs)`
Descripció: Afegeix codi html segons la llista d'atributs de l'objecte tipus TEXTAREA.
Retorna: Codi html generat (string).
Exemple: `$str .= $this->afegeixAtributsTextArea($this->atributs);`

HTMLdata
`private function HTMLdata()`
Descripció: Genera l'html d'una dada de tipus DATA.
Retorna: Codi html generat (string).
Exemple: `$str=$this->HTMLdata();`

afegeixAtributsData
`private function afegeixAtributsData(array $atributs)`
Descripció: Afegeix codi html segons la llista d'atributs de l'objecte tipus DATA.
Retorna: Codi html generat (string).
Exemple: `$str.=$this->afegeixAtributsData($this->atributs);`

HTMLradio
`private function HTMLradio()`
Descripció: Genera l'html d'una dada de tipus RADIO.
Retorna: Codi html generat (string).
Exemple: `$str=$this->HTMLradio();`

afegeixAtributsRadio
`private function afegeixAtributsRadio(array $atributs)`
Descripció: Afegeix codi html segons la llista d'atributs de l'objecte tipus RADIO.
Retorna: Codi html generat (string).
Exemple: `$str.=$this->afegeixAtributsRadio($this->atributs);`

afegeixBotoRadio
`private function afegeixBotoRadio(array $atributs)`
Descripció: Afegeix el botó de l'objecte RADIO en codi html.
Retorna: Codi html generat (string).
Exemple: `$str .= $this->afegeixBotoRadio($this->atributs);`

HTMLcheckbox
`private function HTMLcheckbox()`
Descripció: Genera l'html d'una dada de tipus CHECKBOX.
Retorna: Codi html generat (string).
Exemple: `$str=$this->HTMLcheckbox();`

afegeixAtributsCheckbox
`private function afegeixAtributsCheckbox($atributs)`
Descripció: Afegeix codi html segons la llista d'atributs de l'objecte tipus CHECKBOX.
Retorna: Codi html generat (string).
Exemple: `$str.=$this->afegeixAtributsCheckbox($this->atributs);`

afegeixBotoCheckbox
`private function afegeixBotoCheckbox(array $atributs)`
Descripció: Afegeix el botó de l'objecte CHECKBOX en codi html.
Retorna: Codi html generat (string).
Exemple: `$str.=$this->afegeixBotoCheckbox($this->atributs);`

11. REFACTORITZACIÓ DE FORMULARIS

11.1 REFACTORITZAR UN FORMULARI DE RECOLECCIÓ DE DADES AMB LES CLASSES

Per comprovar si les classes desenvolupades al TFC funcionaven correctament es va decidir desenvolupar-les alhora que refeia un apartat que ja estava en producció. D'aquesta forma s'ha pogut veure si el resultat de l'aplicació dels mètodes corresponia amb la funcionalitat que es tenia fent servir Programació Estructurada ja que era un dels principals objectius d'aquest projecte

Aprofitant que la normativa bancària europea ha canviat i que ara en comptes de número de compte corrent cal l'IBAN s'ha triat per provar a refer els formularis que fan servir els usuaris bàsics de l'apartat "*Inscripció a congressos, seminaris, etc a l'estranger*" que ara passarà a anomenar-se simplement "*Inscripció a congressos, seminaris, etc*".

Aquest canvi en l'estructura del formulari implicava certs canvis menors a la base de dades ja que calia eliminar certs camps i crear-ne de nous per les noves dades i també implicava un canvi més important: fer servir els mètodes que s'han definit a les classes Formulari i Dada.

A continuació expliquem els principals canvis que ha suposat passar de Programació Estructurada a Programació Orientada a l'Objecte:

1. **Definició de les dades del formulari:**
   Caldrà definir tots els camps del formulari amb les seves característiques. S'ha decidit fer-ho al principi del codi, just després de declarar l'ús de les classes.

   Per fer la declaració d'un camp només cal posar el nom, el tipus de dada i un array amb les seves característiques especials.

   Aquí es pot veure un petit exemple:

```
$formulari=new formulari();

$formulari->afegeixDada("data", "data", array('value'=>'avui',
'readonly'=>'',  "gris"=>''));
$formulari->afegeixDada("termini", "data", array('value'=>'post',
'obligatori'=>'', 'calendari'=>''));
$formulari->afegeixDada("interessat", "text", array('size'=>50,
'maxlength'=>100, 'value'=>'usuari', 'obligatori'=>'obligatori'));
$formulari->afegeixDada("titular_compte", "text", array('size'=>50,
'maxlength'=>100, 'value'=>'post', 'obligatori'=>'obligatori'));
$formulari->afegeixDada("bank", "text", array('size'=>50,
'maxlength'=>50, 'value'=>'post'));
$formulari->afegeixDada("swift_bic", "text", array('size'=>15,
'maxlength'=>15, 'value'=>'post'));
```

```
$formulari->afegeixDada("iban", "text", array('size'=>37,
'maxlength'=>40, 'value'=>'post'));
```

Amb les classes desenvolupades al TFC declarar les dades s'ha tornat la feina més feixuga però com es pot veure es bastant intuïtiu.

Per exemple, el que anteriorment, el codi que en Programació Estructurada calia per fer el camp "*data*" era:

```
<input name="data" type="text" id="data" size="9" readonly="readonly"
style="background-color:#E6E6E6" title="<?php echo
Camp_No_Editable;?>" value="<?php echo date('d-m-Y');?>"/>
```

Ara, un cop declarada la dada fent servir POO ha esdevingut:

```
<?php $formulari->generaHTML("data");?>
```

Per exemple, a la declaració de la dada "*data*" es pot veure que especifiquem que es tracta d'un camp per escriure una data, que el valor per defecte és el de la data d'avui, que no es pot modificar i que el fons de la casella el volem en color gris. Abans per fer-ho calia fer servir html i funcions PHP específiques. Ara ha esdevingut una feina molt més senzilla

2. **Llibreria Tipsy:**
   A cada formulari fem servir la llibreria Tipsy per donar format al camp html "*title*" de forma que aparegui destacat quan es passa el cursor per sobre de l'element de formulari, oferint més informació a l'usuari.

   Amb Programació Estructurada era de la següent forma:

```
<script type='text/javascript' src='js/jquery-1.7.2.min.js'></script>
<script type='text/javascript'
src='js/tipsy/javascripts/jquery.tipsy.js'></script>
<script type='text/javascript'>
    $(function() {
      $('#data').tipsy({gravity: 'w',fade:true});
      $('#termini').tipsy({gravity: 'w',fade:true});
      $('#account_holder').tipsy({gravity: 'w',fade:true});
      $('#descripcio1').tipsy({gravity: 'w',fade:true});
      $('#import1').tipsy({gravity: 'w',fade:true});
      $('#projecte').tipsy({gravity: 'w',fade:true});
      $('#centreCost').tipsy({gravity: 'w',fade:true});
    });
</script>
<link rel="stylesheet" href="js/tipsy/stylesheets/tipsy.css"
type="text/css" />
```

   A la classe Formulari s'ha desenvolupat un mètode per estalviar tot aquest codi de forma que ara només cal cridar-lo per aconseguir el mateix resultat:

```
<?php $formulari->generaTipsy();?>
```

3. Control de camps obligatoris:

El control de camps obligatoris era també fins ara una feina feixuga. Calia comprovar si cada camp obligatori estava omplert un cop enviat el formulari. Ara només cal fixar la característica "*obligatori*" amb el paràmetre "*obligatori*". El nou mètode *controlCampsObligatoris()* retorna cert o fals segons totes les dades obligatòries del formularis estan emplenades.

A més es dona el cas molt concret que no es dona en general a un formulari: donats dos camps és obligatori omplir només un del dos. Es poden omplir tots dos però si un està omplert, l'altre camp, encara que estigui buit no es obligatori.

Per exemple, el camp "*Centre de Cost*" i el camp "*Projecte*". Amb un dels dos n'hi ha prou per identificar de quin projecte sortiran els fons per pagar el servei o la comanda per tant si l'usuari ha omplert un l'altre deixa de ser camp obligatori.

Tal i com s'ha desenvolupat només cal passar com a paràmetre de l'atribut "obligatori" el nom de l'altre camp.

Per exemple, al definir la dada "*Centre de Cost*" passarem l'atribut "obligatori" de la següent forma:

```
"obligatori"=>"projecte"
```

El codi mirarà el valor del camp "*Projecte*" i si està buit farà el camp "*Centre de Cost*". En cas contrari no ho serà.

4. Recull de les dades amb mètode POST:

Si l'usuari omple el formulari però es deixa d'informar un camp obligatori el formulari s'ha de recarregar, indicar l'error i mantenir les dades correctes a cada element de formualri, de forma que l'usuari no hagi de tornar a omplir tot el formulari. Fins ara per fer-ho calia recuperar la dada a cada camp "*value*" de l'element de formulari. Ara només cal indicar que les dades arribaran per mètode POST com una de les característiques de l'objecte. Per exemple:

```
$formulari->afegeixDada("bank","text", array('size'=>50,
'maxlength'=>50, 'value'=>'post'));
```

5. Inserció de les dades a la base de dades:

Amb Programació Estructurada calia fer una sentència específica de tipus INSERT en SQL amb totes les dades recollides del formulari.

Calia tenir en compte el tipus de cada dada per definir-la entre cometes o no, el risc de SQL Injection i calia tractar els errors possibles durant la inserció. Un cop feta la inserció a la base de dades calia fer una sentència INSERT a la taula de log per anotar qui ha introduït el formulari i a quina hora.

Donat que hi ha formularis amb desenes de camps feia que la tasca d'escriure la sentència fos molt lenta i amb poc reaprofitament de codi.

La classe Formulari té el mètode *inserirBBDD($taula)* de forma que la sentència d'inserció a la base de dades es genera de forma automàtica.

```
$idFormulari=$formulari->inserirBBDD($taula);
```

Aquest mètode retorna l'identificador del registre generat al fer la inserció. Aquesta dada juntament amb altres la guardem en una taula de log on es guarden tots el canvis i l'usuari que els ha fet.

La única condició per un funcionament correcte és que el camp de formulari i el camp a la taula de la base de dades han de tenir el mateix nom. Això no ha suposat cap canvi ja que era una política de noms que ja feia servir abans.

11.2 REFACTORITZAR UN FORMULARI DE MODIFICACIÓ DE DADES AMB LES CLASSES

La principal diferència entre un formulari buit que recull dades i un que permet modificar-les és que en aquest darrer ha de contenir les dades que es troben emmagatzemades a la base de dades.

1. **Recuperar les dades de la base de dades:**

   Per fer-ho es passa un identificador que fa referència a quin formulari es vol modificar. Amb aquest identificador es consulta la base de dades, es recuperen les dades i s'omple el formulari. Fins ara calia dissenyar una consulta SQL per recuperar les dades i després posar cada dada al camp "*value*" de cada element de formulari.

   Ara, només cal fer servir el següent mètode:

   ```
   $formulari->recuperaBBDD($taula,$id);
   ```

   Al principi del codi ja s'han definit tots els objectes que formen part del formulari tal i com es feia en el formulari anterior però sense especificar cal valor al camp "*value*".

   El mètode *recuperaBBDD($taula,$id)* s'encarrega d'omplir cada element de formulari amb el seu valor corresponent a la base de dades.

2. **Modificació de les dades a la base de dades:**

   Anàlogament a com es feia la inserció de dades en el cas anterior ara per fer la modificació ja no cal escriure la sentència SQL UPDATE si no que no més cal executar el mètode corresponent:

   ```
   $idFormulari=$formulari->actualtizarBBDD($taula,$id);
   ```

11.3 FER NOUS FORMULARIS FENT SERVIR LES CLASSES

La tasca de fer nous formularis esdevé ara una feina molt menys feixuga. El temps de programació s'ha reduït molt clarament i la complexitat d'haver de tenir en compte molts paràmetres a cada tipus de dades ha esdevingut més senzilla.

Aquesta prova també ha servit per fer noves funcionalitats a les classes. Un cop implementades ja es pot fer qualsevol element de formulari dels següents tipus:

- Text
- Data (una variant de text específica per dates)
- TextArea
- RadioButton
- Select
- Checkbox

Ara en un dia de feina es pot fer tot un apartat (tres formularis, un d'inserció, un de modificació i un només de visualització) quan abans era feina per tres o quatre dies.

11.4 PRIMERES CONCLUSIONS DESPRÉS DE FER LES REFACTORITZACIONS

L'estalvi de temps i esforç és clarament considerable. En menys de dues hores s'ha aconseguit modificar un codi de més de 300 línies mantenint la funcionalitat però fent del procés de modificació una feina molt més senzilla. Si en el futur cal afegir una dada més suposarà més temps el canvi de disseny CSS i la creació del camp a la taula de la base de dades que el canvi en la programació. Només caldrà afegir una línia fent la definició de la dada i una crida al mètode *generaHTML($dada)*.

A la feina de fer un formulari de nou l'estalvi de temps és encara més apreciable. La feina de la programació ha esdevingut secundaria. Ara ocupa més temps el disseny del formulari i de la base de dades.

Ha estat una sorpresa que fent servir POO la mida dels fitxers no s'ha reduït de forma considerable. A priori esperàvem una reducció al fer servir les classes però veiem que l'estalvi en espai o no es produeix o és molt petit. La primera conclusió és que s'escriu una quantitat de codi però és molt més fàcil fer-ho. Sobretot per que hi ha molta reutilització de codi.

12. EXEMPLE DE FORMULARI

Figura 5

A continuació passem a detallar el codi que genera alguns dels camps que es poden veure a la figura 5:

- **Data:**
  Com es pot veure és un camp de text que conté la data d'avui de forma automàtica i amb el fons gris que indica que és un camp no editable. Com es pot veure a la captura al passar el cursor per sobre ens avisa. Amb la llibreria Tipsy li podem donar aquest aspecte al globus d'informació. Per generar aquest camp s'ha de fer servir el següent codi:

```
$formulari->afegeixDada("data", "data", array('value'=>'avui',
'readonly'=>'',  "gris"=>''));
```

- **Data de la conferència:**
  Es tracta també d'un camp de text però en aquest cas volem seleccionar una data. Es pot veure la icona a la dreta del camp on fer click per obrir un calendari. El codi per generar el camp:

```
$formulari->afegeixDada("data_conferencia", "data",
array('value'=>'post', 'obligatori'=>'', 'calendari'=>'',
'readonly'=>''));
```

  Es poden apreciar les diferències amb el codi anterior. El camp és igualment no editable directament però li afegim la possibilitat de seleccionar la data mitjançant un calendari desplegable. La dada es recollirà per POST i es tracta d'una dada obligatòria.

- **Import abans de descomptar IRPF:**
  Es tracta d'un camp de text amb la peculiaritat de que ha de contenir un número. L'usuari pot escriure qualsevol número amb els decimals separats per una coma (necessari per efectuar els càlculs). El codi per generar-ho és el següent:

```
$formulari->afegeixDada("import_abans_irpf", "text",
array('size'=>10, 'maxlength'=>10, 'value'=>'post',
'importObligatori'=>'', "onChange"=>"actHonorarisIRPF();",
"justificat"=>"dreta", "onkeypress"=>"return validaNaN(event);"));
```

  Com es pot veure fixem la mida del camp i el màxim de caràcters que es poden escriure. Al tractar-se d'un camp numèric el justifiquem a la dreta per estètica i li afegim funcionalitat Javascript de forma que al introduir una quantitat els dos camps que hi ha immediatament a sota s'actualitzaran de forma automàtica.

- **IRPF:**
  És un camp de selecció. Podem triar entre diverses opcions. Ho podem generar amb el següent codi:

```
$arrayTipusIrpf=array('nacionals'=>Honoraris_Nacionals,'estrangers'=>
Honoraris_Estrangers,'estrangers_conveni'=>Honoraris_Estrangers_Conve
ni);

$formulari->afegeixDada("tipus_irpf", "select",
array('option'=>$arrayTipusIrpf, "onChange"=>"actHonorarisIRPF();",
'value'=>'post'));
```

  Fixem-nos que s'han de definir els elements a seleccionar en un array a priori. També li afegim funcionalitat Javascript al igual que la dada anterior per actualitzar diferents camps segons la selecció que fem.

13. REQUISITS D'INSTAL·LACIÓ

13.1 SOFTWARE

Instal·lar aquest projecte és una tasca senzilla. A nivell de software els únics requeriments que necessitarem és tenir instal·lat un servidor web Apache amb PHP amb una versió superior a la 5 i el SGBD MySQL en qualsevol versió superior a 4.

13.2 HARDWARE

L'únic requeriment hardware que necessitem és un servidor estàndard. L'aplicació no consumeix gaires recursos ni de càlcul ni de xarxa ni d'espai al disc dur i per tant no cal fer una gran inversió en maquinari.

13.3 FORMACIÓ/CONEIXEMENTS

La persona o persones encarregades d'instal·lar l'aplicació cal que tinguin coneixements de:

1. S.O. del servidor per tal d'instal·lar els requeriments de software, copiar fitxers i gestionar els permisos sobre aquests.
2. Servidor web que es faci servir (Apache o IIS) per tal de crear el lloc web.
3. MySQL a nivel administrador ja que cal crear una base de dades, un usuari i gestionar els permisos de l'usuari creat sobre la base de dades creada.

14. INSTRUCCIONS D'INSTAL·LACIÓ

Els passos a seguir per instal·lar l'aplicació són:

1. **Base de dades**
    a. Crear una base de dades.
    b. Crear un usuari amb permisos de consulta, insert i update sobre la base de dades.
    c. Executar el codi SQL que genera totes les taules i les dades imprescindibles.
    d. Introduir un usuari administrador a la taula Usuaris. Només cal omplir el NIU, el camp *admin* igual a 1 i l'idioma per defecte (ca=català, en=anglès).
2. **Aplicació**
    a. Generar una espai web.
    b. Copiar els fitxers PHP que contenen el codi.
    c. Modificar el fitxer /config/config.php introduïnt les dades següents:
        i. Nom del departament.
        ii. Nom de la facultat.
        iii. Primeres 5 lletres/números del Centre de Cost del departament.
        iv. Adreça electrònica on rebre els avisos.
        v. Nom de l'adreça electrònica.
        vi. Mida màxima en bytes dels adjunts.
        vii. Llista d'extensions de fitxers que es permeten adjuntar separades per comes (Per defecte: PDF).
        viii. Nom del servidor de bases de dades.
        ix. Nom de la base de dades.
        x. Nom de l'usuari.
        xi. Paraula de pas de l'usuari.

Un cop realitzats aquests passos ja tindrem l'aplicació en marxa.

15. CONCLUSIONS DEL TREBALL DE FINAL DE CARRERA

Amb la realització d'aquest treball de fi de carrera s'ha realitzat amb èxit la transformació del procés de creació de formularis de gestió de Programació Estructurada a Programació Orientada a l'Objecte.

S'han desenvolupat dues classes, Formulari i Dada, que faciliten enormement la realització de nous formularis així com la refactorització i modificació de formularis ja fets.

La connexió amb la base de dades es fa mitjançant PDO (Objectes de Dades en PHP) fent servir Programació Orientada a l'Objecte que a més fa l'aplicatiu independent del Sistema Gestor de Bases de Dades triat.

S'ha aconseguit amb èxit la modularització de l'aplicatiu. Ara cada departament pot triar quins formularis farà servir de tota la llista de formularis disponibles.

L'objectiu secundari d'implementar un flux controlat per rols no ha estat possible ja que per problemes de confidencialitat i de seguretat no s'ha tingut accés a la base de dades on es relacionen els Investigadors Principals (IP) amb els projectes que estan duent a terme. Sense accés a aquesta informació es podria haver fet igualment però cada departament hauria de mantenir aquesta informació manualment a l'aplicatiu i suposaria una duplicació de feina i de introducció de dades, justament una de les coses que la e-administració intenta evitar. Per aquest motiu es va decidir esperar a poder accedir a aquesta informació per desenvolupar aquesta part.

A continuació es compara l'anterior sistema amb el nou segons diferents aspectes:

RENDIMENT

Des de el punt de vista del rendiment de les pàgines web generades amb Programació Estructurada i amb Programació Orientada a l'Objecte no s'ha vist un increment significatiu en el temps de càrrega.

Per fer les comparacions s'ha aprofitat un del formularis refactoritzats, Inscripció a congressos, i s'ha comparat el temps de càrrega facilitat per les eines per desenvolupadors del navegador Mozilla Firefox. A la figura 6 es pot veure un exemple en el que el temps total de càrrega de la pàgina generada amb Programació Estructurada és de 0,39s amb un temps de càrrega del fitxer PHP de 29ms. A la figura 7 un exemple de pàgina generada amb Programació Orientada a l'Objecte on es pot apreciar que els temps de càrrega són molt similars.

Aquestes proves s'han realitzat diverses vegades, fent servir memòria cau i sense fer-la servir, amb uns resultats sempre molt equilibrats.

La conclusió és que en aquest cas el rendiment no s'ha vist afectat, ni negativa ni positivament, amb el canvi de paradigma.

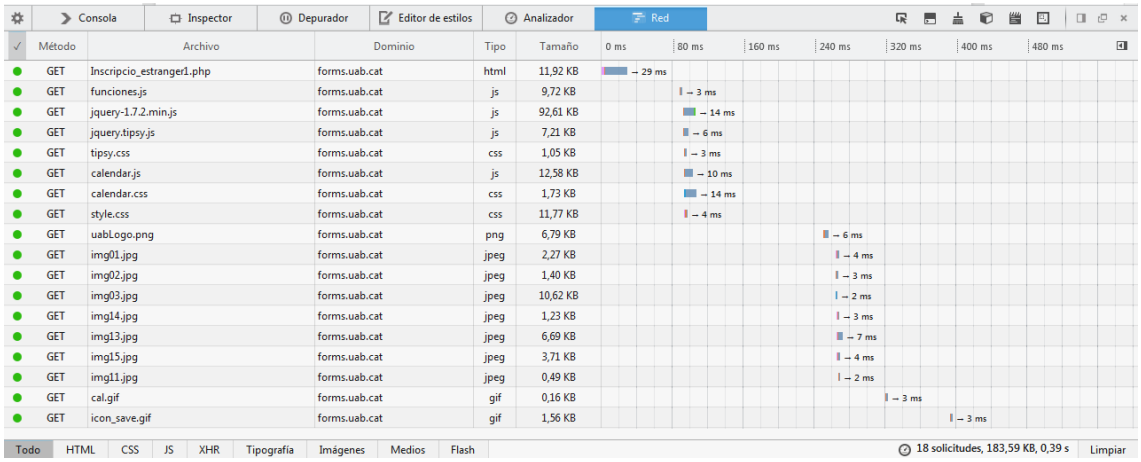


Figura 6

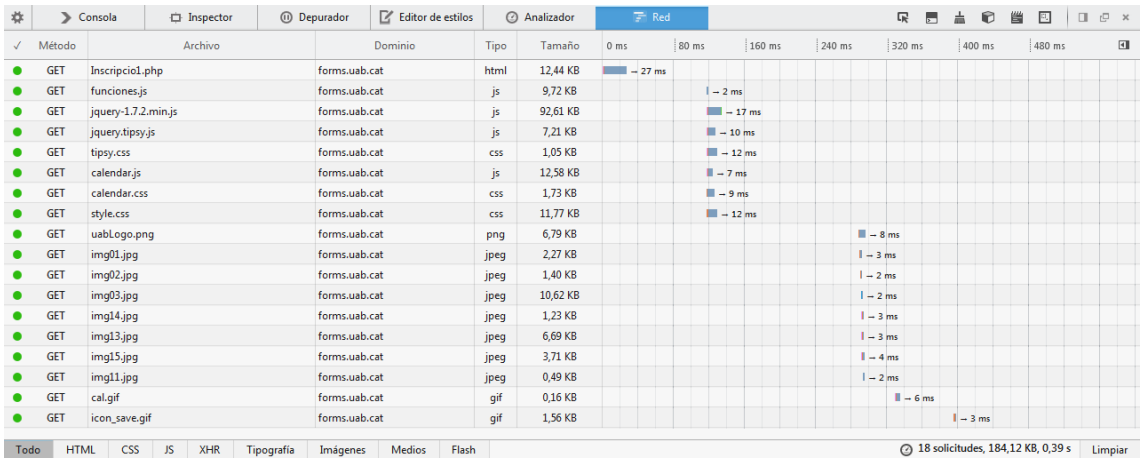


Figura 7

MIDA DELS FITXERS

Un altre aspecte en el que ens hem fixat és en la mida dels fitxers PHP fent servir un tipus de programació i l'altre. En les mateixes figures 4 i 5 es pot veure que la mida dels fitxers és també bastant similar sent el fitxer PHP amb Programació Estructurada lleugerament més petit, 11,92 KB contra 12,44 KB. Aquest resultat ha estat anàleg a la resta de formularis refactoritzats.

La conclusió és que fent servir les classes no ens estalviem generar codi.

TEMPS DE TREBALL

L'aspecte més important que volíem avaluar i un dels objectius del treball era l'estalvi de temps en la realització de nous formularis fent servir la Programació Orientada a l'Objecte.

Per fer l'avaluació s'ha fet un registre del temps destinat a fer els nous formularis per comparar amb el temps que es va destinar en fer anteriors formularis. El resultat, tal i com ja s'apuntava a l'apartat 11.4 ("Primeres conclusions després de fer les refactoritzacions"), ha estat força bo.

En aquest moment la tasca de fer disseny de la base de dades i pensar bé en quin tipus de dades ofert per les classes s'encaixa cada apartat del formulari esdevé la tasca principal. Donat que es fan servir plantilles i codi html i css ja utilitzat en anteriors formularis i que les classes desenvolupades fan molt més senzilla la tasca de programar el temps destinat a programació s'ha reduït en un 70% i ha esdevingut una feina secundaria.

16. PROJECCIÓ A FUTUR

Com ja s'ha explicat a la introducció aquest Treball de Fi de Carrera forma part d'un projecte que ni tan sols va començar com a tal. D'una pàgina web per descarregar uns formularis en Excel s'ha passat a una aplicació de formularis on recollir dades per que es realitzin un conjunt de tasques burocràtiques.

Ara mateix es tracta d'un dels projectes més interessants de la universitat ja que, mica en mica amb la seva implantació, està traient a la llum un dels principals problemes als que s'enfronta una organització tant gran com la universitat: malgrat un conjunt comú de normes, cada facultat, inclús cada departament, treballa i té uns requisits burocràtics diferents. Això no només complica la mobilitat del personal funcionari, que veu com passa un temps fins que és totalment productiu ja que ha d'aprendre les diferents formes de fer si no que també fa la feina del professorat més complicada ja que en alguns casos treballen a diferents facultats veient que les peticions son diferents. També treu a la llum les dificultats per accedir a dades que la universitat té emmagatzemades en diferents bases de dades. Un dels objectius secundaris del treball no es poder assolir per que no s'ha aconseguit el permís per accedir a aquestes dades però si ha fet veure que calen una sèrie de serveis web per facilitar informació entre diferents aplicacions de forma que la no duplicació de dades sigui una realitat.

La implantació d'aquest projecte ha motivat la creació d'una comissió que recull les diferents necessitats de cada facultat i de cada departament i intenta arribar a un acord en la forma de fer els processos i les tasques. Dels acords es deriven modificacions en els formularis, que estan en continu procés de millora i adaptació. Aquest és un dels principals motius per deixar de banda la Programació Estructurada i passar a una Programació Orientada a l'Objecte.

D'un petit projecte s'està derivant una profunda reforma a nivell de tota la universitat, revisant processos i unificant criteris.

Amb les actuals classes desenvolupades per aquest Treball de Fi de Carrera es té una bona base per continuar desenvolupant nous formularis, per modificar ràpida i eficaçment els ja existents i s'ha obert la possibilitat que més persones entrin a formar part del projecte, que amb la redacció d'una API tindran més fàcil començar a desenvolupar. S'ha intentat que els mètodes siguin prou clars i senzills per que qualsevol persona pugui començar a treballar en el projecte amb poc temps d'aprenentatge i s'ha fet un esforç per que les classes siguin prou versàtils per poder fer nous mètodes i modificacions segons les noves necessitats que vagin sorgint.

17. CONCLUSIONS PERSONALS

La realització d'aquest treball ha estat una feina molt gratificant des de el punt de vista personal.

He aprés molt i m'ha mostrat una direcció a seguir. Aquest treball només és el principi d'un camí. Estic segur que el codi de les classes desenvolupades és millorable i que amb l'ús en trobaré errors a corregir. Per sort aquest és un projecte viu i engrescador que crec que em permetrà millorar i continuar aprenent.

ANNEXOS

ANNEX 1. LLIURABLES DEL PROJECTE

Fitxer	Descripció
CodiFont_Cortes_JoseManuel.pdf	Codi font en PHP de les classes desenvolupades.
Presentacio_Cortes_JoseManuel.pdf	Presentació en PDF del Treball de Final de Carrera.

ANNEX 2. LLIBRERIES/CODI EXTERN UTILITZAT

Durant la realització del projecte s'han fet servir les següents llibreries:

Nom	Tigra Calendar
Descripció	Llibreria que permet mostrar un calendari per seleccionar una data en concret. Es fa servir per triar una data als formularis o per canviar-la.
Versió	4.0.2
Llenguatge	Javascript i CSS
Desenvolupador	Soft Complex
Direcció web	http://www.softcomplex.com/products/tigra_calendar/
Llicència	Lliure

Nom	Tipsy
Descripció	Llibreria basada en JQuery. Permet mostrar el camp "title" d'un element HTML amb diverses opcions de format. Es fa servir per oferir informació addicional a l'usuari. Ex: camp obligatori, no editable, etc.
Versió	1.0.0a
Llenguatge	Javascript i CSS
Desenvolupador	Jason Frame (jason@onehackoranorher.com)
Direcció web	http://onehackoranother.com/projects/jquery/tipsy/
Llicència	The MIT License