5734 – GSAI: GESTIÓ D'ESPAIS

Memòria del projecte

d'Enginyeria Informàtica

realitzat per

Ferran Alcázar Sánchez

i dirigit per

Marc Tallo Sendra

Escola d'Enginyeria

Bellaterra, *Setembre* de *2014*

Universitat Autònoma de Barcelona

escola d'enginyeria

El sotasignat, Marc Tallo Sendra

professor/a de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Ferran Alcázar Sánchez

I per a que consti firma la present.

Signat: .....

Bellaterra, 12 de Setembre de 2014

FULL DE RESUM – PROJECTE FI DE CARRERA DE L'ESCOLA D'ENGINYERIA

Títol del projecte: GSAI – GESTIÓ D'ESPAIS	
Autor[a]: Ferran Alcázar Sánchez	**Data:** *Setembre 2014*
Tutor[a]/s[es]: Marc Tallo Sendra	
Titulació:	

Paraules clau

- Català: gestió, espais, reserves, PHP, KumbiaPHP, framework.
- Castellà: gestión, espacios, reserves ,PHP, KumbiaPHP, framework.
- Anglès: management, spaces, reservations, PHP, KumiaPHP, framework.

Resum del projecte

- Català:

L'aplicació GSAI et permet gestionar els diferents espais que existeixen a la teva empresa. Et dona la capacitat de crear els teus espais personalitzats. Aquests espais s'agrupen per entitats així pots diferenciar cada zona de la teva empresa. Cada usuari pot crear reserves dins d'un espai i esborrar-la si es necessari.

- Castellà:

La aplicación GSAI te permite gestionar los diferentes espacios que existen en tu empresa. Te da la capacidad de crear tus espacios personalizados. Estos espacios se agrupan por entidades así puedes diferenciar cada zona de tu empresa. Cada usuario puede crear reservas dentro de un espacio y borrarla si es necesario.

- Anglès:

The GSAI app allows you to manage different spaces that exist in your company. It gives you the ability to create your personalized spaces. These areas are grouped in entities so you can differentiate every area of your business. Each user can create a reservation within a space and delete it if it's necessary.

ÍNDEX

1 INFORME PREVI

1.1 Objectius del projecte

Aquest projecte es basa en dos grans propòsits, un de didàctic i l'altre personal.

L'objectiu didàctic és desenvolupar una projecte pràctic per aprendre els passos necessaris que comporta la implantació d'una aplicació en un entorn de treball. Aprendre tots els mecanismes per portar una simple idea a una aplicació acabada i funcional.

L'objectiu personal és solucionar un problema actual detectat a l'empresa on treballo i que es gestiona de forma ineficaç. La creació d'una eina que pugi ésser utilitzada des de totes les plataformes possibles.

Finalment amb la creació de l'aplicació es té l'objectiu de que sigui escalable a altres empreses, tan privades com públiques.

1.2. Breu introducció a l'estat del art del tema proposat

El fet d'aparcar un vehicle en una plaça de pàrquing és una tasca poc informatitzada. La majoria d'aplicacions que trobem al mercat actual es basen en la busca de pàrquings públics a una ciutat o el fet de localitzar el vehicle en la posició d'un mapa per tal de no despistar-se a l'hora d'anar a buscar-lo. Alguns pàrquings comencen a incorporar llums per senyalitzar places lliures però no acaba de ser pràctic quan està molt massificat el pàrquing.

Es troba a faltar una eina que abans de sortir de casa es pugui saber amb exactitud si un pàrquing té places lliures. També en certs entorns de treball les places s'assignen segons el estatus de la persona a l'empresa i la gent que no té plaça assignada s'ha de conformar en aparcar fora del recinte en algun pàrquing públic o directament al carrer.

A l'empresa on estic treballant hi ha una curiosa distribució de les places de pàrquing. Actualment hi ha 3 pàrquings i segons el rang del teu lloc de treball pots accedir a una plaça de pàrquing.

El problema està en que no tothom té plaça al pàrquing de més categoria, i per tant ha d'aparcar al pàrquing de fora. Existeix un tercer pàrquing en el qual s'ubiquen les visites a l'empresa, treballadors externs, i personal de servei. Degut a la forta incorporació de personal en tots els àmbits les places estan molt buscades.

El pàrquing 1, té les places que estan just al costat de les oficines i és on aparquen directors i treballadors amb més nivell. El pàrquing 2 està destinat a treballadors interns de menys categoria. El pàrquing 3 és per visites i externs.

El treballadors externs només poden aparcar al pàrquing 3, però si algú del pàrquing 1 no ve a treballar un dia, podem aparcar a la seva plaça si dona el seu permís. Els treballadors interns també tenen aquesta possibilitat d'aparcar a places del pàrquing 1. Tota aquesta gestió es fa en persona, via aplicacions de comunicació com Whatsapp i

és molt caòtica ja que mai se sap quan falta algú, si ja ha donat aquesta plaça a algú altre o si la persona que falta tornarà demà o estarà uns dies sense venir. I aquí es on entra la aplicació, una gestió de tots aquests pàrquings, amb les places de pàrquing assignades a cada persona i que cadascú pugi gestionar la seva plaça i assignar-la a la persona que desitgi.

1.3. Estudi de viabilitat del projecte

Un cop hem definit el tema de la aplicació i el seu funcionament sobre paper toca estudiar la viabilitat del projecte. Aquesta es basa en quatre punts bàsics.

El primer punt és cercar en quina plataforma serà desenvolupada l'aplicació. Necessitem que sigui accessible, és la característica més important. Es pot pensar que lo ideal seria desenvolupar una aplicació per una plataforma mòbil, però això implica haver de crear dos aplicacions (com a mínim) donat que el mercat està repartit entre iOS i Android. Donat que el temps per la creació del projecte és limitat i es vol arribar al màxim de persones es desenvoluparà en entorn web. D'aquesta manera podem accedir via ordinador de treball o personal, o via navegador mòbil.

El següent punt és buscar en quin llenguatge s'ha de programar l'aplicació. Donada la diversitat d'opcions es farà un petit estudi per veure quin és el més adient pel tipus d'aplicació que s'està desenvolupant. Haurem de complir tots els objectius bàsics que ens hem proposat.

Un altre punt important tracta de les futures actualitzacions i adaptacions de l'aplicació. Aquesta s'haurà de dissenyar de forma que sigui fàcil programar noves funcionalitats. També és un tema important l'adaptació del programa. Estem dissenyant una aplicació per un pàrquing en concret, però totes les característiques hauran de ser escalables a

qualsevol tipus de pàrquings, fins i tot els espais públics podrien utilitzar aquesta aplicació per informar de les places lliures als usuaris.

L'últim punt és el cost econòmic. S'utilitzarà programari lliure i es minimitzarà el cost per tal de que el projecte no tingui cap despesa important associada.

Com a conclusió podem afirmar que el projecte és totalment viable.

1.4. Planificació temporal del treball

- **Elaboració del informe previ**: realitzarem un informe previ per tal d'estudiar la viabilitat del projecte.

- **Creació de l'aplicació**: Busca de les eines necessàries per la creació de l'aplicació de gestió. Necessitem saber en quina plataforma (web, aplicació Windows, etc), llenguatge estarà implementada la nova aplicació. Aquesta tasca la dividirem en documentació, programació i actualitzacions.
    - **Documentació**: adquirir el coneixement necessari per desenvolupar l aplicació.
    - **Programació**: Implementació de l'aplicació.
    - **Actualitzacions**: elaborarem una llista de possibles actualitzacions per l'aplicació i si es possible s'implementaran.
- **Memòria**: creació d'una memòria del projecte.

1.5 Altres comentaris

No hi ha cap comentari a destacar.

2 INTRODUCCIÓ

2.1 Presentació

Aquest projecte neix d'una petita necessitat trobada a la empresa on estic treballant. La seva finalitat és crear una aplicació web que ens permeti gestionar espais. En qualsevol empresa, universitat, hospital o entitat les persones conviuen en espais els quals ocupen per diferents tipus d'inquietuds. Aquests espais abans de l'arribada tecnològica de la informàtica han estat gestionats de forma manual, via comunicació verbal o amb una simple fulla de paper. De fet a l'actualitat he comprovat que encara es segueixen fent servir aquests mètodes. Ara tenim una eina molt més eficaç i potent que un simple full de paper, la informàtica, i així neix aquest projecte per cobrir aquesta necessitat.

En un primer moment tal i com s'explica a l'informe previ, el projecte va sorgir de la necessitat de gestionar els pàrquings de l'empresa on estic actualment treballant. Donada la peculiar estructura d'aquests pàrquings era una molt bona idea la creació d'una aplicació per poder gestionar-los. Però a mida que el projecte va anar avançant es va decidir pensar de forma més amplia i es va decidir fer una aplicació que pugi gestionar qualsevol tipus d'espai d'una empresa.

2.2 Objectius

A l'informe previ vam remarcar que els dos grans punts importants d'aquest projecte eren l'aprenentatge del desenvolupament d'una aplicació web i la solució de la gestió dels pàrquings de la empresa on treballo. L'objectiu personal del projecte ha variat a mida que ha passat el temps. Hem ampliat l'objectiu de gestió de pàrquings d'una empresa, a la creació d'una eina on puguem registrar qualsevol espai d'una empresa o lloc comú de totes les entitats on les persones desenvolupen diferents activitats.

L'objectiu s'ha ampliat degut a que la idea inicial de la gestió de diferents pàrquings queda englobada en la gestió d'espais que desenvoluparem en aquest projecte. Així podrem registrar com a espais llocs com sales de reunions, aules d'estudi, despatxos, etc.

L'objectiu didàctic d'aquest projecte és desenvolupar una aplicació senzilla, que no necessiti cap corba d'aprenentatge complicada per l'usuari bàsic i que cobreixi les necessitats bàsiques de la gestió d'un espai. En el temps que porto treballant per una gran empresa he detectat que quan més complicada i més opcions té una aplicació els usuaris finals fan servir lo essencial i quasi mai s'aprofita el potencial de l'eina. A més a més aquesta fita didàctica m'ajudarà a veure com es desenvolupa una aplicació des de zero i quins camins s'ha de seguir per portar a terme un projecte d'aquestes característiques.

2.3 Estat del art

En totes les empreses, ja siguin petites o grans multinacionals, en les universitats, escoles, hospitals i un gran número de llocs on conviuen les persones cada dia existeixen espais comuns, compartits que necessiten d'una organització per tal de ser gestionats correctament. Si parlem d'un pàrquing amb places assignades a persones concretes, és necessari controlar de qui és cada plaça. Si pel contrari les places son lliures, potser es vol saber quin numero de llocs estan buits en un moment concret. Tal i com es va descriure al informe previ, un pàrquing pot ser molt complex i potser a la empresa li interessa cedir certes places de pàrquing assignades a persones que no tenen plaça per un temps concret (com les vacances).

En la nostra aplicació no s'implementarà cap gestió personalitzada, ja que el que es busca és la creació d'una aplicació que gestioni diversos espais. És a dir, una aplicació que englobi tot tipus d'espais. Hem començat parlant de pàrquings al informe previ doncs era l'objectiu principal, però amb les noves fites es busca ampliar les mires i englobar tot tipus d'espais.

En la empresa on estic treballant per exemple, a part dels pàrquings hi han despatxos que serveixen per fer petites reunions o trucades en conferencia. Aquests també poden ser considerats espais comuns que necessiten ser gestionats. En les universitats o escoles, les aules estan assignades a certes assignatures o professors en diferents horaris i potser amb una simple fulla de càlcul es poden visualitzar totes les aules i horaris però a vegades és necessari gestionar classes que s'han canviat d'horari o que per algun imprevist. Aquí és on la nostra aplicació seria d'utilitat ja que una simple fulla de càlcul es queda curta per aquestes gestions.

2.4 Motivacions

Les motivacions del projecte van sorgir degut a l'observació d'un problema de gestió en les places de pàrquing de la empresa on estic treballant.

L'empresa té tres tipus de pàrquing, dos de privats i un públic. El pàrquing públic va començar a tenir mancança de llocs degut a un augment de personal contractat a l'empresa. Un dels pàrquings privats és només per empleats de perfil bàsic i les places no estan assignades. Aquest també va notar un augment de volum fins pràcticament esgotar totes les places lliures. L'últim pàrquing és el que queda just a sota de les instal·lacions i és per empleats de perfil alt, és a dir, directors. Aquest té places assignades i molts cops queden places que no s'utilitzen alguns dies.

Els directors degut als viatges i compromisos a vegades deixen les places del pàrquing a altres empleats, però es fa de forma verbal i només quan estan de forma present. Així que molts cops queden places lliures que es podrien aprofitar per altres empleats. Quan es dona el cas que s'aprofiten, a vegades sorgeixen confusions de places deixades a varies persones. Tot això passa degut a que la comunicació verbal és molt efímera i no queda res registrat a cap lloc. Fins i tot per comunicacions molt noves com per exemple Whatsapp surten els mateixos problemes, ja que el canal de comunicació és individual o en un grup tancat i no arriba a tot els usuaris desitjats.

D'aquí va néixer la motivació de crear una aplicació web per l'empresa, on tothom pogués registrar les seves places de pàrquing i tots els empleats poguessin deixar la seva plaça lliure.

2.5 Estructura de la memòria

La memòria esta estructurada en vuit parts. Tenim una introducció on presentarem el projecte i explicarem els objectius inicials, l'estat de l'art i les motivacions.

La següent part farem un estudi de viabilitat, on marcarem uns objectius més profunds i descriurem les especificacions funcionals, no funcionals i tècniques. També és important la planificació del projecte on marcarem uns límits per cada part del projecte. També farem una petita valoració de la viabilitat del projecte i els seus riscos. Finalment determinarem si el projecte és viable a les conclusions.

La quarta part farem una explicació del fonaments teòrics del framework triat. Aquesta explicació serà de forma resumida però que ens permeti entendre el funcionament del framework.

La cinquena part farem un anàlisis de com portarem tots aquests fonaments teòrics a la practica per poder desenvolupar l'aplicació.

La sisena part plasmarem com hem implementat l'aplicació, com s'instal·len tots els fitxers necessaris per desenvolupar i que funcioni la nostra aplicació. Descriurem tots els processos creats i com interaccionen entre ells.

La setena part de la memòria està dedicada al joc de proves. Ens dedicarem a provar totes les parts de l'aplicació, menús, formularis, etc.

Finalment tindrem les dues ultimes parts que seran les conclusions i la bibliografia del projecte.

3 ESTUDI DE VIABILITAT

3.1 Introducció

Ara que ja tenim definits els objectius i les motivacions del projecte, és l'hora de fer un estudi de viabilitat.

En aquest estudi tractarem de dilucidar si el projecte que volem portar a terme és viable. Marcarem els objectius de forma més precisa del que vàrem fer a l'informe previ i crearem unes especificacions funcionals, no funcionals i tècniques per al nostre projecte.

També marcarem una planificació per tal d'organitzar el temps que tenim per portar a terme el desenvolupament de l'aplicació.

Finalment farem una valoració de si aquest projecte es pot realitzar i avaluarem els riscos que hem de prendre per tal de implementar aquesta aplicació. Les conclusions d'aquest estudi marcaran el futur del projecte.

3.2 Objectius

La viabilitat del projecte té quatre objectius bàsics:

- Escollir en quina plataforma desenvoluparem l'aplicació.
- Una programació en un llenguatge accessible i fàcil.
- L'escalabilitat de l'aplicació.
- El cost econòmic ha de ser zero o el mínim possible.

El primer objectiu aborda el tema d'escollir en quina plataforma desenvoluparem l'aplicació. Volem buscar una plataforma que s'adapti a les necessitats d'una empresa i sigui accessible des de un entorn segur. La creació de l'aplicació podria estar orientada a una universitat o simplement per el públic en general però s'ha decidit que ens centrarem en que l'aplicació va dirigida a una empresa que vol gestionar els espais que té disponibles a la seva seu central. Si fem una ullada al nostre entorn i a les noves tecnologies que han evolucionat els últims anys, la plataforma natural a escollir seria la mòbil. Tenim tres grans plataformes mòbils, iOs, Android i Windows Phone, però donat que no podem desenvolupar un projecte per les tres ens hauríem de centrar en una plataforma en concret. Segurament la nostra empresa utilitza una d'aquestes tres plataformes però donat el meu punt de vista les empreses canvien per molts motius els seus proveïdors de terminals i d'aquí uns anys podria ser que la nostra aplicació ja no servis donat el canvi de plataforma per part de la empresa.

Això ens fa arribar a la conclusió de que la millor plataforma per implementar la nostra aplicació sigui el entorn web. Aquest podria ser accessible via navegador mòbil i des de els ordinadors de l'empresa. L'aplicació es desenvoluparà sota Windows 7 i serà accessible des de qualsevol navegador de Windows, Linux o Mac o smartphone. Aquests últims ara tenen uns navegadors tan potents com podem trobar a qualsevol ordinador per tan la nostra web serà accessible des de totes les plataformes.

El pròxim objectiu es bada en trobar una llenguatge de programació per la nostra aplicació. Volem un llenguatge de programació que sigui prou conegut per tenir una bona documentació. Volem una implementació que sigui escalable en un futur i es pugin fer modificacions sense haver d'experimentar una corba d'aprenentatge complicada. En aquest punt es té molt clar que la opció més adient per ser el llenguatge triat és PHP, un dels llenguatges més importants i més utilitzat pel desenvolupament d'aplicacions web. Aquest compleix totes les característiques que necessitem pel nostre projecte. Junt amb PHP utilitzarem una base de dades com MySQL. Els dos seran una combinació prou potent per implementar tot el que necessitem.

El següent objectiu és la escalabilitat de l'aplicació. Pot semblar un objectiu secundari ara mateix però es un punt important a tenir en compte. En un futur necessitarem actualitzar, canviar parts del programa o simplement adaptar-lo a noves empreses i necessitem que l'aplicació tingui aquesta funcionalitat. Aquesta característica està lligada amb el segon objectiu de buscar un llenguatge ben documentat i senzill d'aprendre.

L'últim objectiu que ens hem marcat és econòmic. Es buscarà un cost zero o el mínim possible. PHP i MySQL són programari lliure i no tenen cap cost. A part dels paquets necessaris per utilitzar PHP i la base de dades necessitarem varis programes per implementar la nostra aplicació. Tot seguit mostrem una relació dels més importants:

- El sistema operatiu Windows 7 edició servida gracies al conveni entre la Universitat Autònoma de Barcelona i Microsoft. Totalment gratuït. No és el sistema preferit dels programadors, però ens servirà perfectament per implementar tot el codi i fer el joc de proves per l'aplicació.
- Notepad++ , un dels més famosos editors de fitxers de codi que és totalment gratuït.
- Varis navegadors com Chrome, Firefox, Opera I Internet Explorer en les seves versions més noves. Tots són gratuïts.

3.3 Estat de l'art

En aquest projecte quan hem avaluat l'estat de l'art prèviament hem vist que l'aplicació que volem desenvolupar no forma part de la col·lecció d'aplicacions que tota empresa té. La majoria d'empreses tenen un processador de textos, fulls de càlculs, un sistema d'identificació pels treballadors, etc. En canvi, una gestió d'espais és una aplicació massa especifica i que la majoria d'empreses no creuen que necessitin. Per tant, aquesta gestió es fa de manera més manual en full de càlcul o de forma escrita en un paper. El nostre objectiu és fer veure a aquestes empreses de que una aplicació de gestió d'espais és necessària i molt útil.

Fent una busca per la xarxa d'Internet d'una aplicació que puguem provar i ens doni una solució vàlida al tema que estem tractant en aquest projecte, hem vist que no existeix cap que puguem fer servir i que s'adapti a la nostra problemàtica.

3.4 Especificacions (funcionals/no funcionals/tècniques)

Passem a descriure les especificacions funcionals, les no funcionals i les tècniques.

3.4.1 Especificacions funcionals

Definirem les especificacions funcionals que hem pensat que hauria de tenir la nostra aplicació. En el nostre sistema hi ha dos tipus d'actors, la persona que s'encarrega de administrar l'aplicació i l'usuari final. S'ha pensat que al ser una aplicació per una empresa, aquesta ja disposa d'una base de dades amb els seus empleats (usuaris finals) i no els farem registrar-se de nou a la nostra aplicació, volem aprofitar els usuaris ja creats i els seus credencials.

Els usuaris finals al accedir a l'aplicació poden entrar amb el seu usuari, fer login i accediran a una pantalla on trobaran els tipus d'espais disponibles (anomenats a partir d'ara entitats). Seleccionaran la entitat en el que volen fer una reserva i accediran a una altre pantalla on veuran tots els espais disponibles d'aquell tipus d'espai. Quan sàpiguen en quin espai volen fer la reserva podran pitjar en ell i entrar a la pantalla de reserva. Aquí mitjançant un formulari faran la reserva.

L'usuari administrador tindrà un petit rol dins l'aplicació. Els tipus d'espais i espais han estat prèviament pactats amb l'empresa i s'han creat des de la base de dades, ja que espais com despatxos o pàrquings no es creen de la nit al dia i no volem que els usuaris finals esborrin o editin aquests espais. Tot i que l'empresa prefereix que no hi hagi possible edició d'aquests espais, es crearan uns petits formularis per donar d'alta els espais nous de forma més senzilla que crear-los a la base de dades. Aquesta feina només la podrà fer el usuari administrador.

Els dos tipus d'usuaris podran sempre tancar la sessió mitjançant un botó i sortir de l'aplicació.

Els usuaris finals podran crear tantes reserves com vulguin al sistema, ja que per exemple al reservar un espai com un pàrquing volem reservar amb antelació varis dies. Seguint amb el tema de les reserves, els usuaris podran editar i esborrar les reserves creades per ells mateixos, no les reserves d'altres usuaris.

3.4.2 Especificacions no funcionals

Les especificacions no funcionals del projecte que definirem marcaran quin framework buscarem per tal de complir els requisits de la nostra aplicació. Necessitem una aplicació que tingui un bon rendiment i funcioni perfectament amb els nous navegadors que surten al mercat.

Ha de ser una aplicació que tingui implementada de sèrie una seguretat i no permeti l'entrada de persones no autoritzades. A més volem tenir una aplicació que sigui estable i no generi errors. També es important que la interfície sigui amigable i no sigui molt complicada de fer servir.

El cost de la nostra aplicació ha de ser pràcticament zero, ja que buscarem programes i aplicacions per desenvolupar-la que siguin de llicencia lliure.

Finalment cal a dir que una de les propietats més importants que necessitem és que l'aplicació es pugi actualitzar, tan a nivell d'adaptació a noves tecnologies com a nivell funcional per adaptar noves funcionalitats.

3.4.3 Especificacions tècniques

La part tècnica de les especificacions ha de cobrir la necessitat de que el framework ha de permetre accedir als registres a la base de dades de forma senzilla per tal de mostrar la informació que emmagatzemen a cada taula. Aquesta funcionalitat s'anomena ActiveRecord.

Hauria d'implementar algun sistema tipus CRUD per poder llegir i esborrar registres.

També s'haurà de fer servir PHP5 que és la ultima versió del llenguatge i incorpora totes les millores de seguretat i optimització.

La base de dades haurà de ser semblant a MySQL o aquesta mateixa ja que per la implementació del programa necessitem una base robusta i amb les característiques d'aquesta.

El patró model vista controlador ha d'estar present al nostre disseny ja que les bondats ens ajudaran en la tasca de programació.

3.5 Planificació

Per portar a bon terme aquest projecte necessitem una bona planificació del treball. Ja que tenim un temps limitat passarem a descriure el passos principals que hem de donar.

		Abril	Maig	Juny	Juliol	Agost	Setembre
Documentació	Elecció framework						
	Estudi framework						
Programació	Instal·lació principal						
	Creació Base de dades						
	Programació aplicació						
Actualitzacions							
Memoria							

IL·LUSTRACIÓ 3-I

3.5.1 Documentació

El primer pas un cop triat el llenguatge de programació i les eines que farem servir per la seva realització, hem de valorar si farem el projecte directament amb PHP o buscarem l' utilització d'un framework. Aquesta ultima opció crida bastant l'atenció ja que la proliferació de frameworks fa molt atractiva la seva utilització. Permeten automatitzar operacions com les CRUD, ja porten implementats el MVC i un llarg de funcionalitats que són molt interessants en un projecte.

Tenint en compte totes les bonances dels frameworks es farà un petit estudi de quin s'adapta millor al nostre projecte.

La busca de framework ens portarà un mes sencer.

Quan estiguem segurs de quin framework utilitzarem ens haurem de familiaritzar amb les funcions que té per poder utilitzar-lo correctament. Es buscaran exemples senzills i farem proves per tal de fer un aprenentatge basic de l'eina. Totes aquestes tasques ens portarà quinze dies.

3.5.2 Programació

Un cop hem après com funciona el framework ens haurem d'organitzar i pensar quines funcionalitats de l'aplicació implementem primer.

S'ha pensat que després de fer tota la instal·lació de les bases de dades i el projecte del framework, desenvoluparem un mòdul per fer el login dels usuaris.

Seguidament es crearan les taules a la base de dades que contindran les entitats, els espais i les reserves.

Tot seguit s'organitzarà com crear les estructures entitat, espai i reserva. Haurem d'implementar cada model per separat i veure el funcionament de cadascun per tal de polir la interacció entre ells.

També dissenyarem la interfície de l'aplicació. La feina per aquesta tasca pot durar 15 dies.

Finalment haurem de deixar un espai de temps per poder fer proves a l'aplicació i trobar possibles errors. Així podrem solucionar-los amb temps.

3.5.3 Actualitzacions

A mida que implementem el projecte pensarem quines actualitzacions i millores podria tenir la nostre aplicació. El nostre objectiu és crear un esquelet bàsic amb el projecte i que pugem afegir actualitzacions de forma senzilla i ordenada.

Per tal de portar a terme noves actualitzacions buscarem un framework de PHP que compleixi els requeriments inicials que ens hem proposat. Volem un projecte obert que sigui fàcil la introducció de nous mòduls.

3.5.4 Memòria

La creació de la memòria serà incremental i seguirà el mateix camí del projecte. Es prendran notes separades de cada procés i s'incorporaran poc a poc a la memòria per documentar tots els passos que hem seguit.

3.6 Valoració

Finalment ens toca fer una valoració de tots els aspectes que hem tractat al l'estudi de viabilitat.

Volem una aplicació amb PHP5. El llenguatge ha de tenir el recolzament de la comunitat a més de que aquesta sigui activa i participativa. Volem una aplicació que el seu desenvolupament sigui senzill i que el framework que triem pugui fer automatitzacions de tips CRUD (crear, llegir, actualitzat i esborrar) o ABM (alta, baixa i modificació).

També necessitem una aplicació que sigui flexible i s'adapti a les especificacions i pròximes actualitzacions.

Que suporti el patró model - vista – controlador ja que ens facilitarà molt el treball de desenvolupament.

En fi, totes aquestes característiques les podrem trobar a un framework i haurem de valorar la utilització de un per tal desenvolupar la nostra aplicació.

Si parlem de la planificació veiem que tenim el temps just per portar a terme el projecte i qualsevol contratemps pot ser crític per poder acabar. Haurem de posar tot l'esforç possible per poder finalitzar-lo i complir amb el termini.

3.7 Riscos i costos

Després d'analitzar tots els aspectes del estudi de viabilitat podem afirmar que no hi ha cap risc important en l'elaboració de l'aplicació. Els projectes mai tenen risc zero, sempre sorgiran imprevists però s'ha buscat la simplicitat per tal de poder assolir tots els objectius del projecte.

La pròpia busca d'un framework sobre el que implementarem la nostra aplicació es un petit risc, ja que si no trobem un d'adequat el projecte podria perillar i no obtindríem els objectius que ens hem marcat.

Un cop haguem trobat un framework sobre el que treballar hi haurà un risc innat en el desenvolupament de l'aplicació. Ens podem trobar que el funcionament del framework no sigui el desitjat i no assolim totes les especificacions del projecte. Un canvi de framework un cop començat el projecte es un altre tipus de risc que haurem de prendre ja que podria retardar la planificació que ens hem marcat del projecte.

Sobre els costos de l'aplicació ja hem comentat abans que pràcticament són zero a nivell d'utilització de programes. Les eines utilitzades i el framework en si són totalment gratuïts i no aporten cap cost al nostre projecte.

L'únic cost associat a aquest projecte vindrà donat dels desplaçaments que hem mantingut amb l'empresa per la presa de requeriments. Aquest pot pujar a la xifra de 50€ entre gasolina i desplaçament.

També hi haurà un cost que pertany a les hores dipositades pel desenvolupament de l'aplicació. Si prenem com a inici l'abril ja que és quan es va entregar l'informe previ, i contem uns cinc mesos de treball, en els quals cada mes són vint dies laborals i cada dia s'ha treballat una mitja de dues hores. El preu d'hora es cobra a 10€. En total el cost puja a 2000€. Per tant tenim un cost total de 2050€ del projecte.

2050€ = (5 (mesos) * 20(dies) * 2 (hores) * 10€ (hora)) + 50€ (desplaçaments)

3.8 Conclusions

Com a conclusió sobre l'estudi de viabilitat es pot afirmar que és totalment factible l'elaboració del projecte.

Un cop analitzats tots els punts d'aquest estudi i tenint clar que el llenguatge de programació serà PHP hem get una recerca del framework que necessitem per implementar l'aplicació. Ha quedat descartat la opció de programar PHP a pèl, ja que ens trauria temps i segurament no aconseguiríem una organització tant eficaç en codi i funcionalitats per al nostre projecte.

Un cop iniciada la busca del framework que treballi amb PHP hem fet trobat a Internet molts articles que parlen del tema. Una conclusió a la que hem arribat és que l'elecció d'un framework és difícil, ja que condiciona totalment la implementació de l'aplicació i un cop triat no es fàcil adaptar el que tenim a un nou framework. Per tant volem fer una bona elecció i que no ens penedim més tard.

Però el que tenim clar és que necessitem un framework que treballi amb PHP5, tingui una gran comunitat darrere i implementi el MVC i el patró ActiveRecord. Finalment després de totes aquestes valoracions vam deixar com a finalistes dos frameworks, Symfony i KumbiaPHP.

- Symfony té una gran comunitat darrere, implementa el disseny Model-View-Controller (MVC), té plantilles, rutes, un sistema de fitxers propi i una llista molt interessant de característiques. Un cop analitzat ens dona la sensació de que és un framework molt potent
- KumbiaPHP té la característica principal de que ha estat implementat per la comunitat hispanoparlant. Aquest fet fa que tingui un gran suport darrere i aquest és totalment en castellà. Implementa el disseny model vista controlador, el patró ActiveRecord i un mapeig objecte-relacional. Implementa un sistema de plantilles, un sistema de rutes, un generador de formularis i un sistema de caché. Totes

aquestes característiques i moltes més, junt amb un suport totalment en espanyol fan molt atractiu aquest framework.

Com a conclusió final al estudi de viabilitat podem afirmar que el framework triat és KumbiaPHP i la nostra aplicació acaba de néixer.

4 FONAMENTS TEÒRICS

4.1 Introducció

KumbiaPHP és un framework per aplicacions webs lliures escrit en PHP5. Està basat en les pràctiques de desenvolupament web com DRY ("Don't Repeat Yourself") i el Principi de KISS ("Keep It Simple, Stupid") per software comercial i educatiu. KumbiaPHP fomenta la velocitat i eficiència en la creació i manteniment d'aplicacions web reemplaçant de tasques de codificació repetitives donant així al programador poder, control i plaer.

A més és un projecte lliure publicat sota llicencia GNU/GPL.

4.1.1 Característiques del framework

Les característiques principals del framework son les següents:

- Sistema de plantilles senzill.
- Administració de cache.
- Scaffolding Avançat.
- Mapeig Objecte Relacional (**ORM**) i Separació **MVC**.
- Suport a **AJAX**.
- Generació de formularis.
- URL amigables.
- Seguretat ACL (llistat d'accés).
- Patró **ActiveRecord** per models.
- Orientat al públic de parla castellana.

El número de requisits per instal·lar i configurar és Unix o Windows amb un servidor web i PHP5 instal·lat. A més és compatible amb motors de base de dades com MySQL, PostgreSQL, Oracle i SQLite. Aquest framework intenta proporcionar facilitats per construir aplicacions robustes per entorns comercials. Això significa que el framework és molt flexible i configurable. Està dissenyat per ajudar a reduir el temps de desenvolupament d'una aplicació web sense produir efectes adversos sobre els programadors. Aquesta característica es dona gràcies al principis bàsics en els que està basat el framework. Aquests principis són els següents:

- KISS: "Mantén-ho simple, ruc!", en anglès "Keep It Simple, Stupid".
- DRY: "No et repeteixis", en anglès "Don't Repeat Yourself".
- Convenció de configuració.
- Velocitat.
- Fàcil d'aprendre.
- Fàcil d'instal·lar i configurar.
- Preparat per aplicacions comercials.
- Simple en la major part de cassos però flexible per adaptar-se a més cassos complexes.
- Suporta moltes característiques d'aplicacions webs actuals.
- Suporta les pràctiques i patrons de programació més productius i eficients.
- Produir aplicacions fàcils de mantenir.
- Basat en software lliure.

El principal objectiu és produir aplicacions que siguin pràctiques per l'usuari final i no només pel programador. La major part de tasques que treuen temps als desenvolupadors haurien de ser automatitzades per KumbiaPHP per permetre als disenyadors centrar-se en la lògica de negoci de la seva aplicació.

A continuació mostrem una taula de la evolució de versions de KumbiaPHP:

Versió	Descripció	Data de llançament
0.3.2	Primer llançament	Gener 2007
0.4		Febrer 2007
0.4.4		Abril 2007
0.4.5		Maig 2007
0.4.6 beta		Juliol 2007
0.4.7	Última versió 0.4.x	Setembre 2007
0.5 alpha	Kumbia estrena nova estructura de directoris	Setembre 2007
0.5 RC1	Release Candidate 1	Juny 2008
0.5 RC2	Release Candidate 2	Setembre 2008
0.5 RC3	Release Candidate 3	Novembre 2008
0.5 Stable	Versió Estable Rev. 731	Gener 2009
1.0-beta1	Versió 1.0 Code Name **Spirit** Beta1	13 Agost 2009

4.2 Kumbia i PHP5

KumbiaPHP només treballa amb PHP5 ja que és la versió més avançada, estable i és el futur d'aquest llenguatge. Aquesta versió té un suport més complet en quan a la orientació d'objectes. Aquesta característica proporciona un toc professional a les aplicacions desenvolupades amb la intenció de que PHP4 vagi desapareixien.

Finalment si ens fem la pregunta de perquè hem d'utilitzar KumbiaPHP, algunes de les respostes seran les següents nombrades:

1. És molt fàcil de utilitzar (Zero-Config). Començar amb aquest framework és realment senzill, simplement has de descarregar el projecte, descomprimir la carpeta principal i començar a treballar. Aquesta filosòfica es coneguda com Convenció sobre la configuració.
2. Agilitza el treball. Crear una aplicació molt funcional amb KumbiaPHP és qüestió de poc temps, així podem donar el gust als nostres clients sense que afecti al nostre temps.
3. Separar la "lògica de la presentació". Una de millors pràctiques de desenvolupament orientat a la web és separar la lògica de les dades i la presentació, amb aquest framework és senzill aplicar el patró Model – Vista – Controlador i fer les nostres aplicacions més fàcils de mantenir i escalar.
4. Reducció del ús de altres llenguatges, gràcies als Helpers i a altres patrons com ActiveRecord evitem l'ús de llenguatges HTML i SQL (aquest últim en menor percentatge). Tot això ho proporciona el framework i ens brinda un codi més clar, natural i amb menys errors.
5. Parla en castellà. La documentació, missatges d'error, arxius de configuració, la comunitat, desenvolupadors, etc, tot està en castellà. Així és mes fàcil entendre com funciona el framework i quins errors ens retorna el codi. A més, amb la gran comunitat que hi ha darrere podem tenir una gran ajuda al nostre costat.
6. La corba d'aprenentatge és curta, i si a aquest fet li afegim l'experiència en el maneig de programació orientada a objectes, serà encara més ràpida.
7. Software lliure. No t'has de preocupar per llicenciar res, KumbiaPHP promou les comunitats d'aprenentatge , el coneixement és de tothom i cadascú sap com aprofitar-lo millor. Aquesta és la seva filosofia.
8. Aplicacions robustes. Avui en dia les aplicacions requereixen arquitectures robustes i KumbiaPHP les ofereix des del primer moment. A més proporciona una arquitectura fàcil d'aprendre i d'implementar, sense complicar-nos els conceptes i sense sacrificar qualitat.

4.3 Model Vista Controlador - MVC

Anem a parlar ara de què és MVC i una mica d'història de la seva creació. A l'any 1979 Trygve Reenskaug va crear una arquitectura per desenvolupar aplicacions interactives. En aquest disseny existien tres parts: models, vistes i controladors. Els sistemes MVC permeten fer la separació de capes d'interfície, model i lògica de control. La programació per capes és un estil de programació en el qual l'objectiu essencial és separar la lògica de negocis de la lògica de disseny. Un exemple bàsic d'això és separar la capa de dades de la capa de presentació de l'usuari.

La avantatge principal d'aquest estil, és que el desenvolupament es pot dur a terme en varis nivells i en cas d'algun canvi només afecta al nivell requerit sense tenir que revisar entre codi barrejat. A més permet distribuir el treball de creació d'una aplicació en nivells, d'aquesta manera, cada grup de treball esta totalment abstret de la resta de nivells, simplement és necessari conèixer l'API (interfície de programació) que existeix entre nivells. La divisió en capes redueix la complexitat, facilita la reutilització i accelera el procés d'acoblar o desacoblar alguna capa, o substituir-la per una altre de diferent.

En una aplicació web una petició es realitza utilitzant HTTP i és enviada al controlador. Aquest pot interactuar de moltes formes amb el model. Després el primer crida a la respectiva vista la qual obté l'estat del model que és enviat des del controlador i el mostra al usuari.

4.3.1 Com aplica KumbiaPHP el MVC?

El objectiu d'aquest patró és realitzar i mantenir la separació entre la lògica de la nostra aplicació, les dades i la presentació. Aquesta separació té algunes avantatges importants, com poder utilitzar més fàcilment en quina capa s'està produint un problema amb tan sols conèixer la seva naturalesa. Podem crear varies presentacions sense necessitat d'escriure varis cops la mateixa lògica d'aplicació. Cada part funciona independent i qualsevol canvi centralitza l'efecte sobre els altres, així podem estar segurs que una modificació en un component realitzarà bé les tasques en qualsevol part de l'aplicació.

La base de KumbiaPHP és MVC y POO (Programació orientada a objectes), i aplica aquest patró amb tres capes:

- **Models**: Representen la informació sobre la qual la aplicació opera, la seva lògica de negoci.
- **Vistes**: Visualitzen el model utilitzant pàgines web i interactuen amb els usuaris finals. Una vista pot estar representada per qualsevol format de sortida, ens referim a un xml, pdf, json, svg, png, etc.
- **Controladors**: Responen a accions d'usuari i invoquen canvis en les vistes o en els models segon sigui necessari.

Les vistes estan separades a KumbiaPHP en **templates**, **views** i **partials**. Més endavant explicarem una mica quina és la funció de cadascú.

Els controladors estan separats en parts, en crides "front controller" i en un conjunt d'accions. Cada acció sap com reaccionar davant un determinat tipus de petició.

Els models ofereixen una capa d'abstracció de la base de dades, a més dona funcionalitat agregada a dades de sessió, validació d'integritat relacional. Aquest model ajuda a separar el treball en lògica de negocis (Models) i presentacions (Vistes).

Un exemple senzill per entendre aquest funcionament seria una aplicació que corre tan en equips d'escriptori com en dispositius mòbils. Llavors podríem crear dos vistes diferents compartint les mateixes accions en el controlador i la lògica del model. El controlador ajuda a ocultar els detalls de protocol utilitzats en la petició (HTTP, mode consola, etc) pel model i la vista.

Finalment el model abstreu la lògica de dades, que fa als models independents de les vistes. La implementació d'aquest model és molt lleugera mitjançant petites convencions es pot assolir molt poder i funcionalitat.

4.3.2 El controlador

La capa del controlador conté el codi que lliga la lògica de negoci amb la presentació. Està dividida en varis components que s'utilitzen per diversos propòsits.

- El controlador frontal (front controller) és l'únic punt d'entrada a l'aplicació. Carrega la configuració i determina l'acció a executar.
- Les accions verifiquen la integritat de les peticions i preparen les dades requerides per la capa de presentació.
- Les classes "Input" i "Session" donen accés als paràmetres de la petició i a les dades persistents del usuari. S'utilitzen molt sovint a la capa del controlador.
- Els filtres son trossos de codi executats per cada petició, abans i/o després d'un controlador, inclús abans i/o després d'una acció. Per exemple, els filtres de seguretat i validació son comunament fets servir a aplicacions web.

4.3.2.1 Controlador frontal

Totes les peticions webs son portades per un sol controlador frontal (front controller), que és el punt d'entrada únic de tota l'aplicació. Quan el controlador frontal rep una petició utilitza el sistema d'enrutament de KumbiaPHP per associar el nom del controlador i de l'acció mitjançant la URL escrita pel usuari de l'aplicació.

Com a exemple posem que tenim aquesta URL:

http://localhost/kumbiaphp/micontroller/miaccion/

Aquesta crida al script "index.php" (que trobem al controlador frontal) que serà atesa com a trucada per una acció. Degut a la reescriptura de URL mai es fa una crida de forma explícita a index.php, només es col·loca el controlador, acció i paràmetres.

El "front controller" de KumbiaPHP s'encarrega de atendre les peticions, el que implica alguna cosa més que detectar una acció que s'executa. De fet, executa el codi comú a totes les accions, incloent les següents:

1. Defineix les constants del nucli de l'aplicació. (APP_PATH, CORE_PATH i PUBLIC_PATH).
2. Carrega i inicialitza les classes de nucli del framework (del fitxer boostrap).
3. Carga la configuració (del fitxer Config)
4. Descodifica la URL de la petició per determinar l'acció a executar i els paràmetres de la petició (router).
5. Si l'acció no existeix, redireccionarà l'acció del error 404 (router).
6. Activa els filtres (per exemple si la petició necessita autentificació) (router).
7. Executa els filtres , primera passada (before) (router).
8. Executa l'acció (router).
9. Executa els filtres, segona passada (after) (router).
10. Executa la vista i mostra la resposta (view).

A grans trets aquest és el procés del “front controller”. Internament és bastant més complicat però per a un programador és lo necessari per entendre com funciona en general.

4.3.3 El model

En els models resideix la lògica de negoci (o de l’aplicació). Equivocadament, molts programadors creuen que els models nomes serveixen per treballar amb les bases de dades.

Els models poden ser de molts tipus i serveixen per diferents propòsits:

- Crear miniatures d’imatges.
- Consumir i utilitzar webservices.
- Crear passarel·les Scaffold de pagament.
- Utilitzar LDAP.
- Enviar correus o consultar servidors de correu.
- Interactuar amb el sistema de fitxers local o via FTP.

4.3.4 Estructura de KumbiaPHP

L'estructura de l'aplicació en quan a carpetes és la següent:

- Aplicació
  - Core
  - Default
    - App
    - Public
    - index.php
    - htaccess

La carpeta core o nucli conté tota la lògica de KumbiaPHP. Aquesta no es pot modificar ja que faria fallar l'aplicació.

A default estaran tots els fitxers de codi que implementem per l'aplicació. A la carpeta App tindrem els controladors, models i vistes. A la carpeta Public tindrem la carpeta de css, imatges i tota la part pública de l'aplicació.

4.3.5 Partials

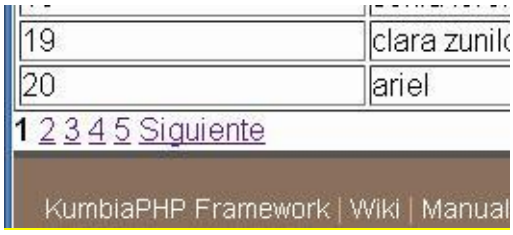
Els partials o vistes parcials son fragments de vistes que són compartits per diferents vistes, de manera que constitueixen la lògica de presentació reutilitzable en l'aplicació. En general els partials són elements com: menús, capçaleres, peus de pàgina, formularis entre d'altres.

Per construir un nou partial s'ha de crear un arxiu amb extensió.phtml en el directori views/_shared/partials/ el qual ha de correspondre amb el nom del partial.

4.3.5.1 Partials de paginació

Com a complement pel paginador d'ActiveRecord, a través de vistes parcials s'implementaran els tipus de paginació mes comuns. Aquests es troben al directori "core/views/partials/paginators" llestos per ser utilitzats. Són completament configurables via CSS. També existeix la possibilitat de crear partials propis per paginar vistes.

Mostrarem un d'ells que és el d'estil clàssic.



IL·LUSTRACIÓ 4-I

4.3.6 Templates

Els templates constitueixen la capa més externa de la vista que es mostrarà després d'executar una acció del controlador, de manera que permet establir el format de presentació apropiat per la vista.

Quan parlem de format no ens referim únicament al tipus de document, sinó també a elements com capçaleres o menús. Per tant, el template està compost per aquells elements que en conjunt són utilitzats per la presentació de diverses vistes, donant d'aquesta manera un format de presentació reutilitzable.

4.3.6.1 Creació d'un template

Per construir un nou template s'ha de crear un fitxer nou amb extensió .phtml en el directori "*views/_shared/templates/*".

El manegador de vistes de KumbiaPHP utilitza el patró de disseny de "vista en dos passos". En primer lloc es processa la vista d'acció i s'emmagatzema al buffer de sortida i en segon pas es processa el template. En conseqüència, com la vista d'acció processada s'acumula en el buffer de sortia és necessari invocar el mètode View::content() en el lloc on es desitja mostrar la vista.

4.3.7 Helpers

Els helpers (ajudes) s'utilitzen a les views. Encapsulen codi de mètodes per poder ésser utilitzats varis cops sense repetir codi.

El framework ja porta de sèrie helpers creats. Encara que lo realment útil és que els usuaris poden crear els seus propis helpers i col·locar-los *en "/app/extensions/helpers/".* La carga d'aquests helpers a les vistes és totalment transparent.

4.3.8 Scaffold i CRUD

Scaffold és un mètode de meta-programació per construir aplicacions de software que suportin base de dades . Aquesta és una nova tècnica suportada per alguns framework del tipus MVC, on el programador ha d'escriure una especificació que digui com ha de ser utilitzada l'aplicació de base de dades. El compilador després utilitzarà aquesta especificació per llegir, crear, actualitzar i esborrar entrades a la base de dades (el que es coneix com CRUD, l'acrònim de Create, Read, Update i Delete en anglès), tractant de posar plantilles com una bastida Scaffold, la qual esdevé una aplicació molt potent.

Scaffolding és l'evolució de codis generadors de base de dades des de ambients més desenvolupats, com és el cas de "CASE Generator" de Oracle i altres tants serveis 4GL per serveis al client.

Scaffolding es va fer popular al framework "Ruby on Rais", i posteriorment s'ha adaptat a altres frameworks com Django, Monorais, KumbiaPHP i d'altres.

KumbiaPHP té la característica de que podem fer servir un CRUD sense la necessitat d'un Scaffold. Tot seguit explicarem de forma resumida com s'hauria d'implementar aquest sistema. Tot té a veure amb que ActiveRecord ja porta implementat un CRUD.

El primer pas és crear una taula a la base de dades:

```
CREATE TABLE menus
(
id          int           unique not null auto_increment,
nombre      varchar(100),
titulo      varchar(100)   not null,
primary key(id)
)
```

Tot seguit definirem el model el qual ens permetrà interactuar amb la base de dades.

```
<?php
class Menus extends ActiveRecord
{  public function getMenus($page, $ppage=20)
  {     return $this->paginate("page: $page", "per_page: $ppage", 'order: id desc');
  }
}
```

Aquest codi va dins de "*[app]/models/menus.php".* El nom de l'arxiu php que conté el model té el mateix nom que assignem a la classe i també coincideix amb el nom de la taula. La classe "Menus" hereta totes les característiques de la classe ActiveRecord i una d'aquestes es el poder fer servir CRUD.

El controlador és l'encarregar d'atendre les peticions del client (per exemple el navegador) i a la vegada de donar resposta. En aquest controlador definirem totes les operacions CRUD que necessitem.

[app]/controllers/menus_controller.php:

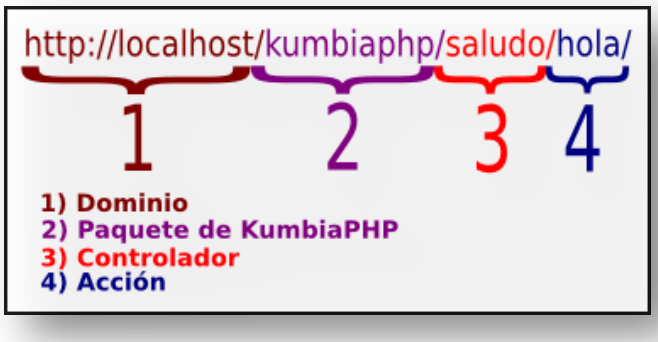
```
<?php
Load::models('menus');
class MenusController extends AppController {
  public function index($page=1)
  {
     $menu = new Menus();
     $this->listMenus = $menu->getMenus($page);
```

```
 }
public function create ()
 {
    if(Input::hasPost('menus')){
        $menu = new Menus(Input::post('menus'));
        if(!$menu->save()){
            Flash::error('Falló Operación');
        }else{
            Flash::valid('Operación exitosa');
            Input::delete();
        }
    }
 }
```

No copiarem tot el codi necessari per implementar el CRUD, tant sols el tros del principi on veiem la carga del model, la definició del controlador i dos funcions, la de index i la de crear un registre. Faltarien les de editar (Update a la base) i la de esborrar (delete).

4.4 KumbiaPHP i les seves URLs

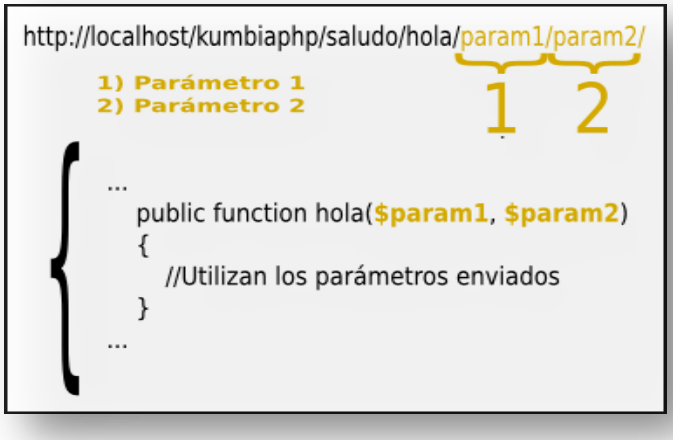
Per entendre el funcionament del framework és important entendre les seves URLs. La figura següent mostra una URL típica en KumbiaPHP.



IL·LUSTRACIÓ 4-II

Aquest framework no existeixen les extensions .php degut a que en primera instancia i ha reescriptura de URLs i també compte amb el controlador frontal que s'encarrega de rebre totes les peticions com ja vàrem parlar.

Qualsevol altre informació passada per URL es considera com un paràmetre de l'acció. La següent figura mostra dos paràmetres passats a l'acció "hola" que pertany al controlador "saludo".



IL·LUSTRACIÓ 4-III

Això és útil per evitar haver d'estar enviant paràmetres GET de la forma "?var=valor&var2=valor2" (la manera de sempre de PHP), la qual revela informació sobre l'arquitectura de software que disposa el servidor. A més, fan que les URL siguin més llargues i complicades d'entendre.

4.5 Les accions

Les accions són la part fonamental de l'aplicació, posat que contenen el flux en que l'aplicació actuarà envers certes peticions. Les accions utilitzen el model i defineixen variables per la vista. Quan es realitza una petició web en una aplicació KumbiaPHP, la URL defineix l'acció i els paràmetres de la petició.

Les accions són mètodes d'una classe controladora que es diu "ClassController" que hereta de la classe AppController i poden o no ser agrupades per mòduls.

4.5.1 Les accions i les vistes

Cada cop que s'executa una acció el framework buscarà després una vista (view) amb el mateix nom de l'acció. Aquest comportament s'ha definit per defecte. Normalment les peticions han de donar resposta al client que les sol·licita, així doncs si tenim una acció amb una crida que es diu *Comprar()* hauria d'existir una vista associada a aquesta acció que es tingui per nom "*comprar.phtml*".

4.6 ActiveRecord

És la principal classe per la administració i funcionament dels models. ActiveRecord és una implementació d'aquest patró de programació i esta molt influenciada per la funcionalitat de la seva anàloga en "Ruby on Rails".

ActiveRecord proporciona la capa objecte-relacional que segueix rigorosament el estàndard ORM: Taules en classes, registres en objectes i camps en atributs. Facilita la comprensió del codi associat a base de dades i encapsula la lògica especifica fent-la fàcil d'utilitzar pel programador.

4.6.1 Avantatges del patró ActiveRecord

Les avantatges principals d'aquest patró són les següents:

- Es treballa amb entitats del model de forma més natural com objectes.
- Les accions com inserta, consulta, actualitza, borra, etc, d'una entitat del model estan encapsulades així que es redueix el codi i es fa més fàcil de mantenir.
- Codi més senzill de entendre i mantenir.
- Reducció del ús del SQL en un 80%, així tenim una alt grau de independència del motor de la base de dades.
- ActiveRecord protegeix en un gran percentatge d'atacs "SQL inyection" que poden arribar a sofrir les aplicacions.

4.6.2 Les columnes i atributs

Els objectes ActiveRecord corresponen a registres d'una taula de una base de dades. Els objectes tenen atributs que corresponen al camps d'aquestes taules. La classe ActiveRecord automàticament obté la definició dels camps de les taules i els converteix en atributs de la classe associada. Això és el que ens referim amb mapeig objecte relacional.

5 ANÀLISIS

Un cop tenim tots els fonaments teòrics clars hem passat a fer un anàlisis de com implementarem la nostra aplicació.

Aquesta ha de mantenir una estructura senzilla i complir els objectius que ens hem proposat inicialment.

Primer de tot per fer funcionar KumbiaPHP hem d'instal·lar PHP5 i un servidor web. Aquest servidor serà Apache ja que és codi lliure i no té cap cost associat. També haurem d'instal·lar MySQL com a base de dades. Hem analitzat de quina forma és més senzill instal·lar tot aquests serveis necessaris i s'ha arribat a al conclusió de que el millor és fer servir un paquet anomenat XAMPP que porta PHP5, Apache server i MySQL. A més té un centre de control per reiniciar, parar o iniciar els serveis. Així podem fer que la configuració de tot lo necessari sigui més senzill i amigable i un cop donem el projecte al client final tindrà un millor control sobre totes les eines que intervenen a l'aplicació.

Un cop que funcioni PHP, el servidor web i MySQL instal·larem el projecte de KumbiaPHP per l'aplicació. Haurem de mirar com configurem tots els arxius que porta KumbiaPHP, com per exemple la comunicació entre l'aplicació i la base de dades.

S'ha analitzat a consciència quina és l'estructura que tindrà la nostra aplicació i la conclusió és que necessitem uns espais i dins aquests espais els usuaris finals faran la reserva. De moment no es posarà cap restricció sobre el numero de reserves a un espai. També s'ha pensat que necessitem una entitat que englobi els diferents tipus d'espai. Així doncs si tenim dos pàrquings a l'empresa, podrem crear dos entitats que formaran aquests dos pàrquings i dins d'aquestes entitats trobarem els espais, és a dir, places de pàrquing que formen cada entitat. Les reserves es faran a dins de cada espai i podrem crear varies reserves.

Hem analitzat també que els usuaris finals de l'empresa ja tenen un usuari a la seva empresa i per tal de no duplicar informació i fer aprendre un altre usuari amb contrasenya s'utilitzarà el que tenen a l'empresa. D'aquesta manera no crearem un registre d'usuaris i només es podran crear des de la base de dades per l'administrador. S'ha pres aquesta decisió perquè el client ens ha manifestat que els seus treballadors prefereixen utilitzar el mateix usuari i no haver de crear un altre per aquesta aplicació. Més endavant i com a actualització es podria crear un formulari senzill per crear usuaris nous i poder modificar la contrasenya d'aquests.

L'estructura de l'aplicació serà senzill, de moment no necessitem grans quantitats de informació, opcions o pantalles. El nostre objectiu és fer una aplicació funcional i que sigui senzilla d'utilitzar. S'ha pensat que podríem tenir una portada on hi hagés una petita explicació de perquè serveix l'aplicació i a quina empresa pertany. Tindrem un formulari d'accés via login que ens portarà a les entitats. Allà podrem escollir quina entitat volem fer la reserva i ens mostrarà els espais disponibles a aquella entitat. Un cop hem triat l'espai desitjat accedirem a una pantalla nova de reserves on podrem crear la nostra reserva i veure quines reserves té fetes aquest espai.

La protecció de dades i la política de cookies no s'implementaran donat que l'entorn d'empesa no ho requereix i és un entorn tancat i segur. Més endavant es podria implementar com a actualització.

6 IMPLEMENTACIÓ

El primer pas en la implementació serà la instal·lació de tot el que necessitem perquè funcioni el projecte de KumbiaPHP. Aquest primer pas es tenir un servidor web (hem triat Apache) i el llenguatge de programació PHP5. També hem de fer l'instal·lació de la base de dades MySQL.

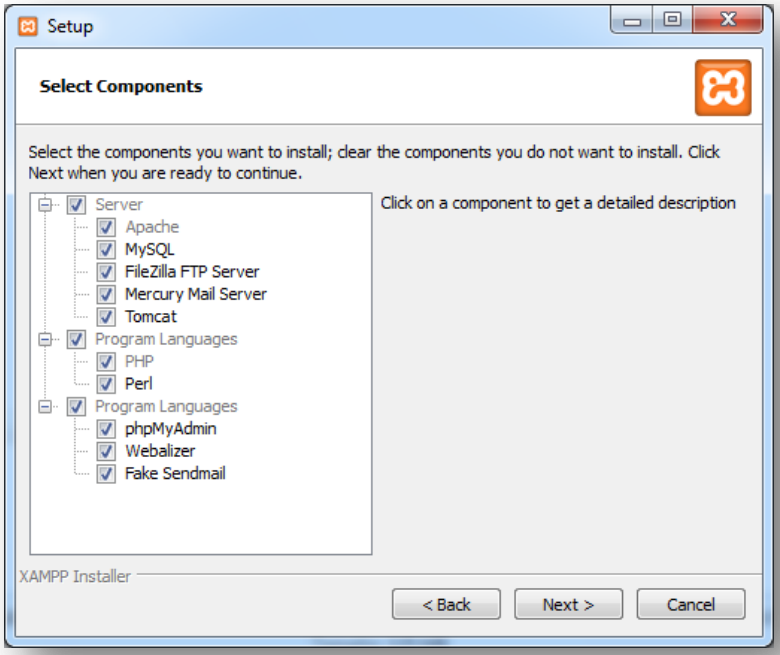
En aquesta fase hem escollit un paquet anomenat XAMPP que porta integrat Apache Server, MySQL i PHP5.

6.1 Instal·lació de XAMPP i configuració

Descarreguem el paquet d'instal·lació de la pagina web de Apache Friends. Triarem la versió per a Windows. Hem triat la versió 1.8.3 que porta PHP versió 5.5.15. Accedim a la següent web:

- *https://www.apachefriends.org/es/index.html*

Aquest paquet porta més programes de sèrie com poden ser FileZilla o un Tomcat. Els instal·larem tots per si els necessitéssim més tard.



IL·LUSTRACIÓ 6-I

Mentre estem instal·lant el paquet ens preguntarà si volem Apache i MySQL com a servei. Ara mateix per implementar l'aplicació no hem marcat aquest opció ja que iniciarem els serveis nosaltres mateixos. Però quan ens toqui instal·lar el paquet al servidor del client és important que ho fem com a serveis, així només iniciar els servidor es carregaran aquests serveis i tindrem l'aplicació funcionant.

El següent pas és editar l'arxiu "*C:\xampp\apache\conf\httpd.conf"* i treure una línia com a comentari (s'ha de esborrar el símbol #) de la línia on diu:

- *LoadModule rewrite_module modules/mod_rewrite.so*

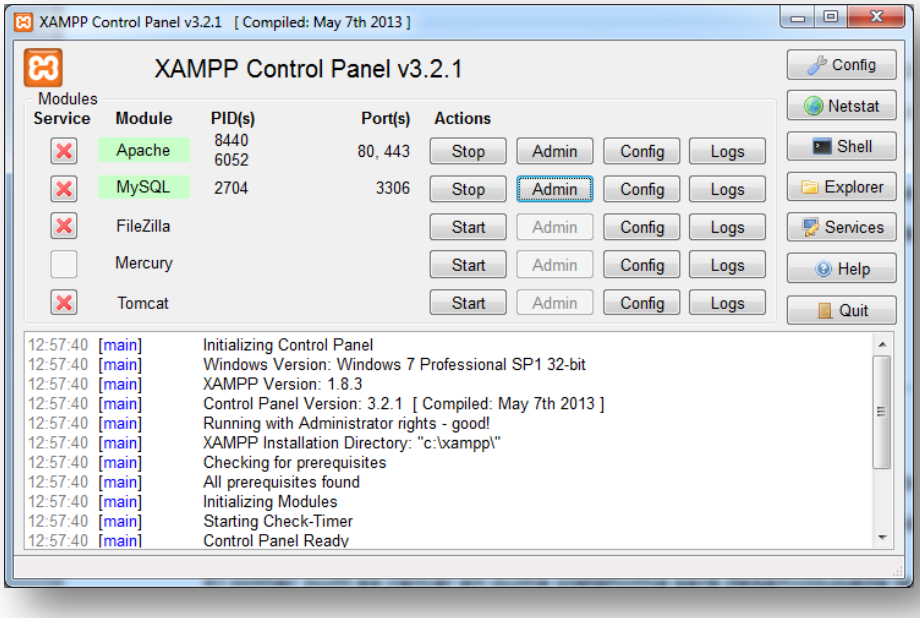
Necessitem activar el mòdul "Rewrite" d'Apache ja que permet reescriure a KumbiaPHP les URL tal i com hem explicat als foments teòrics. Per refrescar la memòria direm que aquest framework encapsula les URL per mostrar-les de forma amigable i senzilla. A

més aquest mòdul també pot protegir les nostres aplicacions davant una possibilitat de que els usuaris pugin veure els directoris del projecte i pugin accedir a arxius de classes, models, lògica etc, sense ser autoritzats. D'aquesta manera el usuaris finals només poden accedir al contingut el directori públic del servidor web, tot lo altre queda amagat.

Un cop hem realitzat aquesta tasca i hem guardat l'arxiu podem reiniciar els serveis des del centre de control de XAMPP. Amb aquests senzills passos ja tenim preparat tot lo necessari per instal·lar el paquet de KumbiaPHP. Aquest paquet l'haurem de copiar a la carpeta "*C:\xampp\htdocs*\".

Cal a dir que la instal·lació es fa tot en local ja que permet més comoditat a l'hora d'implementar l'aplicació.

Un cop feta la instal·lació tindrem accés al panell de control que té aquest aspecte:



IL·LUSTRACIÓ 6-II

Des d'aquí podrem arrencar tots els serveis, accedir als seus panells d'administració, configurar els mòduls, veure "Logs" i altres prestacions que podem veure a la imatge.

6.2 Instal·lació de KumbiaPHP i configuració

Per descarregar el projecte inicial de KumbiaPHP anirem a la seva pàgina oficial (http://www.kumbiaphp.com/) i descarregarem el paquet de "KumbiaPHP 1.0 Spirit Beta2". És la ultima versió que està disponible. La instal·lació d'aquest paquet és molt senzilla, només s'ha de descomprimir a la carpeta "htdocs" i ja podem començar a configurar la nostra aplicació.

El primer pas de tot és canviar el nom de la carpeta principal ja que serà el nom que tindrà la URL quan accedim a l'aplicació. Ja que la nostra es diu "GSAI" , que prové de "Gestió d'espais", canviarem el nom de la carpeta per "gsai" i quedarà la carpeta de la següent forma:

- *C:\xampp\htdocs\gsai*

Per accedir a la nostra aplicació des del navegador utilitzarem l'adreça:

- *http://localhost/gestioespais/*

Ara anem a parlar de com és l'estructura dels directoris a la nostra aplicació.

6.2.1 Estructura dels directoris

La estructura del framework a nivell de carpetes quan s'instal·la per primer cop és bastant senzill i ordenat. Les carpetes principals són "core", "default" i "vendor". El codi que anem incorporant dels controladors, vistes i models va tot a la carpeta "default" a dins de cada carpeta que porta ja el seu nom.

El nom de la carpeta general que conté "core", "default" i "vendor" serà el nom de la nostra aplicació. En el nostre cas "gsai".

La carpeta "core" conté tota la lògica de KumbiaPHP i no hem de tocar res ja que podríem desestabilitzar l'aplicació. Tot seguit mostrem la organització de les carpeta:

- Core
  - Console
  - Extensions
    - Helpers
    - Scaffolds
  - Kumbia
  - Libs
  - Tests
  - Vendors
  - Views
    - Errors
    - Partials
    - Templates

La carpeta "default" es divideix en dos carpetes, "app" i "públic". A "app" és on crearem tots els arxius necessaris pels controladors, vistes i models. També si es dona el cas de incorporar llibreries noves anirien a aquesta carpeta, on podem observar que tenim una

carpeta dedicada a les llibreries (Libs). Aquí també emmagatzemem templates i extensions que puguem necessitar. La carpeta “públic” conté els javascripts, imatges (Img) i una carpeta de css on tenim tota la configuració de css de l’aplicació. Mostrem un exemple de com s’organitza aquesta carpeta:

- Default
    - App
        - Config
        - Controllers
        - Extensions
        - Libs
        - Locale
        - Models
        - Template
        - Views
    - Públic
        - Css
        - Files
        - Img
        - Javascript
        - Temp

L’estructura bàsica dels directoris no es pot canviar ja que ve definida d’aquesta forma i és important que mantingui l’estructura pel correcte funcionament.

6.2.2 Configuració bàsica d'arxius a KumbiaPHP

A KumbiaPHP hem de fer una petita configuració dels arxius principals que permeten que l'aplicació sàpiga quina base de dades ha de fer servir i en quin estat es troba el projecte (desenvolupament o producció). Aquests es troben a la carpeta de:

- *C:\xampp\htdocs\gsai\default\app\config*

A "config.ini" escriurem el nom de l'aplicació, en quina fase horària ens trobem, en el nostre cas serà "Europe/Madrid". També si l'aplicació està en producció, quina base de dades utilitzarem (desenvolupament o producció), el tipus de data a la base de dades, si activem el modo debug o les rutes.

```
[application]
name = "Gsai GESTIÓ D'ESPAIS"
timezone = "Europe/Madrid"
production = Off
database = development
dbdate = YYYY-MM-DD
debug = On
log_exceptions = On
charset = UTF-8
cache_driver = file
metadata_lifetime = "+1 year"
namespace_auth = "default"
;locale = es_ES
routes = On
```

També existeix un arxiu anomenat "databases.ini" que porta tota la configuració de la comunicació entre l'aplicació i la base de dades. Mostrem un petit exemple de com quedaria configurada la part de desenvolupament.

```
[development]
host = localhost
username = gestio
password = 123456
name = gestioespais
type = mysqli
charset = utf8
```

6.3 Bases de dades MySQL

La base de dades que s'ha instal·lat des de XAMPP és MySQL i s'administra des de un panel de control anomenat "phpMyAdmin".

La primera configuració que farem és crear un usuari que tingui permisos de lectura i escriptura. Com hem vist abans als arxius de configuració de KumbiaPHP l'usuari de desenvolupament serà "gestio". Per tant crearem aquest usuari a la base de dades amb els permisos necessaris. Ara que estem implementant l'aplicació creem aquest usuari amb tots els permisos per poder treballar de forma còmode, però un cop passem a producció haurem de crear un usuari nou i donar accés a tots els privilegis menys el de crear i esborrar taules.

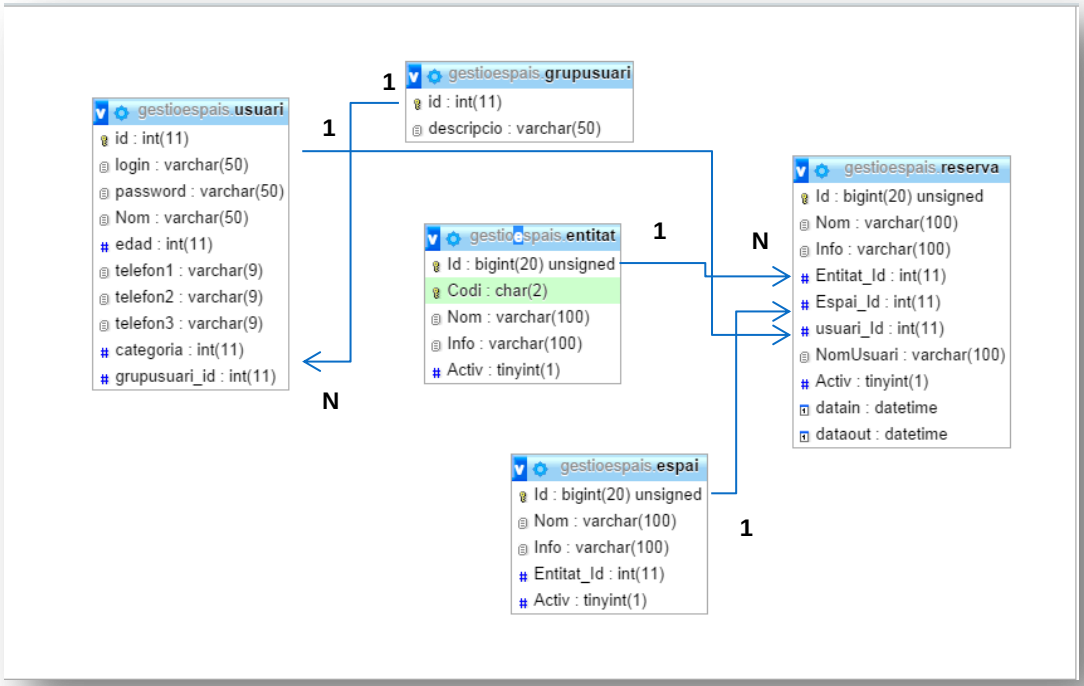
Un cop hem creat l'usuari que utilitzarà l'aplicació per accedir i fer els canvis necessaris a la base de dades toca crear una base de dades que contindrà totes les taules i registres de l'aplicació. L'anomenarem "gestioespais". Aquí dins crearem mitjançant crides SQL les taules que formaran part de l'aplicació.

Les taules principals seran entitat, espai, reserva, usuari, grupsuari. Mostrem com a exemple la taula usuari:

```
CREATE TABLE usuari (
id INTEGER NOT NULL AUTO_INCREMENT,
login VARCHAR(50) NOT NULL,
password VARCHAR(50) NOT NULL,
edad INTEGER NOT NULL,
telefon1 VARCHAR(9) NOT NULL,
telefon2 VARCHAR(9) NOT NULL,
telefon3 VARCHAR(9) NOT NULL,
categoria INTEGER NOT NULL,
grupusuari_id int NOT NULL,
PRIMARY KEY(id)
)
```

Les taules d'entitat, espai i reserva estaran relacionades a través del seus "id". Hem creat una taula de "grupusuari" per una futura actualització en la que podem distingir entre usuaris.

Mostrem un esquema de les relacions entre les taules creades:



IL·LUSTRACIÓ 6-III

Finalment volem esmentar que les taules d'entitat i espai han estat omplertes amb els registres de l'empresa que ens han demanat. S'han creat pàrquings, despatxos i sales de conferències.

Cada espai té una reserva feta que no es mostra a l'usuari perquè el sistema funcioni correctament.

6.4 Models

A la carpeta models tindrem els models que enllaçaran les entitats, espais, reserves i usuaris implementant ActiveRecord per poder accedir als registres de la base de dades.

Els models que hem implementat són els següents:

6.4.1 Entitat

S'encarrega d'administrar la lògica de negoci i permet accedir a les dades de la base de dades que pertanyen a les entitats gracies a que implementa ActiveRecord.

Té dos funcions, la primera s'encarrega de passar tota la entitat al controlador, i l'altre retorna el nom de l'entitat.

A més aquí configurem quants elements veurem a les vistes d'entitats. Hem seleccionat cinc elements per veure a cada vista.

```
<?php
class Entitat extends ActiveRecord {
	public function getEntitat($page, $ppage=5)
	{
		return $this->paginate("page: $page", "per_page: $ppage", 'order: Nom asc');
	}
	public function getNom($id)
	{
		return $this->find($id)->Nom;
	}
}
?>
```

6.4.2 Espai

Implementa i hereta de la classe ActiveRecord tota la funcionalitat per accedir a la base de dades i té dos funcions per passar dades entre espai/entitat i espai/reserva.

Aquí també implementen la funció que passarà quants elements tindrem a la vista d'espais.

6.4.3 Reserva

Com els altres models anteriors la reserva implementa la classe ActiveRecord per poder accedir fàcilment a les dades de la base de dades de reserva i s'implementen tres funcions. La primera retorna tots els atributs de la reserva i l'altre passa el nom de la reserva i la ultima envia el "id" de la reserva.

També cal remarcar que aquí el sistema de paginació de les vistes s'ha marcat en uns vista per pàgina, així la lectura de totes les dades de la reserva és mes fàcil i no desquadra l'aplicació.

6.4.4 Usuari

El model d'usuari és més senzill. Hereta de la classe ActiveRecord i no té cap funció. Simplement carga una llibreria "auth2" perquè funcioni l'autentificació d'usuaris.

```
<?php
// Carga de la libreria auth2
Load::lib('auth2');
class Usuari extends ActiveRecord
{
}
?>
```

6.5 Controladors

A la carpeta dels controladors podem trobar tots els controladors necessaris que carregen els models i enllacen la lògica de negoci amb la presentació. Tenim un controlador que s'encarrega del accés per part dels usuaris finals a l'aplicació i també els controladors que gestionen tota la part de entitats, espais i reserves. Es pot observar que els noms d'arxius tenen la peculiaritat de que acaben en "_controller". Tal i com vam explicar això és necessari perquè KumbiaPHP interpreti correctament les funcions que tindrem dins. Farem una petita explicació de cadascun.

6.5.1 Principal

Aquest controlador hereta de la classe “AppController” i conté tres funcions (accions):

- Login: S’encarrega de gestionar les entrades quan un usuari s’identifica al sistema.
- Index: Quan un usuari final ja s’ha identificat aquesta funció mostra l’aprovació de que tot ha anat bé. També passa com a variable el nom de l’usuari identificat i finalment el redirigeix a la vista d’entitats.
- Logout: Aquesta acció destrueix l’usuari identificat i redirecciona al usuari a la pagina principal de l’aplicació.

6.5.2 Entitat

Aquest controlador el trobem sota el nom de “ConsultarEntitat” hereta de la classe “AppController” i carrega el model d’entitat.

També té la capacitat de canviar el template que necessitem un cop ens hem autentificat al sistema i així el podem visualitzar a la vista. Aquest template es carrega dins una funció protegida anomenada “before_filter” que està definida pel framework i es carrega abans de totes les funcions del controlador.

A més a més s’encarrega de llençar una funció que fa funcionar el sistema de paginat de la vista. Aquesta funció envia a la vista com a paràmetres el numero d’elements (entitats) que volem visualitzar a l’aplicació i quins són.

6.5.3 Espai

Aquest controlador el trobem sota el nom de “ConsultarEspai” i hereta de la classe “AppController” Carrega els models d’entitat i espais. Ja que els espais estan continguts dins d’una entitat crea variables de tipus entitat i espai i les envia a la vista per poder mostrar correctament les relacions.

També té la capacitat de canviar el template que necessitem un cop ens hem autentificat al sistema i així el podem visualitzar a la vista.

6.5.4 Reserva

Per a les reserves tenim dos controladors.

El primer s’encarrega de mostrar totes les reserves d’un espai i té el nom de “ConsultarReserva”. Aquest implementa tres funcions. La funció “index” permet passar l’estructura de reserva i espai per tal de veure a la vista els registres de reserves fetes. També té dos funcions (“create” i “del”) que serveixen per crear i esborrar una reserva.

L’altre controlador anomenat “ConsultarReservaEntitat” ens permet mostrar quines reserves hi ha fetes a una entitat. Aquest es llança a la vista d’entitats.

6.6 Vistes

Les vistes les trobem a dins de la carpeta “views”. Són arxius que tenen l’extensió “.phtml” i s’encarreguen de mostrar quina informació o formularis tenim a cada part de l’aplicació.

A la carpeta de vistes podem veure que s’organitzen amb subcarpetes que prenen el nom dels controladors associats i dins d’aquestes carpetes tenim els arxius de cada vista.

Pràcticament totes les vistes que s’han creat es poden englobar en dos tipus.

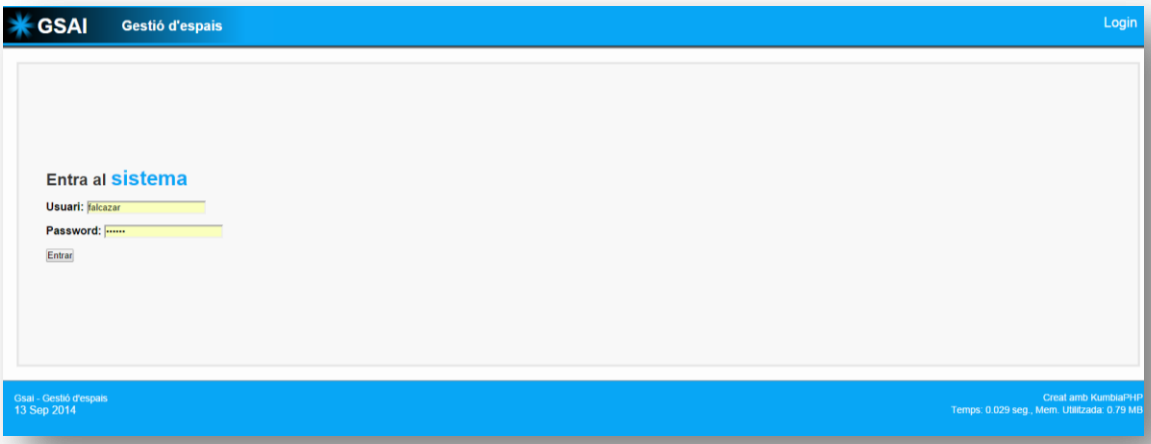
- Formularis
- Visualitzen informació.

Passem a descriure les característiques principals de cada vista.

6.6.1 Principal – Login

Tenim dos vistes a la carpeta de “/principal”, la de login i logout.

Aquestes tenen com a codi font uns formularis que s’encarreguen de mostrar les típiques capces on introduirem el nom i la contrasenya per autentificar-nos al sistema.

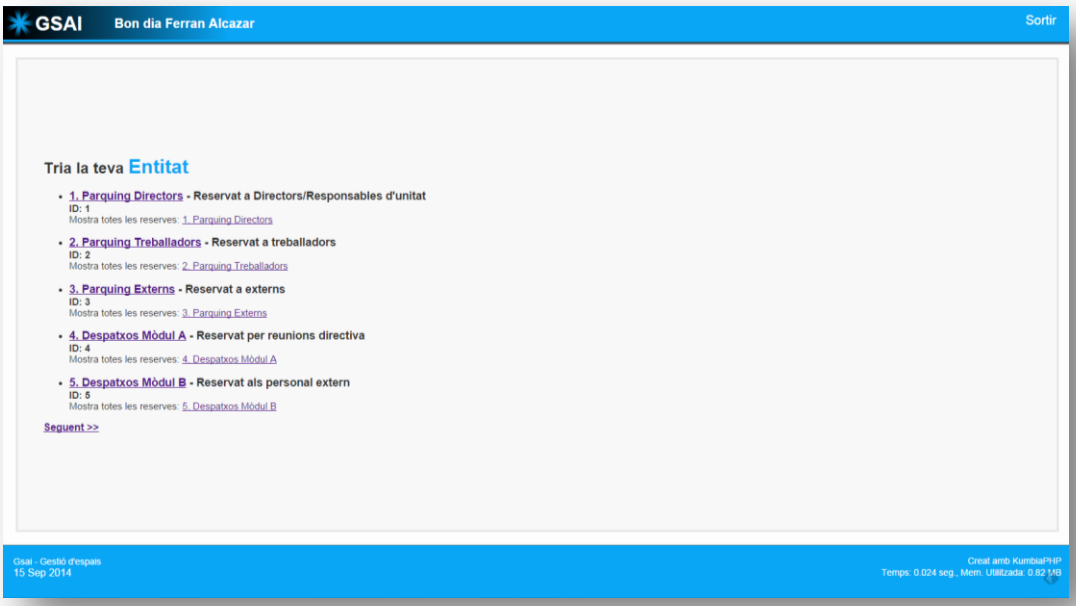


IL·LUSTRACIÓ 6-IV

6.6.2 Entitats

Aquesta vista prové del controlador “ConsultarEntitat” i conté un arxiu index.phtml que s’encarrega de ensenyar la informació relacionada amb totes les entitats que tenim. A més aquestes entitats ja són un propi enllaç web per entrar a cada entitat. Hem inclòs informació relacionada amb les entitats que a l’usuari final li pot interessar.

Finalment existeix un enllaç que ens mostra totes les reserves fetes a aquesta entitat.

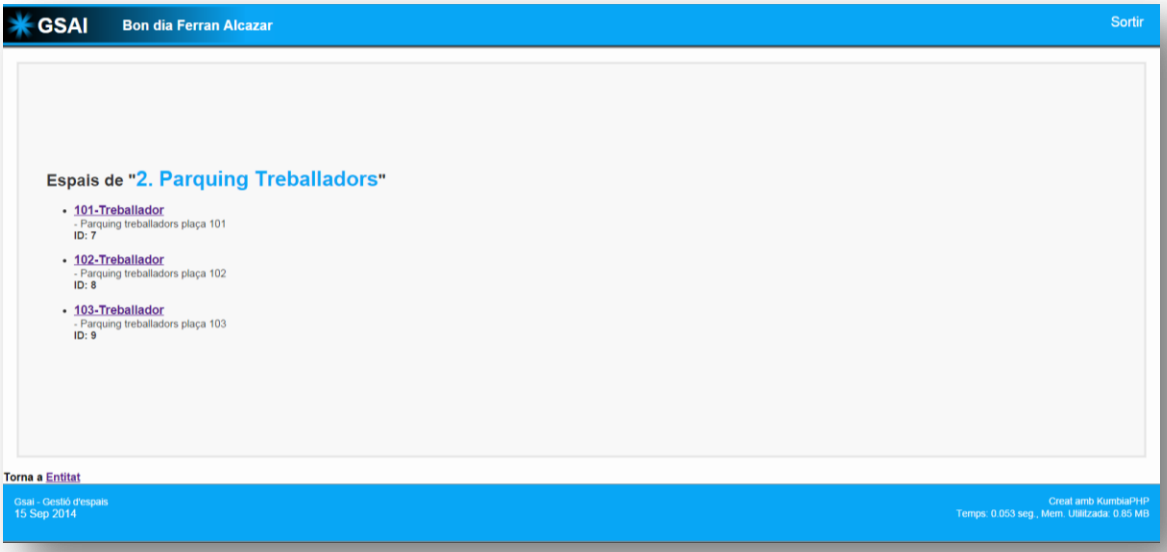


IL·LUSTRACIÓ 6-V

6.6.3 Espais

Aquesta vista és molt semblant a la d'entitats. Hereta el nom del controlador "ConsultarEspai" i només té un arxiu "index.phtml". En aquest arxiu ve el codi que fa que es mostrin tots els espais relacionats amb la entitat que acabem de seleccionar. Cada espai mostrat és un propi enllaç per entrar a cada espai i poder fer una reserva.

A més es mostra informació relacionada amb els espais.

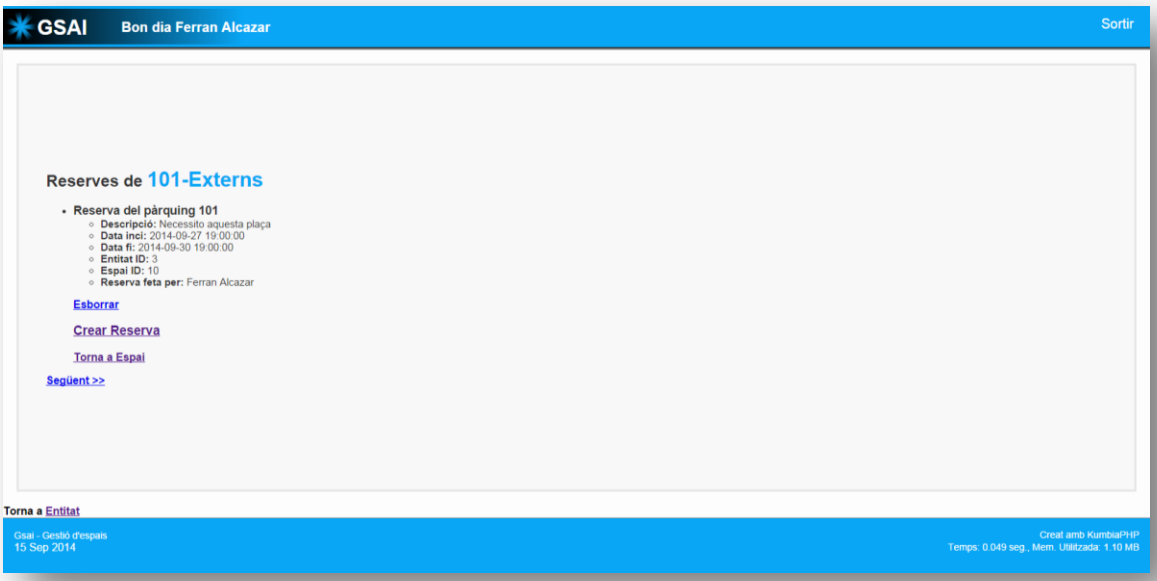


IL·LUSTRACIÓ 6-VI

6.6.4 Reserves

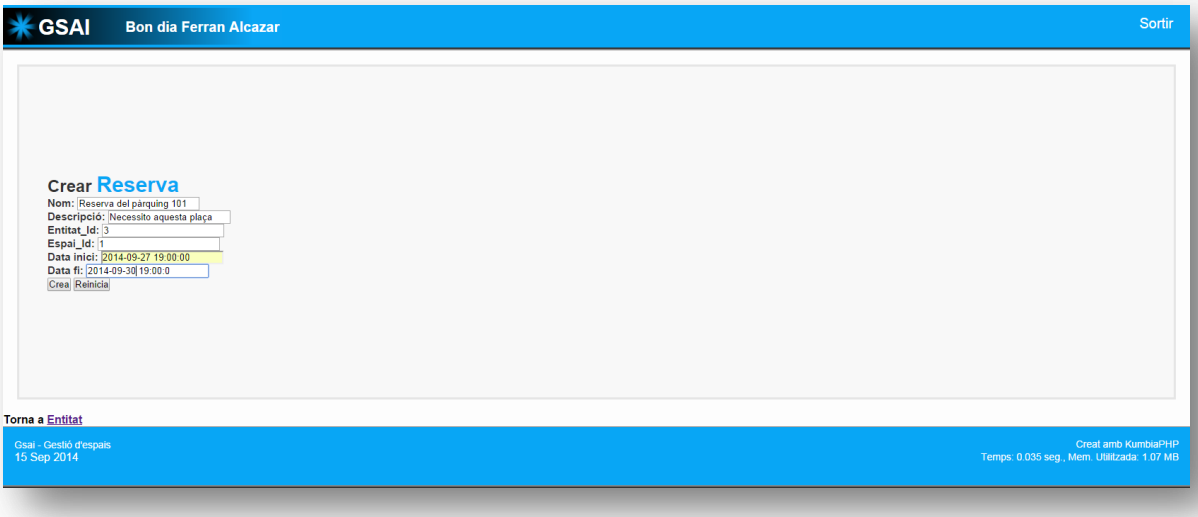
A reserves tenim varies vistes i les trobem a les carpetes que hereten el nom dels controladors que hem definit abans.

El primer grup de vistes les trobem a “ConsultarReserva”. Per una part tenim la vista que mostra les reserves fetes a aquell espai. Aquí es mostra un registre per pàgina de reserva, un enllaç per crear una reserva nova i un per esborrar la reserva actual.



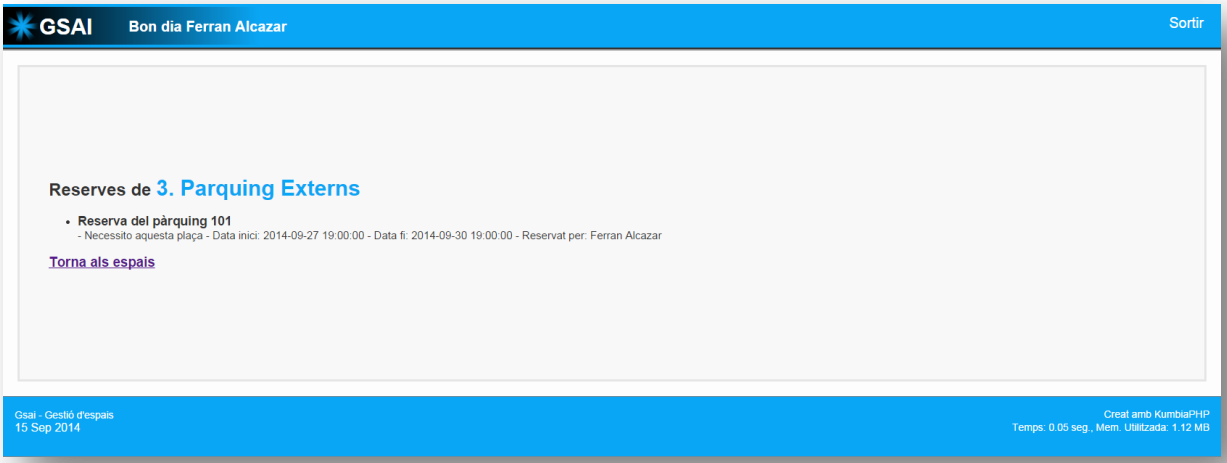
IL·LUSTRACIÓ 6-VII

Per l'altre part tenim una vista més relacionada amb la creació de reserves. Aquesta es criden des de la vista "index" de la que hem estat parlant abans. La vista amb nom "create" té un formulari amb tots els camps per crear una reserva.



IL·LUSTRACIÓ 6-VIII

També tenim la vista on es veuen totes les reserves fetes a una entitat. Aquesta pren el nom de “ConsultarReservaEntitat”. S’accedeix a ella a través de la vista d’entitats.



IL·LUSTRACIÓ 6-IX

6.7 Partials

Els partials com vam veure als fonaments teòrics serveixen per crear vistes parcials.

Els partials els podem trobar al següent directori:

C:\xampp\htdocs\gsai\default\app\views_shared\partials

A la nostra aplicació tenim definits només un partials. Aquest ens acompanya a totes les pàgines de l’aplicació i és la part del final de la pàgina web. Hem anomenat aquest partial com a “footer.phtml”. L’únic que fa és mostrar informació del dia que estem, el temps utilitzat per carregar la pàgina i un enllaç a la pàgina principal. Aquest el fem servir per tancar el disseny de l’aplicació web.

6.8 Templates

Els templates de la nostra aplicació són dos principalment. Els dos són molt semblants ja que un va néixer com a copia del primer.

El template principal ens mostra la configuració de la pàgina amb una capçalera on trobem el títol de l'aplicació i un accés per fer el login d'usuari. A la part central de l'aplicació mostrarem tot el contingut. Quan accedim a la plana principal trobarem al centre el contingut de benvinguda. Un cop fem login i entrem a les diferents pantalles el contingut mostrarà el que ens doni la vista del que estem visitant.

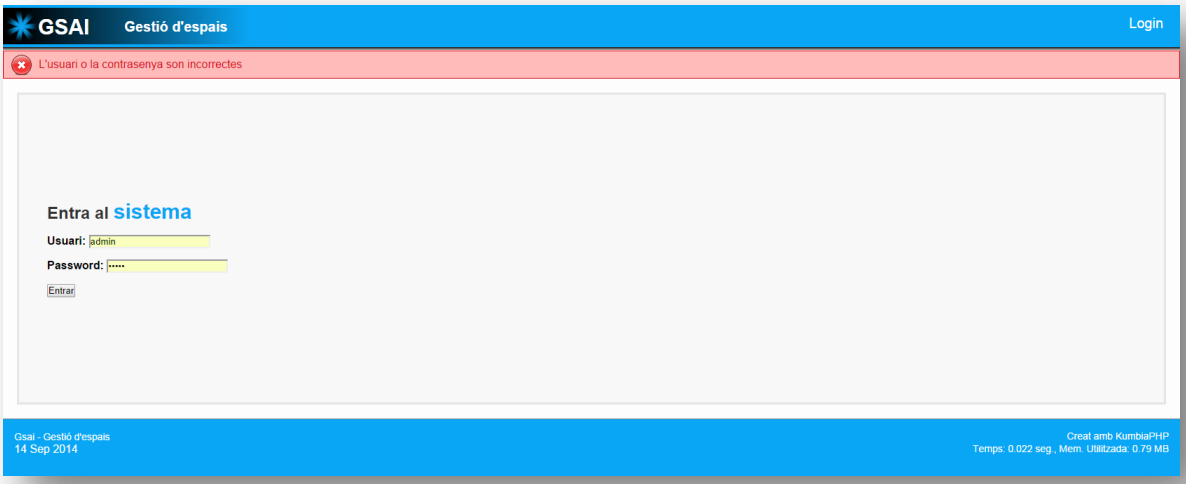
El segon template és idèntic al primer l'únic que canvia són certes parts de la capçalera on podrem veure ara un accés per sortir de l'aplicació i una frase de benvinguda amb el nom de l'usuari.

IL·LUSTRACIÓ 6-X

7 PROVES

El joc de proves que hem portat a terme consisteix a entrar a totes les pàgines de l'aplicació i provar tots els enllaços possibles.

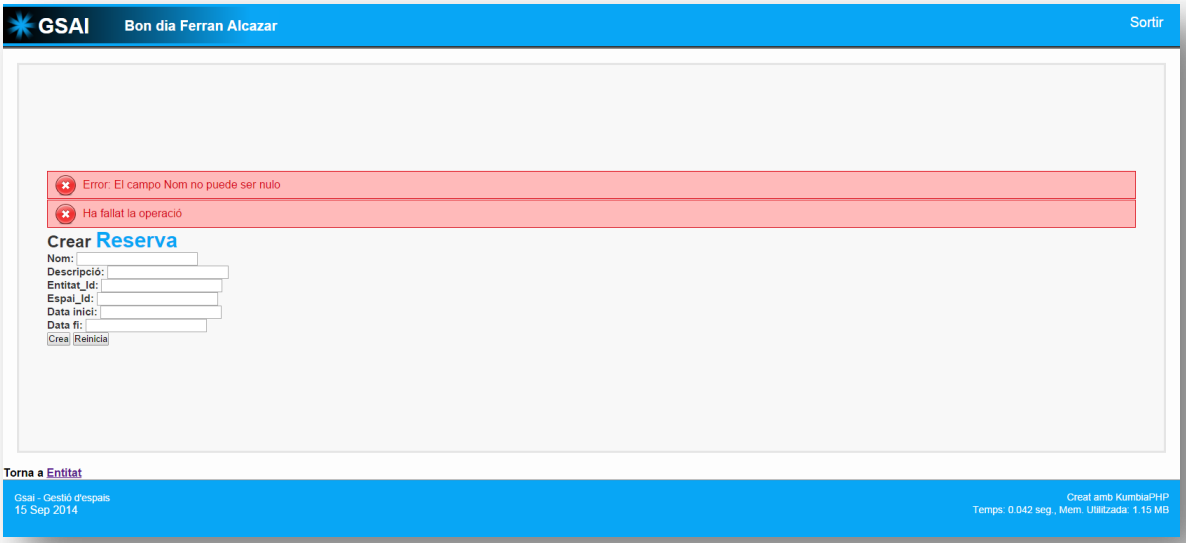
Cal a dir que KumbiaPHP incorpora missatges d'error i a la nostra aplicació s'han implementat per tal de mostrar-los quan és necessari. Per exemple aquí podem veure un missatge d'error al intentar accedir al login amb un usuari incorrecte.



IL·LUSTRACIÓ 7-I

Si accedim de forma correcta també mostra un missatge de color verd confirmant que hem entrat a l'aplicació correctament.

Als formularis de creació de reserves també es comprova que tots els camps estiguin omplerts. El que no tenim és una comprovació de que les dades siguin consistents (per exemple les dates). Aquest tema es podria implementar com a actualització.



IL·LUSTRACIÓ 7-II

Un cop hem analitzat tots els missatges d'error i confirmació hem provat d'accedir a les totes les pagines que tenim implementades i hem comprovat que no ha sortit cap error. La relació de vistes visitades a l'aplicació amb un resultat satisfactori és el següent:

- Login i logout del sistema.
- Visualització d'entitats amb la seva informació. Es mostren 5 entitats per pàgina.
- Visualització d'espais amb la seva informació. Es mostren 5 espais per pàgina.
- Visualització de reserves amb la seva informació. Es mostra una reserva per pàgina.
- Visualització de 25 reserves a cada entitat.
- El sistema de paginació funciona a les entitats, espais i reserves.
- Donar d'alta entitats i espais via base de dades funciona de les dues formes possibles, via SQL i mitjançant el formulari de phpMyAdmin.

Com a conclusió al joc de proves podem afirmar que tot el codi implementat no té errors i funciona tot correctament.

8 CONCLUSIONS

Aquest projecte m'ha servit per poder veure tots els passos necessaris que necessita un enginyer informàtic per desenvolupar una aplicació des de zero. La importància que té la presa de requeriments al principi. Triar totes les característiques que necessitem per portar a bon port el projecte i totes les decisions (que no són poques) que he hagut de prendre per desenvolupar l'aplicació.

A nivell personal ha suposat tot un repte ja que mai havia desenvolupat cap aplicació PHP des de l'inici. Veure com una idea poc a poc creix i es converteix amb un programa que pot ajudar a la gent ha suposat una gran satisfacció. Ha estat un camí molt enriquidor i fructífer.

Aquest projecte va néixer d'una petit problema detectat a l'empresa en la qual estic treballant actualment i poc a poc ha anat evolucionant per arribar al estat en que ha quedat ara mateix. I encara que el projecte s'hagi acabat, la vida de l'aplicació només acaba de començar ja que hi ha moltes millores per implementar en un futur.

8.1 Desviacions

Finalment la nostra aplicació ha quedat acabada. Disposàvem d'un temps limitat però hem aconseguit desenvolupar tots els objectius que havíem planificat.

Algunes parts del projecte ens han portat més temps del que teníem planificat i s'ha hagut de fer un esforç per poder acabar tots els objectius. Especificarem quines parts del desenvolupament s'han desviat de la nostra planificació:

- Trobar un framework adequat: Tot i que sembla una tasca fàcil el trobar el framework adequat va costar més temps del que vam planificar. Es van provar varies opcions i fins trobar KumbiaPHP va passar més temps del pensat. Es tenia planificat uns 15 dies de durada i va esdevenir en total un mes i una setmana.
- La base de dades: Aquesta tasca es va planificar perquè durés dos setmanes però es va trobar l'inconvenient de que a mida que es desenvolupava l'aplicació es va canviar el disseny varis cops. A més els noms dels camps de les taules també es va modificar durant tot el desenvolupament.

8.2 Actualitzacions

La nostra aplicació s'ha desenvolupat perquè tingui la capacitat de ser actualitzable fàcilment. Donat que el framework utilitzat es presta a aquests menesters i el codi ha quedat molt ben organitzat, les actualitzacions no haurien de ser cap problema a l'aplicació. A mida que s'ha implementat el programa han sorgit moltes idees per ampliar les funcionalitats d'aquest, però donat el temps limitat del que vaig disposar no es van arribar a implementar mai.

S'ha fet un llistat de totes les actualitzacions que serien interessants d'aplicar:

- El sistema d'autentificació és bastant senzill i la contrasenya no esta xifrada a la base de dades. Es podria ampliar aquesta tasca de forma que quedés una aplicació 100% segura.
- L'entorn en el que es mourà l'aplicació és de per si un entorn segur dins l'empresa i no necessita complir la política de cookies o la LOPD, però seria interessant implementar el sistema d'avís de cookies i la LOPD que marca la llei del país i que l'aplicació tingui la opció de poder ser utilitzada en un entorn no tan segur com per exemple una escola.
- L'edició d'entitats i espais no existeix ara mateix des de l'aplicació. Aquesta funció no s'ha implementat donat que es va decidir que l'edició seria via base de dades per un administrador de l'aplicació. L'edició de noves entitats i espais a l'empresa es poc freqüent i per temps es va decidir que no s'implementaria.

9 BIBLIOGRAFIA

Qüestions relacionades amb la memòria

- Lògica de negoci:
  - *http://es.wikipedia.org/wiki/L%C3%B3gica_de_negocio*
- Especificacions de requisit del software:
  - *http://es.wikipedia.org/wiki/Especificaci%C3%B3n_de_requisitos_de_software*
- Requisit Funcional, no funcional :
  - *http://es.wikipedia.org/wiki/Requisito_funcional*
  - *http://es.wikipedia.org/wiki/Requisito_no_funcional*
- Cas d'us:
  - *http://es.wikipedia.org/wiki/Caso_de_uso*
- Estat de l'art:
  - *http://es.slideshare.net/veroketchup/estado-del-arte-9598050*

Elecció de framework

  - *http://blog.guebs.com/2013/11/11/consejos-elegir-framework-php/*
  - *http://www.genbetadev.com/frameworks/un-punado-de-frameworks-php-que-te-haran-la-vida-mas-simple*
  - *http://www.genbetadev.com/frameworks/y-6-frameworks-php-mas-que-te-haran-la-vida-mas-simple*
- Model-Vista-Controlador:
  - *http://ca.wikipedia.org/wiki/Model-View-Controller*
- XAMPP:
  - *https://www.apachefriends.org/es/index.html*

KumbiaPHP

- Pàgina web oficial:
  - *http://www.kumbiaphp.com/blog/*
- Definicions:
  - *http://wiki.kumbiaphp.com/P%C3%A1gina_Principal#Objetivos_del_framework*
  - *http://en.wikipedia.org/wiki/Scaffold_(programming)*
- Arxiu instal·lació
  - *https://github.com/KumbiaPHP/KumbiaPHP/archive/master.zip*
- Instal·lació
  - *http://wiki.kumbiaphp.com/Instalar_Kumbia#Requisitos*
- Manual KumbiaPHP
  - *https://docs.google.com/document/d/1kth1GhrmMEBK2cAMyiy_4Dw1qlJFNdXVuXajJ6nMTQg*
- API:
  - *http://www.kumbiaphp.com/api/beta2/index.html*
- Exemple “Hola mundo”:
  - *http://wiki.kumbiaphp.com/Hola_Mundo_KumbiaPHP_Framework*
- CRUD:
  - *http://wiki.kumbiaphp.com/Como_hacer_un_CRUD_en_KumbiaPHP_Framework*
- Utilitzar models:
  - *http://wiki.kumbiaphp.com/Como_Usar_los_Modelos_en_KumbiaPHP*
- Altres pàgines
  - *http://es.wikipedia.org/wiki/Principio_KISS*

MySQL

- Pàgina web oficial:
  - *http://www.mysql.com/*

HTML i CSS

- Referencies al CSS:
  - *http://librosweb.es/referencia/css/*
- Referencies a XHTML:
  - *http://librosweb.es/xhtml/index.html*
- Referencies a colors i els seus codis hexadecimals:
  - *http://html-color-codes.info/codigos-de-colores-hexadecimales/*

ÍNDEX DE FIGURES