EPS
Escola Politècnica Superior
UdG

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Desenvolupament d'una aplicació web per a l'administració i organització acadèmica

Document: Memòria

Alumne: Estefania Hernàndez Hernàndez

Director/Tutor: Ignacio Martín Campos
Departament: Informàtica i Matemàtica Aplicada
Àrea: LSI

Convocatòria (mes/any): Juny/2014

Índex

1. Introducció, motivacions, propòsits i objectius del projecte

Les escoles i les acadèmies de música, com qualsevol altre centre educatiu, necessiten disposar d'una administració i un control estricte d'allò que passa al seu centre. Podríem dir que això inclou com a tasques més bàsiques: controlar les assignatures que s'estan cursant, així com els cursos que ofereixen, els alumnes que estan matriculats al centre, els professors i el personal que intervé en el dia a dia, i també gestionar l'expedient acadèmic dels alumnes. Deixant a part les acadèmies de música, el cas es dóna en general, per qualsevol centre educatiu o universitari.

Actualment, podem trobar en el mercat diverses aplicacions ofimàtiques que s'utilitzen en els centres educatius per aquestes tasques, però quan l'interès recau en trobar una aplicació menys complexa, les opcions a escollir disminueixen considerablement.

Però també hi ha un problema addicional que han d'afrontar: els horaris. Com es poden realitzar uns horaris, tenint cura d'evitar el solapament d'hores? Com podríem aconseguir un bon aprofitament i distribució de totes les aules del centre?

Quan més es necessita una aplicació amb una gestió simple i còmode, ens trobem que existeixen pocs recursos per a dur a terme els nostres propòsits.

Es així com va sorgir la idea de fer aquesta aplicació, realitzant la fusió del més senzill dels serveis amb la gestió mínima necessària per complir tots els requeriments d'una acadèmia, unida amb un generador d'horaris realitzat amb intel·ligència artificial el qual proporcionarà un valor afegit a l'aplicació, mantenint l'objectiu de facilitar l'experiència dels professors i el personal d'una acadèmia, que podran accedir a l'aplicació des de casa a través dels seus ordinadors.

1.1. Propòsits i Objectius

La meva motivació per a realitzar aquest projecte ha estat aprendre quins passos calien fer per desenvolupar un projecte sencer, depenent només de mi mateixa.

En concret, el propòsit principal d'aquest projecte és la creació d'una aplicació web per a l'administració d'acadèmies i escoles. En aquest cas em centraré en la creació d'una aplicació web dirigida al funcionament d'una escola de música.

S'ha de remarcar que el projecte serà una Single Page Application (SPA) (Single Page Application (SPA)), amb un disseny simple, que consumeix una API REST, que em permetrà desenvolupar fàcilment altres clients com p.e apps per smartphones.

1.2. Estructura de la documentació

A continuació exposaré punt a punt els apartats més importants d'aquesta memòria juntament amb un resum del que es veurà en cada part.

1. *Introducció, motivacions, propòsit i objectius del projecte:* serveix per presentar el projecte. S'estableixen les raons per les quals es va desenvolupar el projecte i que se n'esperava obtenir.
2. *Estudi de viabilitat:* Cal justificar els paràmetres que fan possible el desenvolupament del projecte, descriure els recursos necessaris, pressupostos inicials, viabilitat tecnològica i/o econòmica, els recursos humans, etc.
3. *Metodologia*: S'exposa la metodologia emprada pel desenvolupament del sistema informàtic.
4. *Planificació*: En aquesta etapa es defineix l'estratègia seguida per arribar als objectius.
5. *Marc de treball i conceptes previs*: Aquí es descriuran els diversos aspectes relacionats amb el desenvolupament general del projecte. Aquests ajudaran a entendre millor els capítols següents.
6. *Requisits del sistema*: Es descriuran els requisits que ha de complir el sistema, tant funcionals com no funcionals. Això permetrà conèixer tot el que s'ha de fer i perquè.

7. *Estudis i decisions*: En aquest apartat s'ha de descriure la maquinària, les llibreries o programari utilitzats durant el desenvolupament del projecte, i s'ha de justificar les eleccions.
8. *Anàlisi i disseny del sistema*: L'anàlisi és un estudi de les necessitats del sistema, mentres que el disseny és una proposta de solució, la que s'ha seguit per solucionar-ho. Dins el disseny cal determinar els models físics de dades, processos, i objectes.
9. *Implementació i proves*: S'explica el procés de desenvolupament, els problemes i la solució aportada, així com el model de classes i els algoritmes més rellevants.
10. *Integració i resultats*: Es mostren els resultats obtinguts i els corresponents objectius assolits.
11. *Conclusions*: Diferenciar entre els objectius assolits i els requeriments inicials. S'expliquen les desviacions de la planificació original i es comenten els motius.
12. *Treball futur*: Possibles ampliacions, millores o treballs futurs.
13. *Bibliografia*: Referències utilitzades per desenvolupar el projecte.
14. *Annexos*

2. Estudi de viabilitat

Realitzaré aquest projecte sola, però sent supervisada per dues persones. Per una banda, el meu tutor, i per l'altra, una professora de música que m'ha ajudat en el procés de recollida de requeriments. Els costos associats han sigut inexistents, donat que tot el que s'ha utilitzat ha sigut software lliure o codi obert, disponible per a tothom. Tampoc he tingut cap cost associat a l'hora de desplegar la aplicació.

El codi d'aquest projecte també estarà disponible per a tothom, amb una llicència de codi lliure i per tant no té sentit realitzar un estudi econòmic de viabilitat.

3. Metodologia

Una metodologia és un seguit d'etapes i procediments utilitzats durant el procés de desenvolupament d'un sistema informàtic. En el cas d'aquest projecte, més que una metodologia, s'han enfocat a complir un marc de treball, amb un procés iteratiu.

Es podria considerar que s'ha seguit una procés de desenvolupament iteratiu i incremental, que no és res més que cada etapa del procés que s'explica a continuació pot incloure passos enrere degut a revisions. En concret:

1. Recollir els requeriments necessaris per realitzar el treball.
2. Si és correcte, passar al pas 3, altrament tornar al pas 1.
3. Decidir el llenguatge de programació i les eines a utilitzar.
4. Estructurar el treball en parts, seguint el diagrama de classes i el que s'ha decidit en els passos anteriors.
5. Desenvolupar la part corresponent seguint l'ordre de l'estructura del treball, i assegurant la compatibilitat amb les anteriors.
6. Fer comprovacions per tal de confirmar que el funcionament és correcte al finalitzar cada part.
7. Si al fer les comprovacions el resultat no és l'esperat, es torna al punt 6 per a realitzar els canvis oportuns en la última part desenvolupada o en les anteriors, si és necessari.
8. Si al fer les comprovacions el resultat és l'esperat, es desenvolupa la part següent tornant al punt 5, en cas que s'hagin finalitzat les parts amb les seves respectives comprovacions s'inicia el punt 10.
9. Documentar les parts acabades.
10. Revisar documentació, en cas que no es consideri correcta, tornar a redactar-la (pas 10), en cas que es consideri correcta, es considera tancat.

Per tal d'il·lustrar millor el procés, aquest seria el diagrama d'activitats associat (veure figura 1).

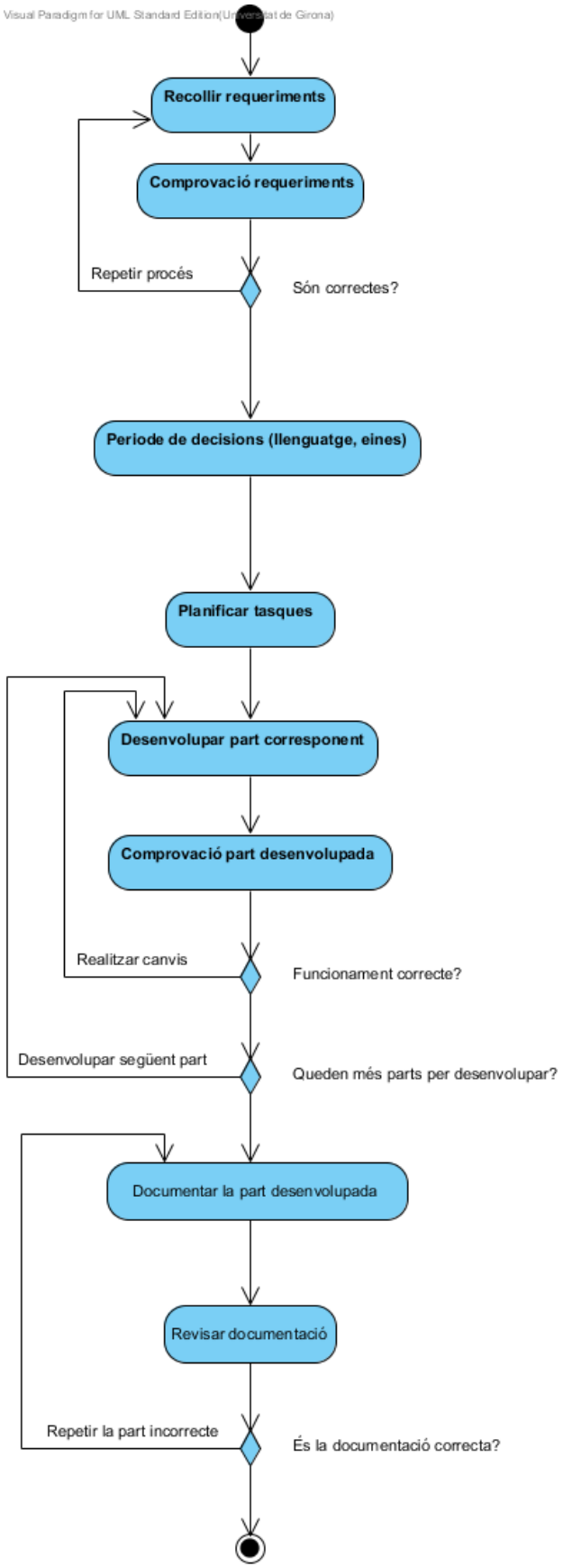


Figura 1: Diagrama d'activitats

4. Planificació

La idea d'aquest projecte va sorgir al juny del 2013, moment en el qual es va començar a parlar amb el tutor. Es va programar que l'entrega del projecte seria el juny del 2014.

Els primers mesos es van dedicar únicament a un procés iteratiu que consistia en plantejar una sèrie de requeriments, estudiar-los i revisar-los.

Just quan es va finalitzar aquesta fase, a mitjans de setembre, es va iniciar la fase de disseny, on es va realitzar un primer esbós del diagrama de casos d'us i posteriorment les fitxes de casos d'ús. Just després es va fer una primera versió del diagrama de classes.

Degut a l'inici del curs acadèmic, el projecte va quedar aturat fins l'inici del segon semestre, al febrer, moment en el qual es va reiniciar l'activitat i es va completar el diagrama de classes. Es van fer varis fins que es va triar un.

Agafant com a base l'estudi de disseny, va començar la fase de implementació. Aquesta fase es va dividir en dos tasques principals:

1. Dissenyar la API que proporcionaria les dades al client, i implementar-la.
2. Dissenyar i implementar un client que consumeixi els recursos anteriors.

Donat que el primer punt es va estimar que consumiria més temps, es va prioritzar, i un cop es va enllestir aproximadament un 75% (només quedaven petits detalls i l'algoritme genètic), es va iniciar l'etapa de programació web.

Durant aquest procés d'implementació s'ha anat realitzant la documentació que es veu recollida en aquest document, tot i que la majoria s'ha redactat més tard.

Per il·lustrar millor aquest procés, s'ha realitzat un diagrama de Gantt, amb l'eina **teamgantt**[1], que proporciona una versió de prova de 30 dies molt complerta (figura 2).

[1] Web de Teamgantt : https://teamgantt.com/

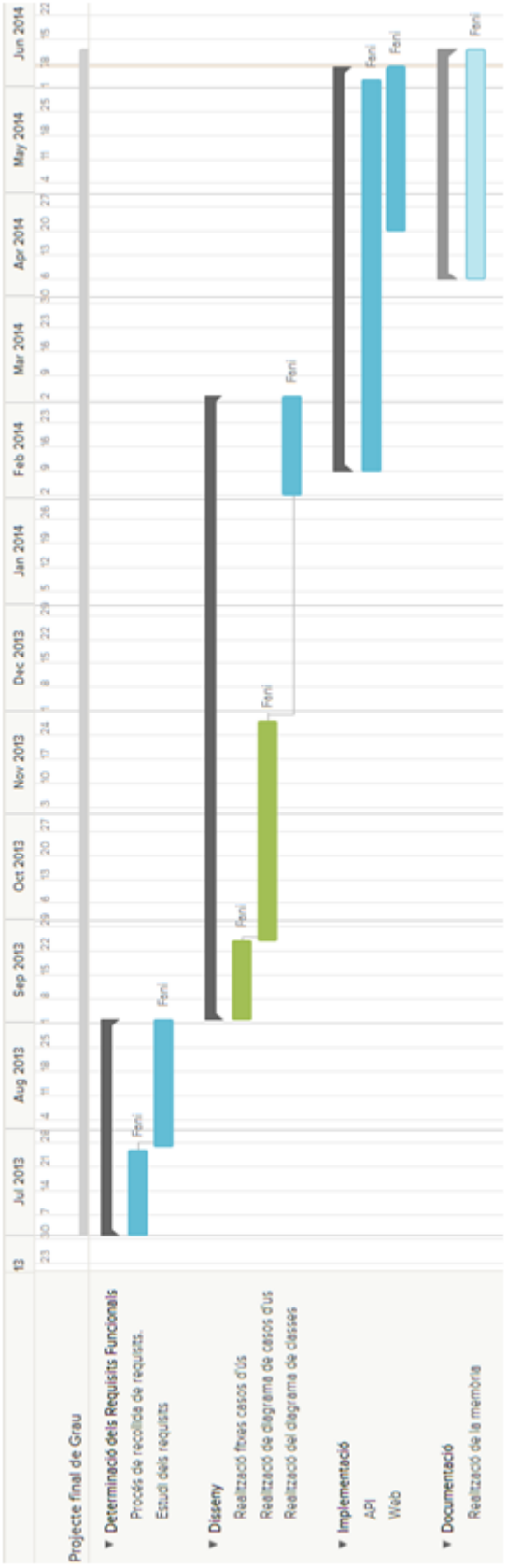


Figura 2: Diagrama de Gantt

5. Marc de treball i conceptes previs

En aquest apartat s'expliquen els conceptes necessaris per tal d'entendre els algorismes desenvolupats i les decisions presses per desenvolupar aquest projecte.

5.1. Arquitectura client – servidor

L'arquitectura client – servidor és un model d'aplicació distribuïda en el qual les tasques es reparteixen entre els proveïdors de recursos o serveis, anomenats **servidors**, i els que demanen aquests recursos, anomenats **clients**.

Un client realitza peticions a un servidor, que retorna una resposta. La separació entre el client i el servidor és lògica.

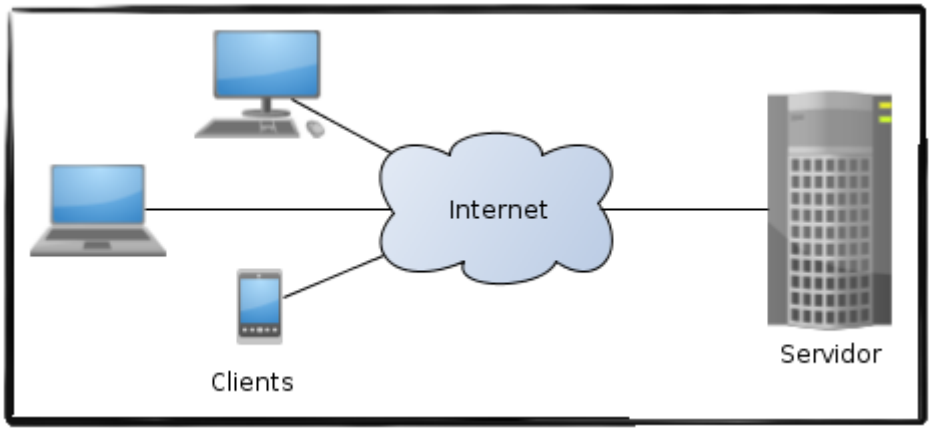


Figura 3: Exemple de client-servidor[2]

Es habitual que d'una entitat servidora se n'aprofitin una o moltes entitats clients.

La comunicació entre els clients i servidors es fa mitjançant un protocol de comunicacions que descriu com es poden comunicar i quines informacions es poden intercanviar (en el cas de pàgines web, seria HTTP).

El servidor té les següents característiques:

- Passiu (esclau).
- Espera peticions.
- Quan rep una petició la processa i retorna una resposta.

[2] By Tiago de Jesus Neves (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Un client, en canvi, es caracteritza per:

- Actiu (mestre).
- Envia peticions.
- A l'enviar una petició espera una resposta i la processa.

5.2. Web Service

Existeixen moltes definicions per el Serveis Web. Això demostra la complexitat a l'hora de donar una definició adequada que englobi tot el que són i impliquen. Una possible definició seria tractar-los com un conjunt d'aplicacions o de tecnologies amb capacitat per interoperar amb la web. Aquestes aplicacions o tecnologies intercanvien dades entre sí amb l'objectiu d'oferir serveis.

Aquests serveis proporcionen mecanismes de comunicació estàndards entre diferents aplicacions que interactuen entre elles per presentar informació dinàmica al usuari.

Per proporcionar interoperabilitat i extensibilitat entre les seves aplicacions, i que al mateix temps sigui possible la seva combinació per realitzar operacions complexes, es necessària una arquitectura de referència estàndard.

5.3. REST

Representational State Transfer (REST) és un estil d'arquitectura dissenyat per aplicacions web que actua com a un Servei Web. La idea és que, en comptes de utilitzar mecanismes complicats com CORBA, RPC o SOAP per comunicar-se entre ordinadors, es millor utilitzar HTTP per aquesta comunicació.

Les aplicacions REST utilitzen peticions HTTP per crear o actualitzar dades (POST), llegir dades (GET), i esborrar-les o modificar-les. De manera que, REST utilitza HTTP per les 4 operacions CRUD (Create/Read/Update/Delete).

Les principals restriccions donades per l'arquitectura són:

1. **Client-Servidor**: Ha de ser una interfície uniforme que separa els clients dels servidors: El client no ha de conèixer res referent sobre com es guarda la informació i on, ja que aquesta responsabilitat recau en els servidors.
   Els servidors tampoc han de conèixer res dels clients, de manera que els servidors són simples i més escalables. A més, tant clients com servidors poden ser canviats independentment, sempre que es respecti la interfície que els manté units per comunicar-se.
2. ***Stateless***: la comunicació entre el client i el servidor no pot ser emmagatzemada al servidor per peticions futures. Cada petició provinent d'un client conté tota la informació necessària pel servir la petició, i la sessió dels clients és controlada en el client.
3. ***Cacheable***: la informació que es retorna en forma de resposta al client pot ser desada a la cache.
4. **Sistema de capes**: un client no podria dir si està connectat directament al servidor o a un intermediari entre ell i el servidor.
5. **Interfície uniforme**: és fonamental per qualsevol disseny REST del Servei Web. Simplifica i redueix l'acoplament de l'arquitectura, facilitant així que cada part d'aquesta arquitectura es pugui desenvolupar individualment.
   Els quatre principis que guien una correcte interfície uniforme són:
   a. Identificació dels diferents recursos (un recurs dins el disseny web és un element fonamental de l'aplicació a desenvolupar).
   b. Manipulació dels recursos a través de les seves representacions.
   c. Missatges auto-descriptius.
   d. Utilitzar els hipermèdia com una representació de l'estat (el que s'anomena HATEOAS[3]).

Un Web Service que compleix amb totes les característiques se li atribueix el nom de RESTful.

[3] HATEOAS: Abreviació per Hypermedia as the Engine of Application State.

El terme de **REST** va ser introduït l'any 2000 en una tesi doctoral sobre arquitectures de programari de xarxes. Aquesta tesi va ser escrita per **Roy Thomas Fielding**[4] i dirigida per Richard N. Taylor. Roy va ser un dels principals autors de les especificacions del protocol HTTP i explica en la seva tesi com es pot aprofitar aquest protocol per tal de desenvolupar aplicacions distribuïdes.

Segons la tesi de Roy es poden dissenyar interfícies XML+HTTP seguint la filosofia de ***Remote Procedure Call***[5] sense utilitzar la complexitat del protocol SOAP[6].

Com els Serveis Web, REST és:

- Independent de la plataforma (és igual si el servidor es Unix, el client Mac i qualsevol altre combinació que pugui fer-se).
- Independent del llenguatge de programació (C# pot comunicar-se amb Java, etc).
- Basat en estàndards.

5.4. Single Page Application : SPA

Single Page Application (SPA) és una aplicació web o pàgina web, que treballa amb una única pàgina amb l'objectiu d'aportar una millor experiència d'usuari.

En una SPA, tot el codi necessari (HTML, JavaScript i CSS) és proporcionat quan es carrega la pàgina. També es podria donar al cas que els recursos s'anessin carregant dinàmicament i s'afegissin a mesura que l'usuari navega per la web.

La pàgina web no és recarrega en cap moment del procés, ni transfereix el control a una altre pàgina, encara que les tecnologies més modernes emprades en la web

[4] Roy Thomas Fielding (1965), és un dels autors principals de la especificació HTTP i una autoritat citada freqüentment dins el món de l'Arquitectura de Xarxes. Més informació: http://roy.gbiv.com/
[5] **Remote Procedure Call** o crida a procediment remot, és una tecnologia que permet a un programa d'ordinador fer que una subrutina o procediment s'executi en un altre espai d'adreces (habitualment en un altre ordinador en una xarxa compartida) sense que el programador hagi de programar explícitament els detalls d'aquesta interacció remota.
[6] **SOAP (Simple Object Access Protocol),** és un protocol de comunicació dissenyat per intercanviar missatges en format XML en una xarxa d'ordinadors, normalment sobre el protocol HTTP. Habitualment s'utilitza per accedir a Serveis Web.

(com ara les que estan incloses amb l'HTML5) poden proporcionar la percepció i la capacitat de navegació de pàgines separades lògicament dintre l'aplicació.

A continuació es presenten algunes característiques d'una SPA:

- *Chunking*: La pàgina web es construeix amb fragments d'HTML i JSON en comptes de rebre HTML directament del servidor web en resposta a les peticions.
- Controladors: El codi JavaScript que manipula les dades i els elements DOM, la lògica de l'aplicació i les crides AJAX es canvien per controladors que separen les vistes i els models utilitzant el patró MVC (*Model View Controller*).
- *Templating*: Plantilles HTML.
- *Routing*: Selecció de vistes i navegació sense recarregues que preserven l'estat de la pagina, els elements i les dades.
- Comunicació a temps real.

5.5. Patró MVC: Model View Controller

Model View Controller (MVC) és un patró o model d'abstracció de desenvolupament de software que separa les dades d'una aplicació, la interfície d'usuari i la lògica de negoci en tres components diferents. El patró MVC es veu freqüentment en aplicacions web, on la vista és la pàgina HTML i les dades es proporcionen de manera dinàmica a la pàgina.

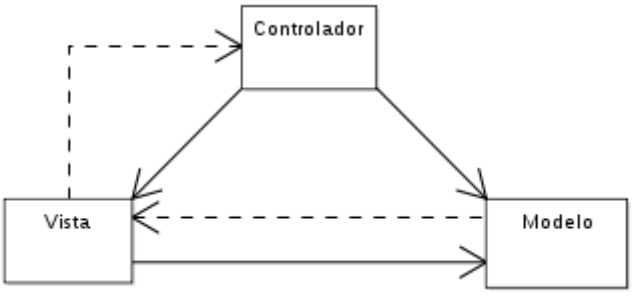


Figura 4: Exemple MVC.

Aquesta imatge és un diagrama senzill on es mostra la relació entre el model, la vista i el controlador. Les línies sòlides indiquen una associació directa, i les altres una indirecta.

Més concretament:

- Model: Representació específica de la informació amb la qual s'opera. Resumint, el model es limita simplement a servir dades a la vista i rebre dades del controlador. El sistema també pot operar amb més dades que no són relatives a la presentació, fent així ús integrat d'altres lògiques de negoci i dades relacionades amb el sistema modelat.
- Vista: Presenta el model en un format adequat per a permetre als usuaris interactuar amb les dades.
- Controlador: Respon als events, normalment ocasionats per accions dels usuaris mentres interactuen amb la vista. Llavors realitza les peticions necessàries per obtenir les dades i enviar-les al model.

Originalment MVC va ser desenvolupat per aplicacions d'escriptori, però ha sigut àmpliament adaptat com arquitectura per dissenyar i implementar aplicacions web en els principals llenguatges de programació. S'han desenvolupat molts frameworks, comercials i no comercials, que implementen aquest patró.

Els primers frameworks MVC per programació web plantejaven un enfocament de client lleuger, on gairebé totes les funcions (vista, model i controlador) requeien al servidor. Actualment això ha canviat i existeixen frameworks que permeten que certs components MVC s'executin de forma parcial o total en el client.

5.6. Sistema de Control de Versions

Es diu Control de Versions a la gestió dels diferents canvis que s'han realitzat en els elements d'algun producte o una configuració del mateix. Una versió, revisió o edició d'un producte, és l'estat en que es troba el mateix en un moment concret del seu desenvolupament o modificació.

El Control de Versions es pot realitzar manualment, però és més adequat utilitzar una de les moltes eines disponibles que faciliten aquesta gestió. Aquestes eines s'anomenen Sistemes de Control de Versions, o SVC (System Version Control). Aquests sistemes faciliten l'administració de les diferents versions de cada producte desenvolupat, així com les possibles especialitzacions realitzades.

Figura 5: SVC més importants.[7]

5.7. Intel·ligència Artificial

El terme d'Intel·ligència artificial (IA) pot tenir tantes definicions com punts de vista diferents. Si s'hagués d'escollir alguna que fos suficient per descriure tot allò que comprèn i tot el que pot aportar, potser caldrien moltes pàgines per explicar-ho. Però si que es pot concretar per a què serveix: construir programes.

Bàsicament, es vol que aquests programes realitzin les següents tasques:

- Actuar com les persones: Test de Turing.
- Raonar com les persones: El model cognoscitiu.
- Raonar racionalment: Les lleis del pensament.

[7] Totes les imatges s'han extret de la wikipedia: http://en.wikipedia.org

- Actuar racionalment: L'agent racional.

Potser no totes, però si que almenys cal que una d'elles hi sigui present, per ser considerat un programa amb intel·ligència artificial.

El camp de la intel·ligència artificial te moltes aplicacions, i concretament per aquest projecte s'estudiarà la cerca d'una solució òptima per a realització d'un horari escolar.

Primer, es presentarà la problemàtica que existeix amb l'elaboració dels horaris i seguidament, s'exposarà una part teòrica sobre la solució aportada a aquest problema: un algoritme genètic.

5.7.1. Problema de *Timetabling*

Un horari es fàcil de fer quan hi ha pocs factors a tenir en compte, com per exemple un conjunt petit de assignatures, sense cap més restricció que la de no solapament.

El problema sorgeix quan comencen a sortir més coses a tenir en compte, com ara, els professors que imparteixen les assignatures. No només s'ha de tenir en compte el no solapament d'hores i dies, també s'ha de tenir en compte que aquell professor no estigui impartint una altre assignatura diferent. Seguint amb aquesta dinàmica, un altre problema afegit seria la disponibilitat d'aquests professors: Si es dóna flexibilitat horària, llavors cada professor te les seves pròpies restriccions (dies que pot treballar, hora d'entrada, i hora de sortida).

Afegit a tot això, s'ha de sumar el control de les aules on es poden realitzar aquestes assignatures. Es poden cursar dues assignatures al mateix temps, sempre que siguin en dues aules diferents i les dues estiguin disponibles, però és obvi que no es podrien cursar dos assignatures a la mateixa aula en el mateix instant de temps.

Cal tenir en compte els alumnes també: els és impossible estar a dues assignatures diferents a la mateixa hora, de manera que dintre totes les possibilitats, si es coneixen els alumnes que estan matriculats a les assignatures, caldria tenir-los en compte també.

Tot això fa que la tasca comenci a ser difícil de realitzar si s'ha de fer manualment: començarien a sortir conflictes. Només cal imaginar el cas hipotètic d'una acadèmia amb 10 assignatures, on cada assignatura la cursa un professor diferent amb les seves pròpies restriccions, i a més hi ha 3 aules sobre les quals repartir les assignatures. Tot això, encara es podria tolerar amb certes dificultats, però en el cas que siguin 30 assignatures, número variant de professors per assignatura, i 15 aules sobre les quals repartir i vigilar el no solapament, seria molt fàcil cometre un error. Per tal d'evitar aquests errors totalment comprensibles, és apropiat aplicar tècniques de cerca avançada que trobin la millor solució.

5.7.2. Algoritmes genètics

Els Algoritmes Genètics (AGs) són mètodes adaptatius que poden utilitzar-se per resoldre problemes de cerca i optimització. Es basen en el procés genètic dels organismes vius. Al llarg de les generacions, les poblacions evolucionen dins la naturalesa, d'acord amb els principis de selecció natural i la supervivència dels més forts (aquests principis van ser postulats per Darwin al 1859). Per imitació d'aquest procés, els AGs són capaços d'anar creant solucions per resoldre problemes del món real. L'evolució d'aquestes solucions fins a valors òptims del problema depèn en bona mesura d'una adequada codificació de les mateixes.

Els AGs treballen amb una població d'individus, cada un dels quals representa una solució factible al problema. A cada individu se li assigna un valor o puntuació, relacionat amb la correctesa de la solució que aporta. Com més gran sigui l'adaptació d'un individu al problema, major serà la probabilitat de que el mateix sigui seleccionat per a reproduir-se, creuant el seu material genètic amb un altre individu que també ha passat pel mateix procés de selecció.

Aquest creuament produirà nous individus, els quals comparteixen alguna de les característiques dels seus pares.

D'aquesta manera es produeix una nova població de possibles solucions, que substitueix la població anterior, amb les millors característiques dels antecessors. Així es van explorant les millors característiques.

El poder dels Algoritmes Genètics prové del fet que es tracta d'una tècnica robusta i poden tractar amb èxit gran varietat de problemes de diferents àrees.

Encara que no es garanteix que els Algoritmes Genètics trobin la solució òptima, existeix l'evidència empírica de que es troben solucions d'un nivell acceptable, amb un temps competitiu comparat amb la resta d'algoritmes d'optimització combinatòria.

A continuació s'explicarà la terminologia necessària per entendre l'algoritme genètic:

Esquema bàsic

Terminologia

- ***Població***: Conjunt de individus que formen una generació.
- ***Individu***: Cromosoma amb material genètic divers.
- ***Cromosoma***: Conjunt de gens que formen una possible solució.
- ***Gen***: Informació necessària per a trobar una solució òptima.

Algoritme

- ***Esquema de l'algoritme***.

  Hi ha moltes variacions dels algoritmes genètics, però l'esquema més bàsic segueix el següent:

```
BEGIN /* Algoritmo Genetico Simple */
    Generar una poblacion inicial.
    Computar la funcion de evaluacion de cada individuo.
    WHILE NOT Terminado DO
    BEGIN /* Producir nueva generacion */
        FOR Tamaño poblacion/2 DO
        BEGIN /*Ciclo Reproductivo */
            Seleccionar dos individuos de la anterior generacion,
            para el cruce (probabilidad de seleccion proporcional
            a la funcion de evaluacion del individuo).
            Cruzar con cierta probabilidad los dos
            individuos obteniendo dos descendientes.
            Mutar los dos descendientes con cierta probabilidad.
            Computar la funcion de evaluacion de los dos
            descendientes mutados.
            Insertar los dos descendientes mutados en la nueva generacion.
        END
        IF  la poblacion ha convergido THEN
            Terminado := TRUE
    END
END
```

Com es pot observar, hi ha un procés d'inicialització amb cert grau d'aleatorietat.

Altres operacions que apareixen, comentades a continuació, serien avaluació, selecció, creuament, mutació i inserció.

- ***Funció d'avaluació***: Proporciona el valor de fitness, o dit d'una altre manera, com de bé s'adapta un cromosoma a la situació del problema com a possible solució. Com més condicions compleixi amb correctesa millor puntuació tindrà.
- ***Operadors***:

Selecció	Consisteix en un procés que: per a cada individu d'una població, se li assigna una probabilitat de supervivència proporcional a la qualitat (fitness), seguidament genera una població intermitja, de la qual escull parelles de forma aleatòria.
Creuament	Consisteix en un intercanvi de material genètic entre dos cromosomes (individus). L'objectiu de l'encreuament es aconseguir que el descendent millori la puntuació dels pares.
Mutació	En aquest cas, es modifica part de la informació dels gens d'un individu, creant així un individu diferent.

Avantatges i desavantatges dels algoritmes genètics

Un avantatge que proporcionen als algoritmes genètics és que són intrínsecament paral·lels, es a dir, operen de forma simultània amb varies solucions, les quals van descartant depenent de si no són les millors. Això supera els algoritmes tradicionals que per cada cerca només poden tenir en compte una solució.

També amb les cerques tradicionals corria el risc de caure en màxims locals (solucions que semblen les millors però que realment no ho són), però en aquest cas els algoritmes genètics es veuen menys afectats. Tampoc s'ha d'oblidar que es pot manipular diversos paràmetres simultàniament, i això permet establir diversos objectius a resoldre.

Un desavantatge important a tenir en compte és el procés de definició d'una representació fidel al problema que s'estudia. Cal adaptar el problema als seus requeriments, i sobretot definir una codificació correcta del material genètic que representi tota la informació necessària per a fer els càlculs.

Poden tardar molt a convergir en una solució, si és capaç de trobar-la. Depèn molt del mida de les poblacions, el número de generacions, entre altres.

També es podria donar el cas que si un individu és molt adequat, llavors aquest afecti a les generacions futures fent que la població en general s'assembli a ell degut a que és el millor, i llavors no hi hagi gaire grau de diferencia entre els individus molt aviat, sense tenir temps de fer gaires generacions. Això podria fer que s'arribés a un màxim local.

6. Requisits del sistema

En aquesta part s'exposa el resultat final de l'etapa de recollida de requeriments que es va realitzar al principi del projecte. Per una banda, s'exposaran els requeriments funcionals, i per l'altre, els no funcionals. Els requeriments funcionals s'agrupen segons els actors que poden realitzar-los.

6.1. Requisits funcionals

Els usuaris de l'aplicació esta previst que puguin ser: *Professors* (Personal Docent), s*ecretaria* (Personal d'administració), i *Director* (Personal de Direcció).
Les tasques que el col·lectiu de **Personal Docent** pot realitzar són:

1. Gestionar les notes dels alumnes.
    a. Entrar les notes dels alumnes.
    b. Modificar notes dels alumnes.
2. Consultar informació dels alumnes.

Les tasques que el col·lectiu de **Personal d'Administració** pot realitzar són:

1. Gestionar els alumnes**:** Consistiria a poder donar d'alta els alumnes i assignar-los les assignatures de les quals s'han matriculat (o cursos complets). En cas que ja estiguessin donats d'alta seria simplement afegir noves assignatures/cursos.
2. Gestionar professors: Donar d'alta els professors de l'escola, acadèmia o universitat i gestionar les seves dades.
3. Gestionar cursos.
    a. Donar d'alta i baixa els cursos.
    b. Mantenir la informació i les assignatures que ofereixen.
4. Gestionar assignatures.
    a. Donar d'alta i baixa assignatures.
    b. Mantenir la informació.
5. Gestionar horaris : Mitjançant uns requisits, elaborar horaris per les seves assignatures.

Les tasques que el col·lectiu de **Personal de Direcció** pot realitzar són:
Podrà realitzar les mateixes tasques que els usuaris anteriors, però a més pot iniciar i tancar els períodes de matricula i avaluació i a més seran els únics que podran gestionar les dades dels personals (això implica donar d'alta nou personal, assignar permisos, esborrar personal, modificar les dades).

Els alumnes no interactuaran amb l'aplicació.

6.2. Requeriments no funcionals

1. Disposar d'un navegador per tal de poder executar correctament el client web.
2. Degut a que s'utilitza Bootstrap, s'ha de tenir en compte que aquest, segons el dispositiu (Device) que accedeixi a la pàgina web, tindrà més acceptació o menys (veure la figura 5, on surten els principals navegadors i el suport de dispositius).

	Chrome	Firefox	Internet Explorer	Opera	Safari
Android	✓	✗	-	✗	-
iOS	✓	-	-	✗	✓
Mac OS X	✓	✓	-	✓	✓
Windows	✓	✓	✓	✓	✗

Figura 6: Taula de suport de device vs navegador[8].

Concretament, per Internet explorer, les versions 8 i 9 poden ser suportades però s'ha de tenir en compta que moltes propietats de CSS3 i elements HTML5 no funcionen per aquestes versions del navegador.

3. Disposar de connexió a Internet.
4. El servidor cal que estigui desplegat correctament i una prèvia configuració de la base de dades.

[8] Imatge extreta del llibre online:
http://librosweb.es/bootstrap_3/capitulo_1/compatibilidad_con_los_navegadores.html

7. Estudis i decisions

En aquest apartat s'anomenen les llibreries, els frameworks i l'entorn que s'ha decidit utilitzat per aquest projecte.

7.1. Client

Per la part client, es va decidir realitzar una aplicació web, concretament una SPA. Hi ha diferents opcions a escollir, les quals tenen en comú que son frameworks de JavaScript. Els principals: Ember, Backbone i Angular.

Figura 7: Frameworks MVC més destacats disponibles per programació web.[9]

Els quatre són frameworks JavaScript basats en el patró MVC. A continuació s'explicaran els pros i contres de els 3 frameworks i la decisió presa.

Backbone.js és un framework que es caracteritza pels següents punts:

- No ocupa gaire: sense comptar les dependències, és un fitxer de unes 800 línies de codi.
- Té dependència amb una llibreria que es diu Undercore.js .
- Corba d'aprenentatge mínima: si es tenen coneixements de Undercore.js i jQuery, altrament cal un estudi previ.
- Flexibilitat alta i gran control: es pot programar i canviar el comportament de pràcticament qualsevol element.

[9] Imatges extretes de: wikimedia.

Desavantatges a destacar: deixa massa responsabilitat a les mans del programador, i pot tornar-se un problema a l'hora d'estructurar el codi. Una aplicació de mida mitjana o gran es pot transformar fàcilment en un malson d'una mida impressionant si no es revisa el codi amb freqüència i es refactoritza.

AngularJS, de Google, és un framework amb una filosofia molt diferent del Backbone.

Les seves característiques principals són:

- Gran popularitat i un gran equip al darrera, ofereixen molta documentació i vídeos amb tutorials.
- Et permet desenvolupar una SPA molt estructurada.

Desavantatges a destacar: és un framework que funciona a base de directives, i això tot i que proporciona una estructura, també limita molt.

Ember.js és també un framework molt interessant, que es basa en la filosofia "*Convention over Configuration"* (vindria a dir que cal seguir les convencions en comptes de configurar). Té a favor un sistema de Routing molt avançat.

Desavantatges a destacar: és complicat entendre bé el llenguatge i és el framework amb més rigidesa del mercat degut al *Convention over Configuration*. La informació no esta degudament actualitzada perquè últimament han hagut canvis molt dràstics.

Sabent tot això, la decisió va ser utilitzar AngularJS, donat que es buscava bona documentació, simplicitat i estructura.

Altres components importants incorporats:

- **Bootstrap**: Framework que et permet desenvolupar webs responsives[10], una necessitat molt important ara que molts clients de l'aplicació poden ser tant ordinadors com mòbils i tabletes.

[10] Web responsive: El disseny web adaptable o adaptatiu (en anglès, *Responsive Web Design*) és una filosofia de disseny i desenvolupament web que mitjançant l'ús de estructures i imatges fluides, així com de *media-queries* dins la fulla d'estil CSS, aconsegueix adaptar el lloc web al entorn del usuari.

- **jQuery**: Llibreria JavaScript senzilla d'utilitzar que facilita molt la gestió dels elements de l'HTML, i els events que es produeixen mentre l'usuari interactua amb la interfície.
- **FullCalendar**: és un plug-in per jQuery que permet realitzar un calendari complert, amb possibilitat de drag&drop dels elements. Utilitza peticions AJAX per actualitzar els events en temps reals per cada més. També es pot adaptar visualment.

Degut que l'AngularJS és bastant restrictiu, cal definir directives per tal de poder utilitzar els components Bootstrap i FullCalendar que abans s'han comentat, de manera que també es van afegir les corresponents directives UI-Bootstrap i UI-Calendar.

7.2. Servidor

Una vegada decidida la part client, calia decidir la part servidor. Ja s'havia decidit que es faria servir un Servei Web RESTful, però calia saber quin llenguatge de programació es faria servir.

Figura 8: Diferents llenguatges de programació.

La figura 7 exposa els llenguatges de programació que s'utilitzen més a l'hora de programar la part servidor. Entre totes les opcions, es va escollir JavaEE, per varis motius:

- És molt utilitzat per les empreses i a més compta amb una comunitat molt madura.
- El coneixia prèviament degut a que s'havia realitzat un treball amb un servidor implementat amb JavaEE.
- Te varies especificacions de API molt útils, entre elles:
    - Java Persistence API (JPA).
    - Enterprise Java Beans (EJB).
    - Web services (JAX-RS).
    - Java Server Faces (JSF).
- Proporciona una estructura multicapa:

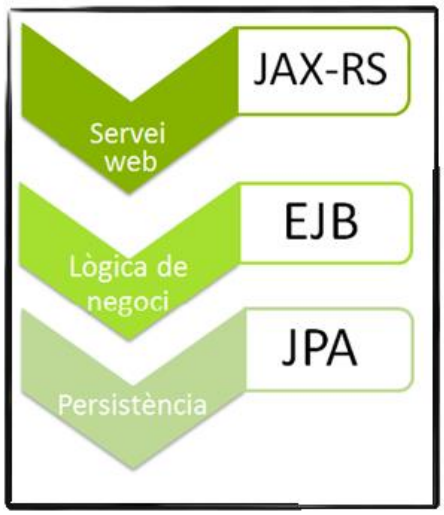


Figura 9: Exemple estructura multicapa.

Per aquest projecte a calgut utilitzar JAX-RS, EJB i JPA, però concretament:

- Jersei – RESTful web service: L'objectiu d'aquest framework és proporcionar una implementació de JAX-RS API, a més de la seva pròpia, que simplifica el servei RESTful.
- EclipseLink: Implementació del JPA, amb suport a varies tipologies de Bases de Dades.

7.3. Entorn de desenvolupament

Per programar he utilitzat l'IDE Netbeans, combinat amb un sistema de control de versions[11] del codi anomenat GIT[12] que ha sigut molt útil. Conjuntament amb el GIT, també s'ha utilitzat una servei online que proporciona espai per desar el codi en repositoris remots. Aquest servei s'anomena Bitbucket[13].

Com a servidor d'aplicacions, es van tenir en compte dos de diferents a l'hora de prendre la decisió: WildFly o Glassfish.

Donat que durant la instal·lació del Netbeans es podia incloure per defecte el servidor Glassfish (també desenvolupat per Sun), i a més la majoria de documentació i projectes que incorporen el JavaEE fan servir per defecte el servidor Glassfish, es va considerar preferible escollir-lo.

Per últim, per la base de dades vaig decidir utilitzar MySql, donat que és una base de dades que, també, s'utilitza per defecte en el JavaEE.

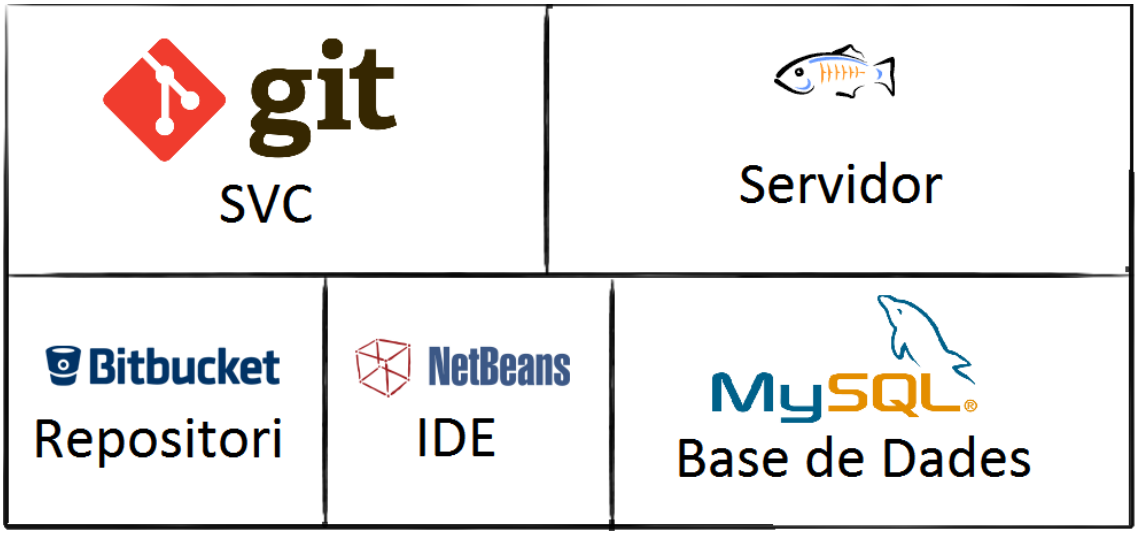


Figura 10: Entorn de desenvolupament.

[11] Sistema de Control de Versions (SVC) http://es.wikipedia.org/wiki/Control_de_versiones
[12] GIT: http://git-scm.com/
[13] Bitbucket: https://bitbucket.org/

8. Anàlisi i disseny del sistema

A continuació es farà un anàlisi dels requeriments funcionals mitjançant un seguit de diagrames de casos d'ús i les seves fitxes corresponents, i es presentarà un disseny que dona una solució, amb el diagrama de classes i el disseny de la API.

8.1. Anàlisi dels requisits

8.1.1. Diagrames de casos d'ús

Els diferents actors que interactuen amb el sistema són:

1. Personal Docent.
2. Personal de l'Administració.
3. Personal de Direcció.

A partir dels requeriments i dels diferents actors s'ha definit el diagrama general de casos d'ús:

8.1.1.1. Diagrama de casos d'ús de l'actor Personal Docent

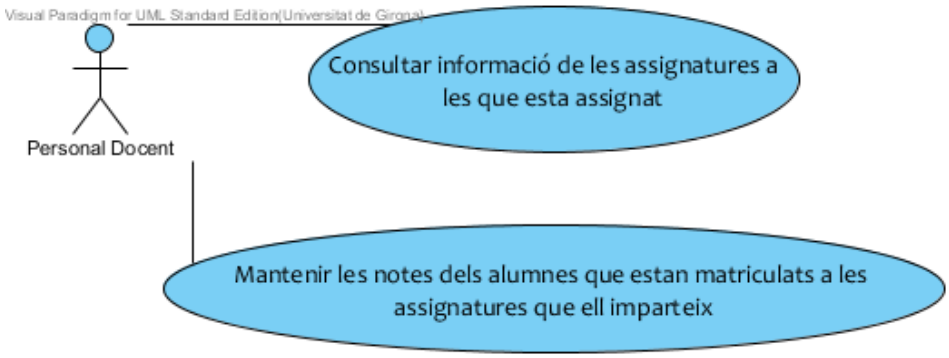


Figura 11: Diagrama casos d'ús (1)

8.1.1.2. Diagrama de casos d'ús de l'actor Personal Administració

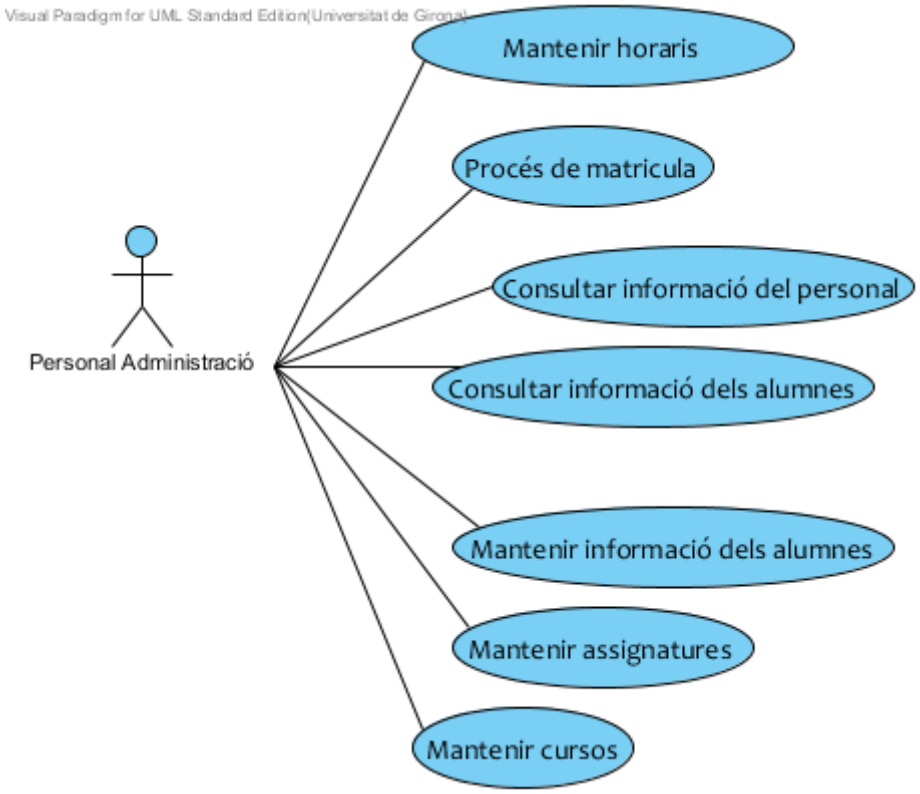


Figura 12: Diagrama casos d'ús (2)

A continuació es descriuen amb més detalls els casos d'ús més importants.

Diagrama de casos d'ús del procés de matricula

El procés de matricula és un procés en cascada, que inclou dues possibilitats: matricular un alumne que vol cursar un curs, o matricular un alumne que vol cursar algunes assignatures.

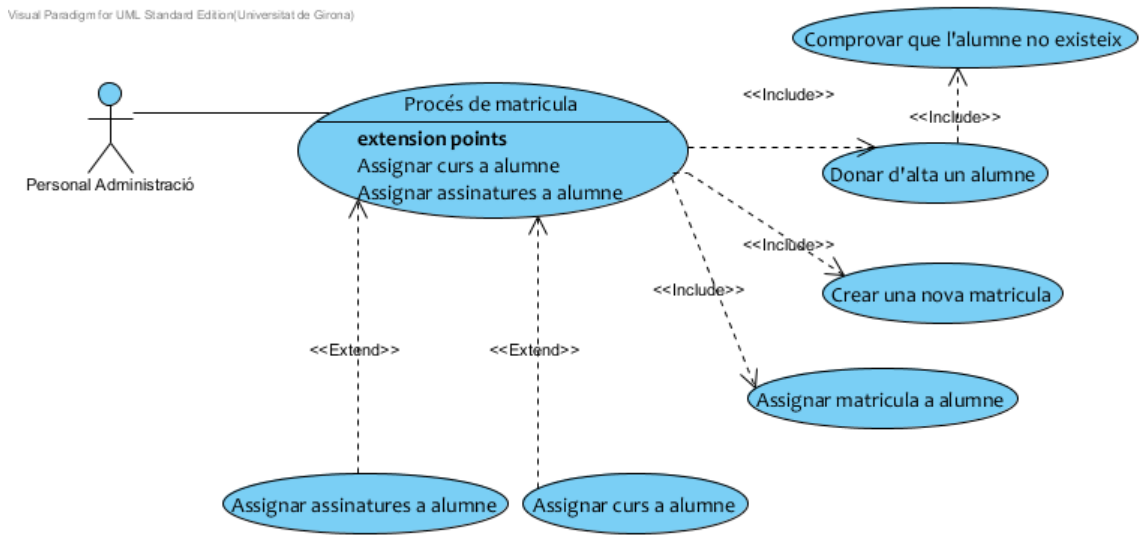


Figura 13: Diagrama casos d'ús (3)

Diagrama de casos d'ús de Mantenir Informació alumnes

En el següent diagrama podem veure en profunditat la gestions necessàries per mantenir els alumnes.

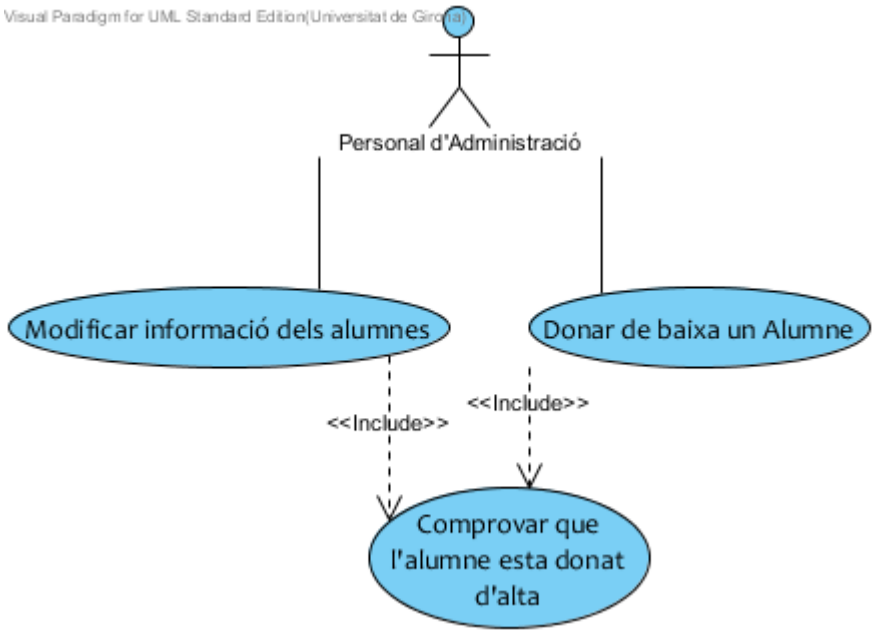


Figura 14: Diagrama casos d'ús (4)

Diagrama de casos d'ús de Mantenir assignatures

En el següent diagrama podem veure en profunditat la gestions necessàries per mantenir les assignatures.

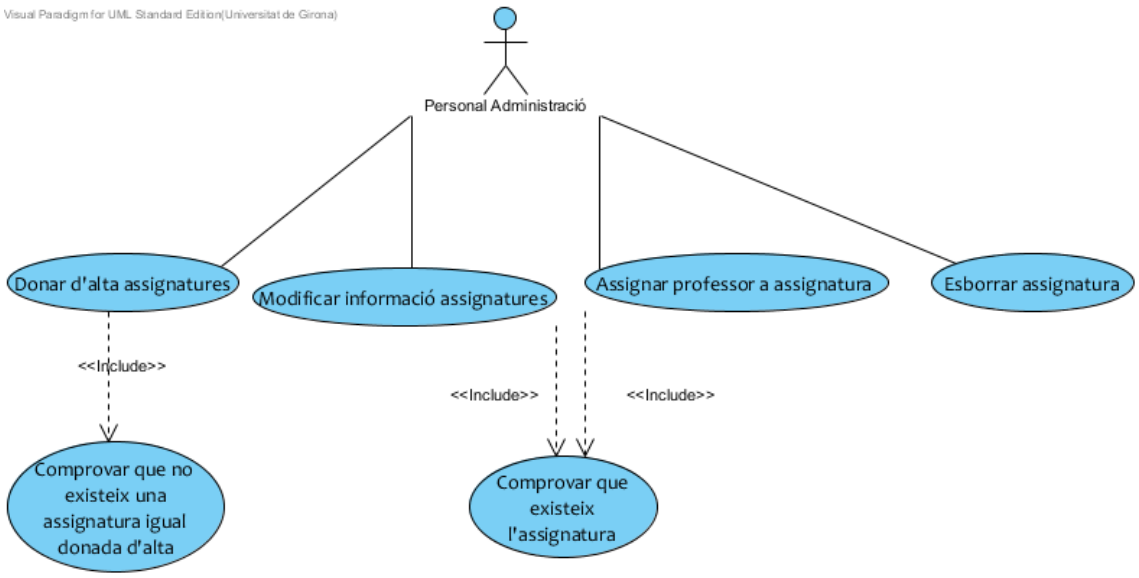


Figura 15: Diagrama casos d'ús (5)

Diagrama de casos d'ús de Mantenir cursos

En el següent diagrama podem veure en profunditat la gestions necessàries per mantenir els cursos.

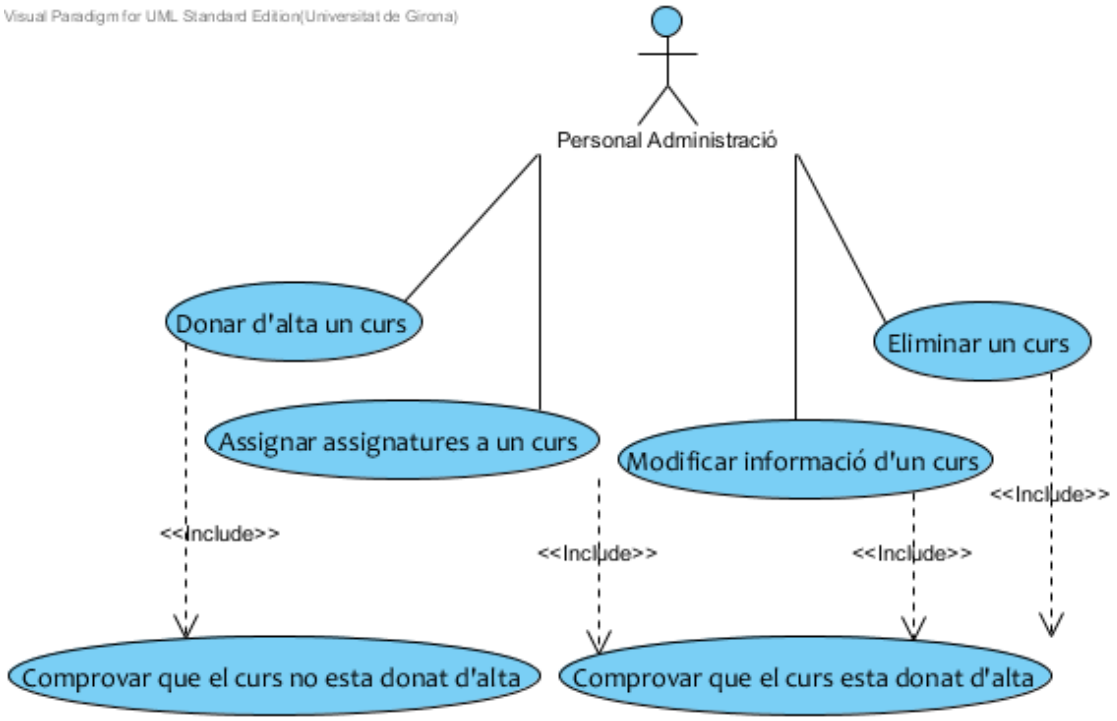


Figura 16: Diagrama casos d'ús (6)

Diagrama de casos d'ús de Mantenir horaris

Mantenir els horaris és una tasca que la majoria la realitza el personal d'administració, però hi ha en concret que s'encarrega el sistema mateix. El procés de creació d'un horari pot esta condicionat segons si es vol crear un esdeveniment relacionat amb una assignatura individual, com ara classes particulars o d'instrument, o si es vol crear un conjunt d'esdeveniments que formaria un horari complet, per totes les assignatures grupals donades d'alta a la base de dades del sistema. Si fos el segon cas, un cop el sistema generés el millor horari possible, caldria donar d'alta tots els esdeveniments relacionats a aquest horari.

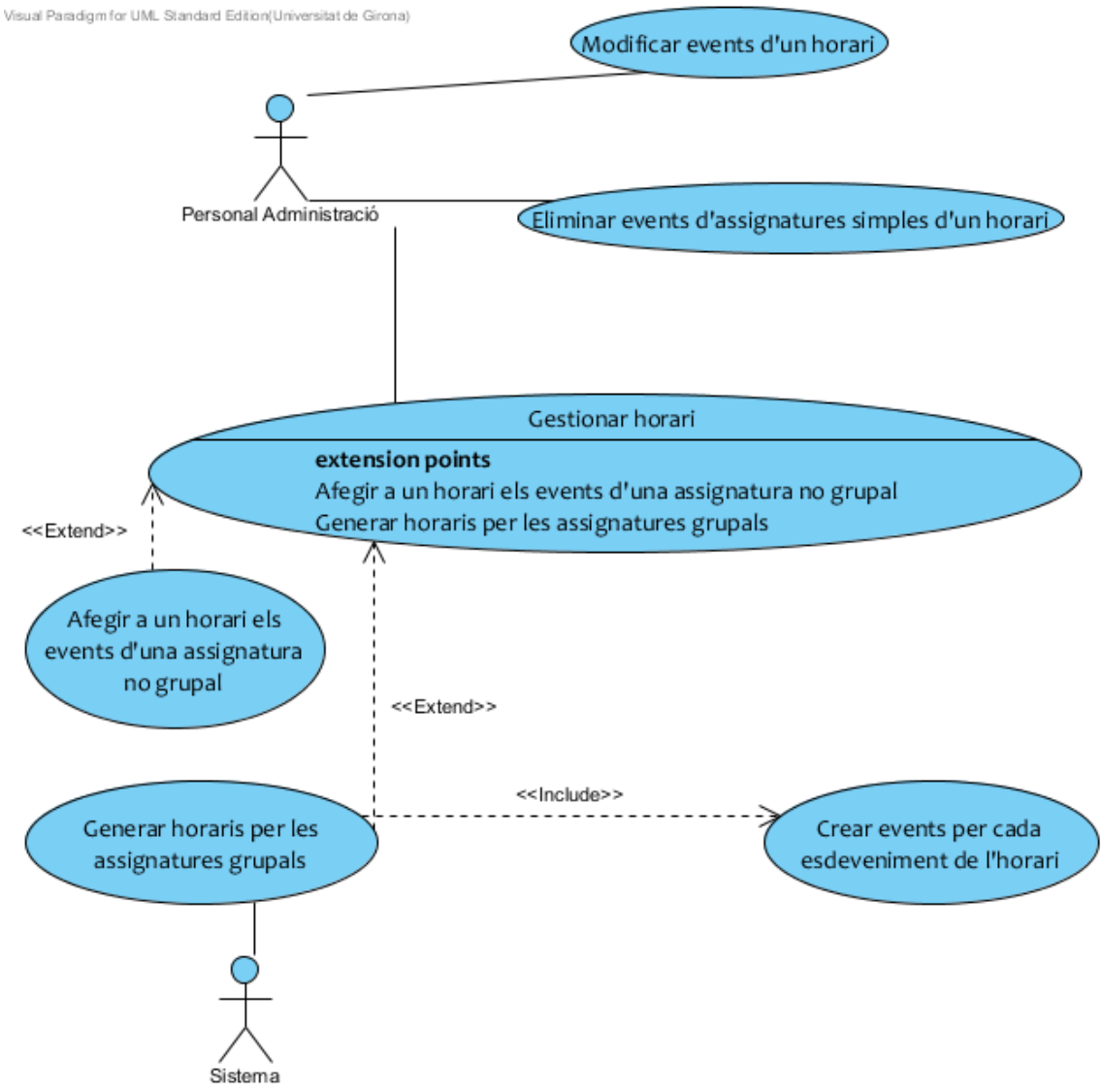


Figura 17: Diagrama casos d'ús (7)

8.1.1.3. Diagrama de casos d'ús de l'actor Personal Direcció

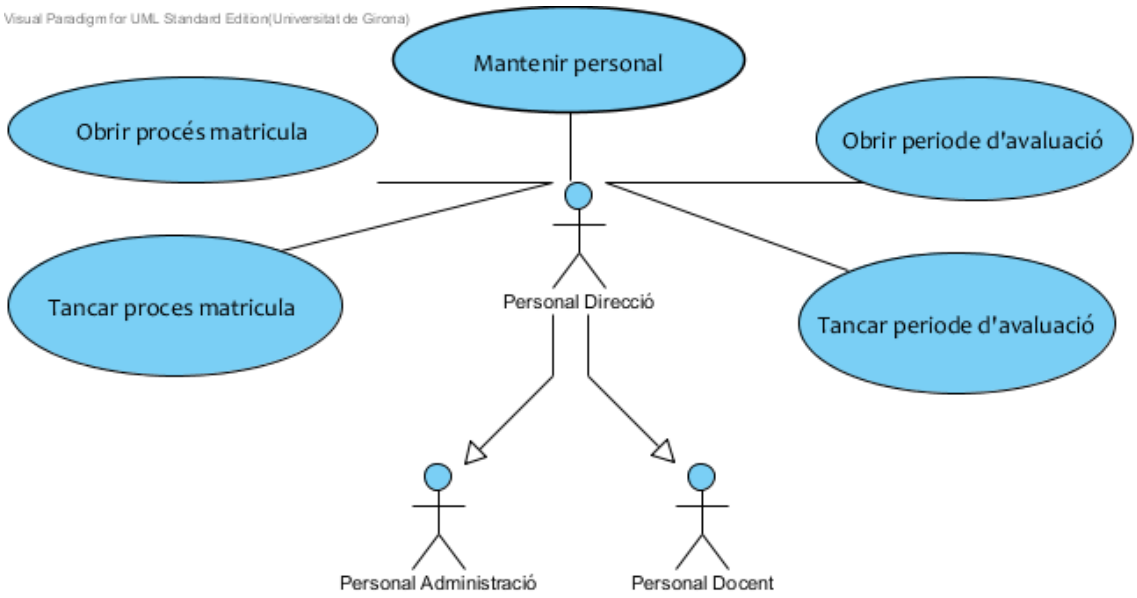


Figura 18: Diagrama casos d'ús (8)

Tal com es pot veure en el diagrama, el personal de Direcció pot fer tot el que els altres usuaris poden fer. A continuació es descriu amb més detall en que consisteix mantenir el personal.

Diagrama de casos d'ús de Mantenir personal

En el següent diagrama podem veure en profunditat la gestions necessàries per mantenir el personal de l'aplicació.

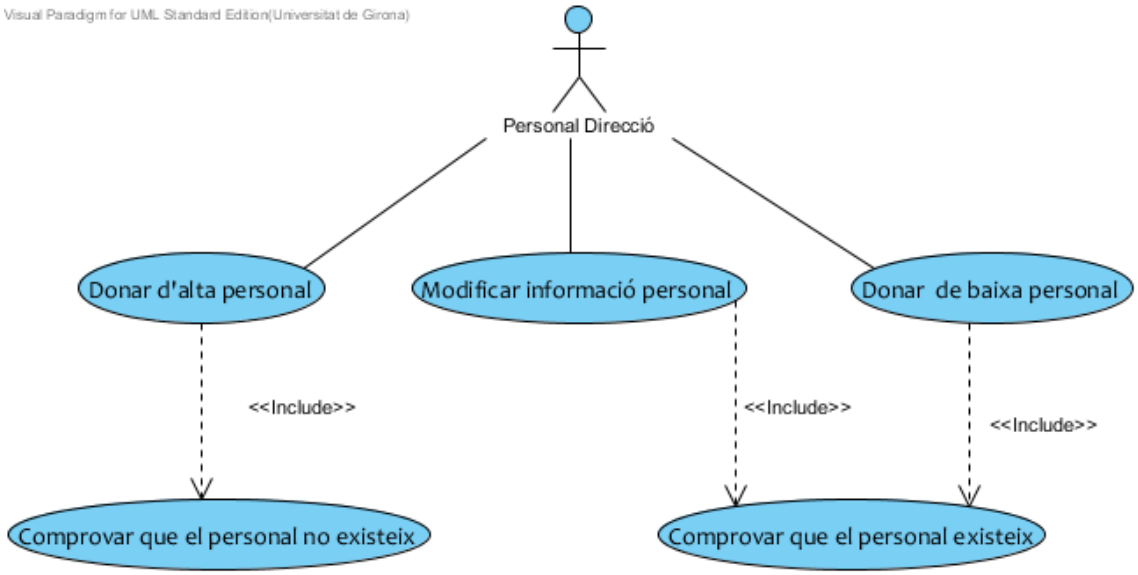


Figura 19: Diagrama casos d'ús (9)

8.1.2.Fitxes de cas d'ús

En aquest apartat s'exposarà amb més detall els casos d'ús més importants que apareixen en els diagrames de casos d'ús anteriors.

8.1.2.1. Fitxes de casos d'us de l'actor Personal Administració

Nom	**Assignar assignatures a un alumne matriculat**		
Data creació	29-08-2013	**Visió**	Usuari
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Un cop la matricula ha estat creada, s'assignarà a l'alumne les assignatures que vol cursar (una o varies assignatures).			
Precondició: Alumne donat d'alta i assignatures donades d'alta. Alumne conegut.			
Escenari Principal 1. Seleccionar assignatures matriculades. 2. Per a cada assignatura. 2.1. Assignar alumne a assignatura. 3. Fper.			
Postcondició: Assignatura amb alumne assignat.			

Nom	**Assignar professor a una assignatura**		
Data creació	29-08-2013	**Visió**	Usuari
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Professor és un personal donat d'alta amb tipus establert com a professor. S'assigna a una assignatura un professor.			
Precondició: Assignatura i professor donats d'alta. Professor no ha estat assignat a l'assignatura prèviament.			
Escenari Principal 1. Mostrar assignatures. 2. Seleccionar assignatura. 3. Mostrar professors. 4. Seleccionar professor. 5. Seleccionar llista de professors d'assignatura. 6. Afegir professor a la llista de professors. 7. Actualitzar llista. 8. Assignar assignatura al professor. 9. Assignar alumne a assignatura.			
Postcondició: Professor assignat.			

Nom	Assignar curs a un alumne matriculat		
Data creació	29-08-2013	**Visió**	Usuari
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Un cop l'alumne esta matriculat (donat d'alta al sistema) se li assignarà el curs (varies assignatures).			
Precondició: Alumne donat d'alta i assignatures donades d'alta. Alumne i curs coneguts.			
Escenari Principal 1. Obtenir assignatures del curs. 2. Assignar assignatures a alumne.			
Postcondició: Alumne amb curs assignat.			

Nom	Donar d'alta assignatures.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Es demanen les dades necessàries per crear una nova assignatura i es guarden.			
Precondició: L'assignatura no ha estat donada d'alta prèviament.			
Escenari Principal 1. Entrar dades assignatura nova. 2. Crear assignatura amb dades.			
Postcondició: Assignatura donada d'alta.			

Nom	Donar d'alta un curs.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Es demanen les dades necessàries per crear un nou curs i es guarden.			
Precondició: El curs no ha estat donat d'alta prèviament.			
Escenari Principal 1. Entrar dades del curs nou. 2. Obtenir assignatures donades d'alta al sistema que es volen assignar al curs. 3. Crear curs amb dades i assignatures. 4. Guardar a les assignatures el curs.			
Postcondició: Curs donat d'alta.			

Nom	Donar d'alta un alumne.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Es demanen les dades necessàries per crear un nou alumne i es guarden.			
Precondició: L'alumne no ha estat donat d'alta prèviament.			
Escenari Principal 1. Entrar dades de l'alumne nou. 2. Crear alumne amb les dades corresponents.			
Postcondició: Alumne donat d'alta.			

Nom	Assignar assignatures a un curs.		
Data creació	29-08-2013	**Visió**	Usuari
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Donada una llista d'assignatures, s'assignen a un curs.			
Precondició: Assignatures i curs donats d'alta.			
Escenari Principal 1. Seleccionar curs. 2. Mostrar assignatures disponibles. 3. Seleccionar assignatura. 4. Seleccionar assignatures del curs. 5. Afegir llista d'assignatures seleccionada. 6. Actualitzar la llista.			
Postcondició: Assignatura amb alumne assignat.			

Nom	Procés de matricula.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal d'administració
Descripció: Es proporcionen les dades de l'alumne i del tutor, les assignatures a cursar o curs complet a realitzar. Un cop totes les dades han estat concretades es genera un comprovant de matricula.			
Precondició: Assignatura/es i/o curs donat d'alta.			
Escenari Principal 1. Crear una nova matricula. 2. Buscar alumne amb identificador que s'ha especificat (es opcional). 3. Si no existeix o si no s'ha especificat identificador: 3.1. Donar d'alta un nou alumne. 4. Altrament 4.1. Buscar alumne amb identificador ID. 5. FSi. 6. Assignar matricula a alumne. 7. Si es matricula d'un curs: 7.1. Assignar curs a alumne. 8. Altrament. 8.1. Assignar assignatures a alumne. 9. FSi.			
Postcondició: Procés de matricula finalitzat.			

Nom	Eliminar una assignatura.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2013	**Actor**	Personal Administratiu.
Descripció: Es seleccionen les assignatures que es volen eliminar i posteriorment es suprimeixen.			
Precondició: Assignatura/es donada/es d'alta.			
Escenari Principal 1. Mostrar assignatures. 2. Seleccionar assignatura. 3. Seleccionar curs que pertany l'assignatura. 4. Eliminar assignatura de la llista de assignatures del curs. 5. Seleccionar alumnes 6. Per cada alumne 6.1. Si estan matriculats a l'assignatura, esborrar-la de la llista d'assignatures. 6.2. Eliminar assignatura de la llista de assignatures. 7. Fper			
Postcondició: Assignatura/es eliminada/des.			

Nom	Eliminar un curs.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Es seleccionen els cursos que es volen eliminar i posteriorment es suprimeixen. Només els cursos, no les assignatures que s'imparteixen.			
Precondició: curs/os donat/s d'alta.			
Escenari Principal 1. Mostrar cursos. 2. Seleccionar curs. 3. Seleccionar assignatures que s'imparteixen en aquest curs. 4. Per cada assignatura 4.1. Eliminar curs de la assignatura. 5. Fper 6. Seleccionar alumnes matriculats al curs. 7. Per cada alumne 7.1. Si estan matriculats al curs, esborrar-li el curs. 7.2. Eliminar curs de la llista de cursos. 8. Fper			
Postcondició: Curs/os eliminat/s.			

Nom	Consultar informació del personal.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Es mostren els professors i la secretària selecciona l'usuari el qual es vol consultar les dades personals (en principi sense restriccions).			
Escenari Principal 1. Mostrar professors. 2. Seleccionar professor. 3. Mostrar una vista amb tota la informació del professor.			
Postcondició: Es mostren les dades del professor.			

Nom	Donar baixa alumne.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: S'elimina l'alumne del sistema.			
Precondició: Alumne donat d'alta.			
Escenari Principal 1. Mostrar alumnes. 2. Seleccionar alumne. 3. Seleccionar assignatures matriculades. 4. Per cada assignatura 4.1. Eliminar alumne de la assignatura. 5. Fper. 6. Seleccionar cursos on apareix l'assignatura. 7. Per cada curs: 7.1. Esborrar assignatura del curs. 7.2. Eliminar alumne de la llista de alumnes. 8. Fper			
Escenari Alternatiu 7. S'ha intentat esborrat un alumne quan esta matriculat d'una altre assignatura que també pertany al mateix curs. Solució: no deixar que s'esborri i passar al següent alumne.			
Postcondició: Alumne eliminat.			

Nom	Consultar informació dels alumnes.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Es mostren els alumnes matriculats i la secretària selecciona l'usuari el qual es vol consultar les dades personals (en principi sense restriccions).			
Escenari Principal 1. Mostrar alumnes. 2. Seleccionar alumne. 3. Mostrar una vista amb tota la informació de l'alumne.			
Postcondició: Es mostren les dades de l'alumne.			

Nom	Modificar informació assignatura.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Seleccionada una assignatura, es modifica la informació que té el sistema.			
Precondició: Assignatura donada d'alta i coneguda.			
Escenari Principal 1. Mostrar una vista amb totes les dades de l'assignatura. 2. Els canvis que es facin es guarden temporalment. 3. Un cop es decideix que són els canvis definitius, es guarden definitivament.			
Postcondició: Informació actualitzada.			

Nom	Modificar curs.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Seleccionat un curs, es modifica la informació que té el sistema.			
Precondició: Assignatura donada d'alta			
Escenari Principal 1. Mostrar una vista amb totes les dades del curs. 2. Els canvis que es facin es guarden temporalment. 3. Un cop es decideix que són els canvis definitius, es guarden definitivament**.**			
Postcondició: Informació actualitzada**.**			

Nom	Modificar alumne.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Seleccionat un alumne, es modifica la seva informació			
Precondició: Alumne donat d'alta			
Escenari Principal 1. Mostrar una vista amb totes les dades de l'alumne. 2. Els canvis que es facin es guarden temporalment. 3. Un cop es decideix que són els canvis definitius, es guarden definitivament.			
Postcondició: Informació actualitzada.			

Nom	Assignar matricula a alumne.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2013	**Actor**	Personal Administratiu.
Descripció: Una vegada donada d'alta la matricula i es crea l'alumne si no existeix, s'assigna la matricula a l'alumne			
Precondició: Matricula creada, alumne donat d'alta, es coneix l'alumne i la matricula.			
Escenari Principal 1. S'obtenen les matricules de l'alumne. 2. Afegir matricula a la col·lecció de matricules de l'alumne. 3. Assignar alumne a matricula.			
Postcondició: Matricula assignada.			

Nom	Gestionar horari.		
Data creació	10-02-2014	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Procés de creació d'un horari.			
Precondició: Assignatures donades d'alta.			
Escenari Principal 1. Es mostres les opcions. 2. Entrar opció 3. Si es vol crear un nou esdeveniment per una assignatura individual. 3.1. Crear Event d'una assignatura no grupal. 4. Altrament 4.1. Generar horari per les assignatures grupals. 5. FSi.			
Postcondició: S'ha guardat els events correctament.			

Nom	Modificar events d'un horari.		
Data creació	10-02-2014	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: Forma part del manteniment d'un horari. Concretament, recull el procés de modificació.			
Precondició: Les dades a modificar són conegudes.			
Escenari Principal 1. Buscar Event. 2. Modificar la informació de l'event. 3. Guardar l'event.			
Postcondició: S'ha modificat la informació de l'event correctament.			

Nom	Crear una nova matricula.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2013	**Actor**	Personal Administratiu.
Descripció: Donada una informació, es dona d'alta una nova matricula.			
Precondició: No existeix cap matricula, per la convocatòria actual, amb el mateix codi identificador.			
Escenari Principal 1. Entrar la informació. 2. Buscar matricules que siguin iguals. 3. Si no n'hi ha: 3.1. Crear nova matricula. 4. Altrament: 4.1. Notificar que no es pot crear. 5. FSi.			
Postcondició: Matricula creada.			

Nom	Eliminar events d'assignatures simples d'un horari.		
Data creació	10-02-2014	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Administratiu.
Descripció: S'elimina un Event que per tant a una assignatura de tipus individual. Els events relacionats a assignatures grupals NO es poden esborrar, degut a que no només tenen relació amb un alumne, sinó amb varis.			
Precondició: Event conegut i donat d'alta.			
Escenari Principal 1. Buscar Event. 2. Obtenir assignatures simples. 3. Per cada assignatura: 3.1. Si tenen l'event assignat: 3.1.1.Esborrar l'event. 3.2. FSi 4. Fper. 5. Obtenir professor de l'event. 6. Esborrar Event del professor. 7. Obtenir alumne de l'event. 8. Esborrar Event de l'alumne. 9. Esborrar definitivament l'event.			
Postcondició: S'ha eliminat el Event correctament.			

Nom	Generar horaris per les assignatures grupals		
Data creació	10-02-2014	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Sistema
Descripció: Forma part del manteniment d'un horari. Genera una combinació d'horaris aleatòria, puntua, escull els millors i torna a fer el procés.			
Precondició: Les dades a modificar són conegudes.			
Escenari Principal 1. Fins que es trobi una solució molt bona: 1.1. Per cada assignatura grupal. 1.1.1.Assignar un dia, hora, professor que la impartirà, i aula. 1.1.2.Puntuar la combinació. 1.2. Fper. 1.3. Per cada assignatura de la solució escollida: 1.3.1.Crear un Event amb la informació associada (assignatura, dia, hora, professor i aula). 1.4. Fper. 2. Mostrar horari.			
Postcondició: S'ha generat l'horari correctament.			

Nom	Crear Event.		
Data creació	10-02-2014	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Personal Acadèmic
Descripció: Donada una informació, es crea un Event.			
Precondició: Coneixem la informació i l'event no existeix.			
Escenari Principal 1. Es Crea l'event amb la informació associada. 2. Si s'ha especificat professor: 2.1. Buscar professor. 2.2. Assignar Event a professor. 3. FSi. 4. Si s'ha especificat alumne: 4.1. Buscar alumne. 4.2. Assignar Event a alumne. 5. FSi. 6. Si s'ha especificat assignatura. 6.1. Buscar assignatura. 6.2. Assignar Event a assignatura. 7. FSi			
Postcondició: S'ha creat l'event correctament.			

8.1.2.2. Fitxes de casos d'ús de l'actor Personal Direcció

Nom	**Donar d'alta personal.**		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2013	**Actor**	Direcció.
Descripció: Es demanen les dades necessàries per crear un nou usuari corresponent al col·lectiu del personal, i es guarden.			
Precondició: Personal no donat d'alta.			
Escenari Principal 1. Entrar dades del personal. 2. Crear personal amb les dades.			
Postcondició: Personal donat d'alta.			

Nom	**Donar de baixa personal.**		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Direcció.
Descripció: S'elimina un personal del sistema.			
Precondició: Personal donat d'alta.			
Escenari Principal 1. Entrar dades personal. 2. Buscar personal. 3. Eliminar personal.			
Postcondició: Personal eliminat.			

Nom	**Obrir procés de matricula.**		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Direcció
Descripció: Es fa possible que el personal d'administració i direcció pugui realitzar matricules.			
Precondició: Sistema d'avaluació diferent als existents.			
Escenari Principal 1. Crear un procés que permeti la modificació de les dades relacionades amb les matricules			
Postcondició: Procés obert.			

Nom	Obrir període d'avaluació.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Direcció.
Descripció: Es fa possible que el personal docent i direcció pugui realitzar l'avaluació dels seus alumnes.			
Precondició: Sistema d'avaluació donat d'alta.			
Escenari Principal 1. Crear un procés que permeti la modificació de les dades relacionades amb l'expedient.			
Postcondició: Obert període.			

Nom	Tancar període d'avaluació.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Direcció.
Descripció: Es tanca el procés de d'avaluació, de manera que no es deixarà crear o modificar cap expedient més.			
Precondició: Professor donat d'alta.			
Escenari Principal 1. Esborrar el procés que permetia la modificació de les dades relacionades amb l'expedient.			
Postcondició: Període tancat.			

Nom	Modificar informació personal.		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Direcció.
Descripció: Seleccionat un personal, es modifica la seva informació			
Precondició: Professor donat d'alta.			
Escenari Principal 1. Buscar personal. 2. Modificar la informació del personal. 3. Guardar personal.			
Postcondició: Informació modificada.			

8.1.2.3 Fitxes de casos d'ús de l'actor Personal Docent

Nom	**Entrar notes dels alumnes.**		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Docent.
Descripció: Un cop escollit el sistema d'avaluació que es farà servir, el docent pot procedir a assignar notes als seus alumnes.			
Precondició: Alumnes donats d'alta.			
Escenari Principal 1. Si es SA1 (Americà): 1.2 Entrar lletra i observació. 2. Si es SA2 (Europeu): 2.2 Entrar decimal i observació. 3. Si es SA3 (Sistema de frases): 3.2 Entrar codi frase i observació. 4. En tots els casos: guardar informació. 5. FSi.			
Postcondició: Notes guardades.			

Nom	**Modificar les notes d'un alumne.**		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Docent
Descripció: Donat un alumne i l'assignatura, es pot modificar la nota guardada sempre que no s'hagi tancat l'acte.			
Precondició: Alumne donat d'alta i amb qualificació assignada a l'assignatura.			
Escenari Principal 1. Obtenir estat convocatòria. 2. Si estat = tancada llavors 2.2 Mostrar missatge i no permetre modificar la nota. 3. Altrament 3.2 Guardar nova nota. 4. FSi.			
Postcondició: Nota actualitzada.			

Nom	**Consultar informació d'assignatures.**		
Data creació	17-09-2013	**Visió**	Usuari.
Data modificació	08-06-2014	**Actor**	Docent
Descripció:			
Escenari Principal 1. Mostrar assignatures assignades al professor que ho demana.			
Postcondició: Es mostren les dades.			

8.2 Disseny proposat

8.2.1 Diagrama de classes

En aquest apartat, s'explicarà el diagrama de classes que es va seguir per realitzar la implementació.

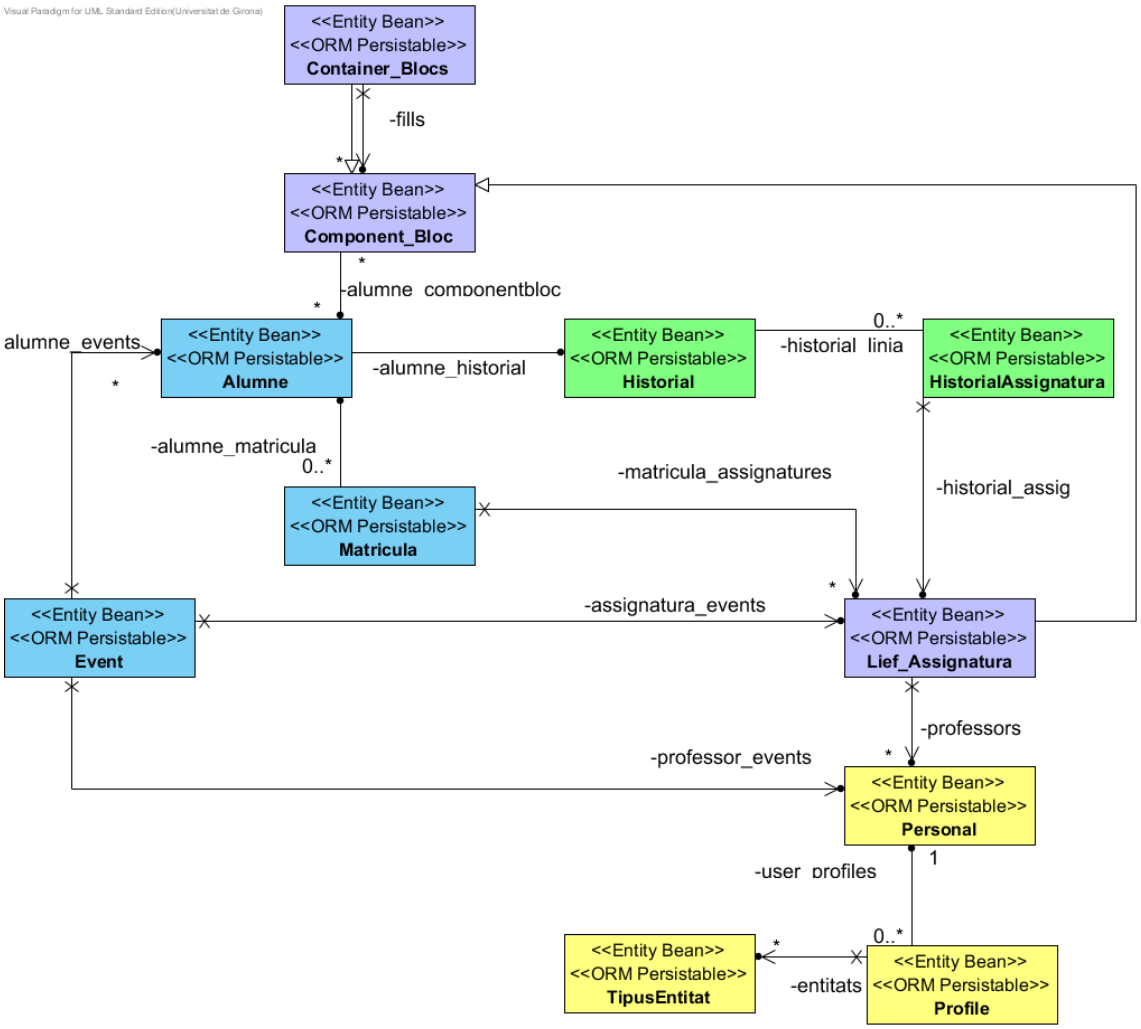


Figura 20: Diagrama de classes

Les classes de diferents colors indiquen diferents "blocs". S'ha classificat així per poder-ho explicar més fàcilment.

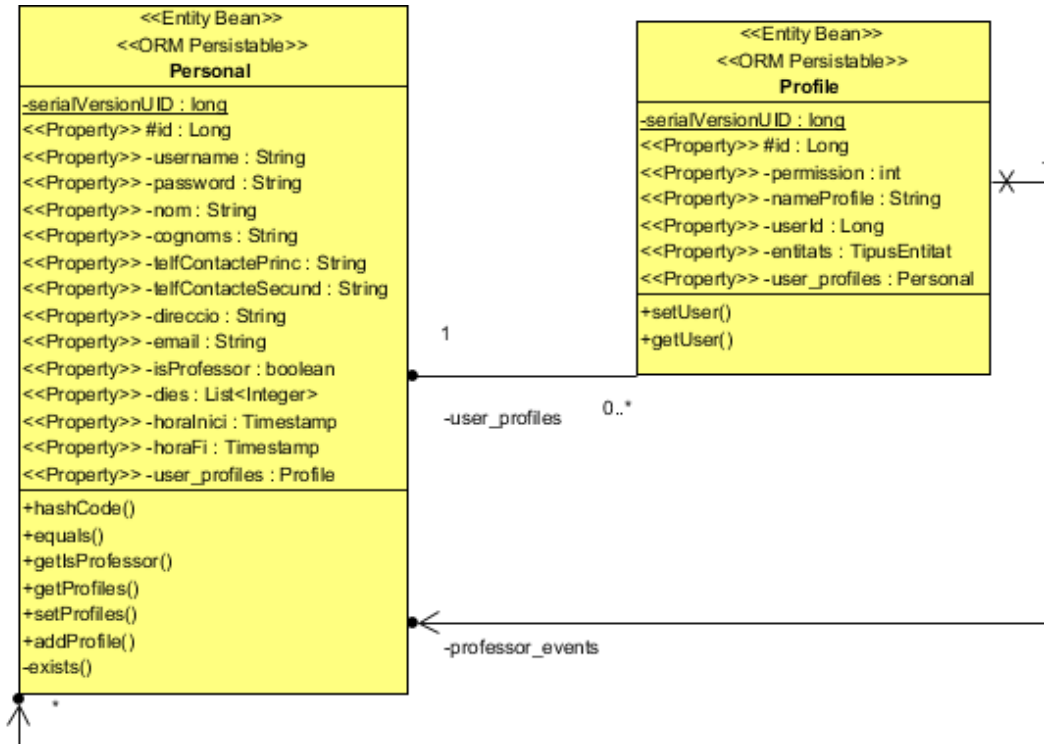


Figura 21: Diagrama de classes, Personal

Personal: El seu nom ja descriu el que representa: el personal de l'aplicació. El terme personal agrupa els 3 actors que comentaven anteriorment: Personal Administratiu, Personal Docent i Personal de Direcció. Aquesta classe guarda tota la informació necessària per tal de posar-se en contacte amb el personal (telèfons i email, direcció).

No cal que ens proporcionin tota la informació, però com a mínim hi hauria d'haver un telèfon mòbil i una direcció de correu electrònic.

Com a informació addicional, en cas que fos un professor (del col·lectiu de personal docent), caldria guardar també els dies que pot venir, i l'hora inici i final de la seva jornada laboral.

Profile: Aquesta classe s'encarrega de guardar un perfil d'usuari, per poder tenir un control de les accions que fan els usuaris de l'aplicació.

El disseny permet que un personal tingui varis perfils associats. Per entendre perquè cal, considerin el següent cas: Es vol que una secretària pugui fer el procés de matricula, però en canvi que no pugui eliminar-la. En canvi, pel que fa a les assignatures, pot crear-les, consultar-les, modificar-les i esborrar-les.

De manera que hauria de tenir com dos perfils diferents, un que li permetés saber que només pot crear una matricula, i un altre que li fes saber que pot crear, modificar, llegir i esborrar les assignatures.

Una altre part del diagrama de classes que s'hauria de comentar és:

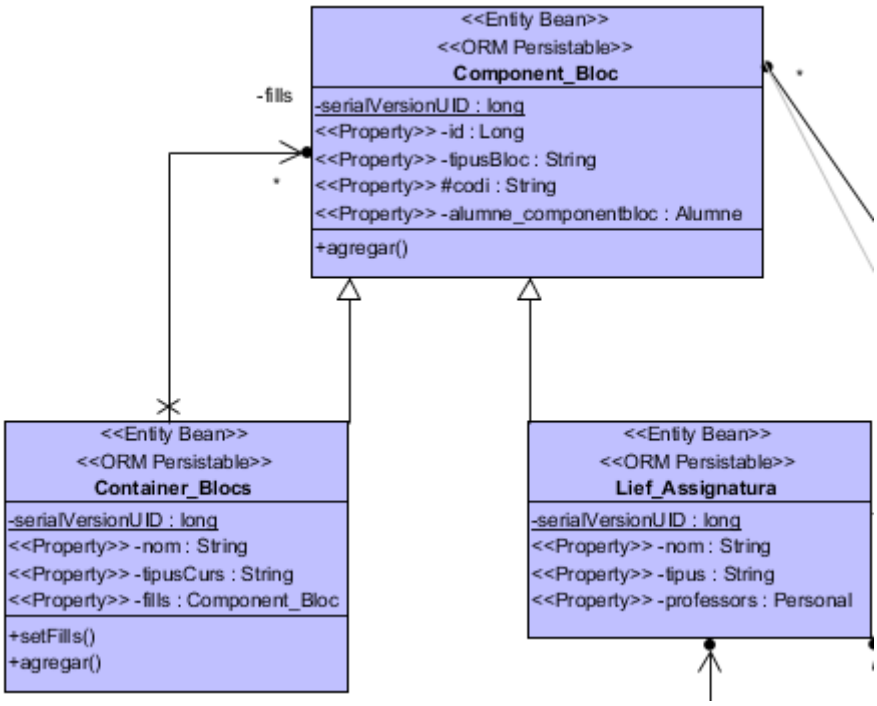


Figura 22: Diagrama de classes, Composite

Com es pot apreciar, s'ha aplicat el patró composite. Per que s'entengui:

Container_Blocs: Seria l'equivalent a un Curs. Un curs conte varies assignatures, però també altres cursos, d'aquí la relació amb el component_bloc.

Lief_Assignatura: Seria l'equivalent a una assignatura.

Els alumnes, l'expedient i la matricula es troben relacionats de la següent forma:

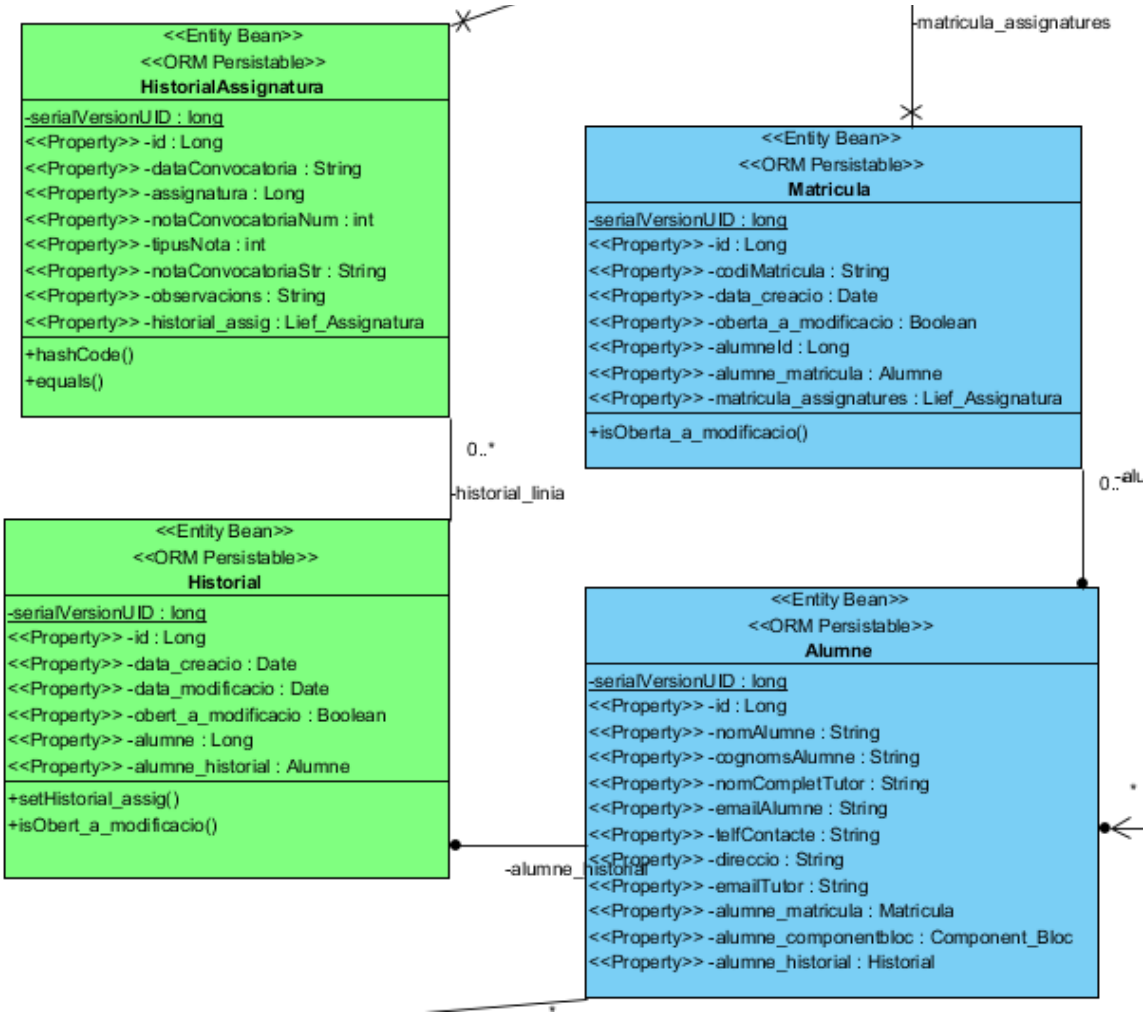


Figura 23: Diagrama de classes, Alumne, Matricula i Historial

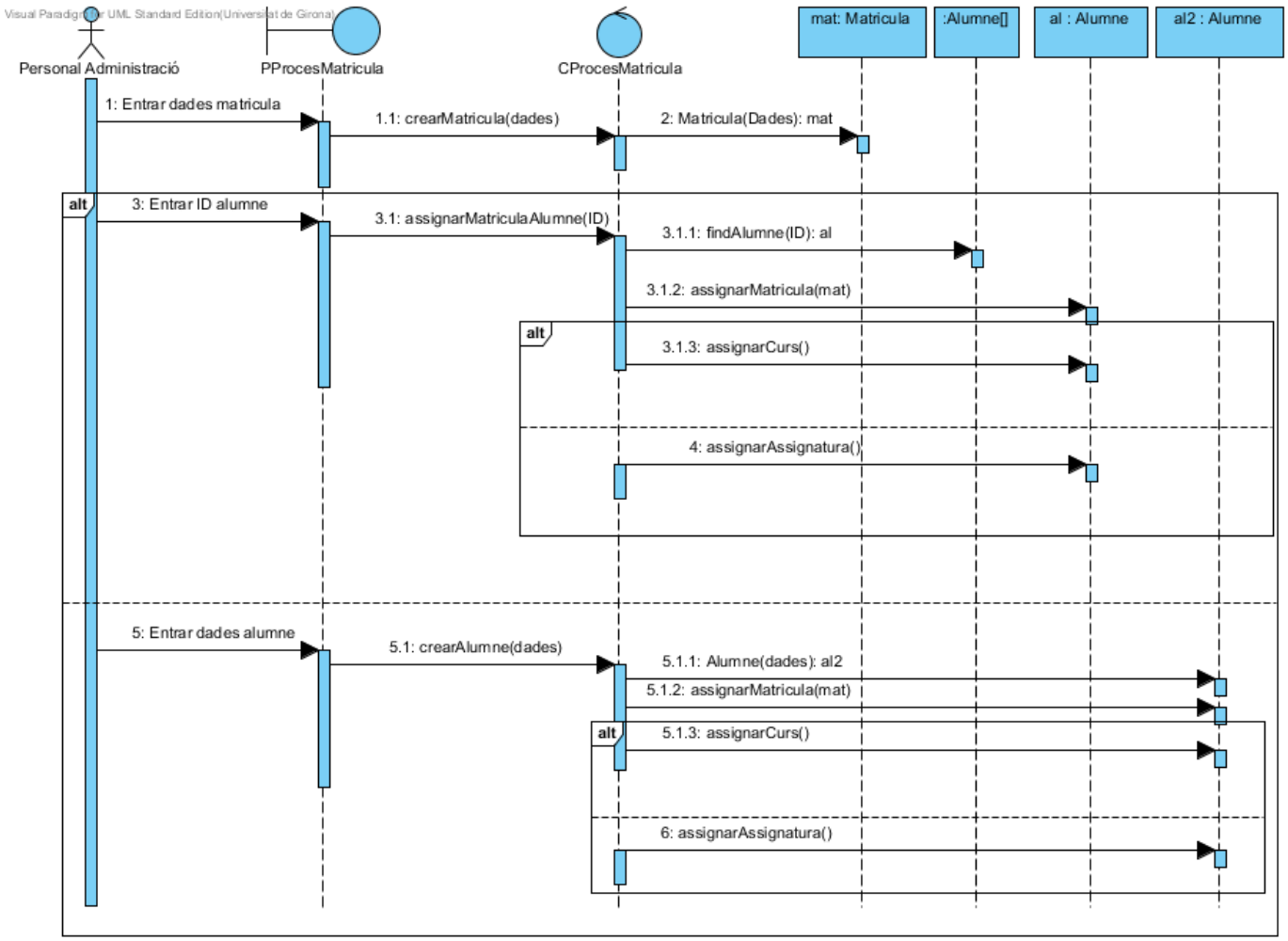
Alumne està relacionat amb un historial, i amb una o varies matricules. **Historial**, per la seva banda, esta relacionat amb la corresponent línia de historial (**HistorialAssignatura**), que al seu torn es relaciona amb l'assignatura a la que es posa la nota.

Per últim, hi ha la classe Event, que relaciona per un dia i hora concrets, una assignatura, un professor i un alumne.

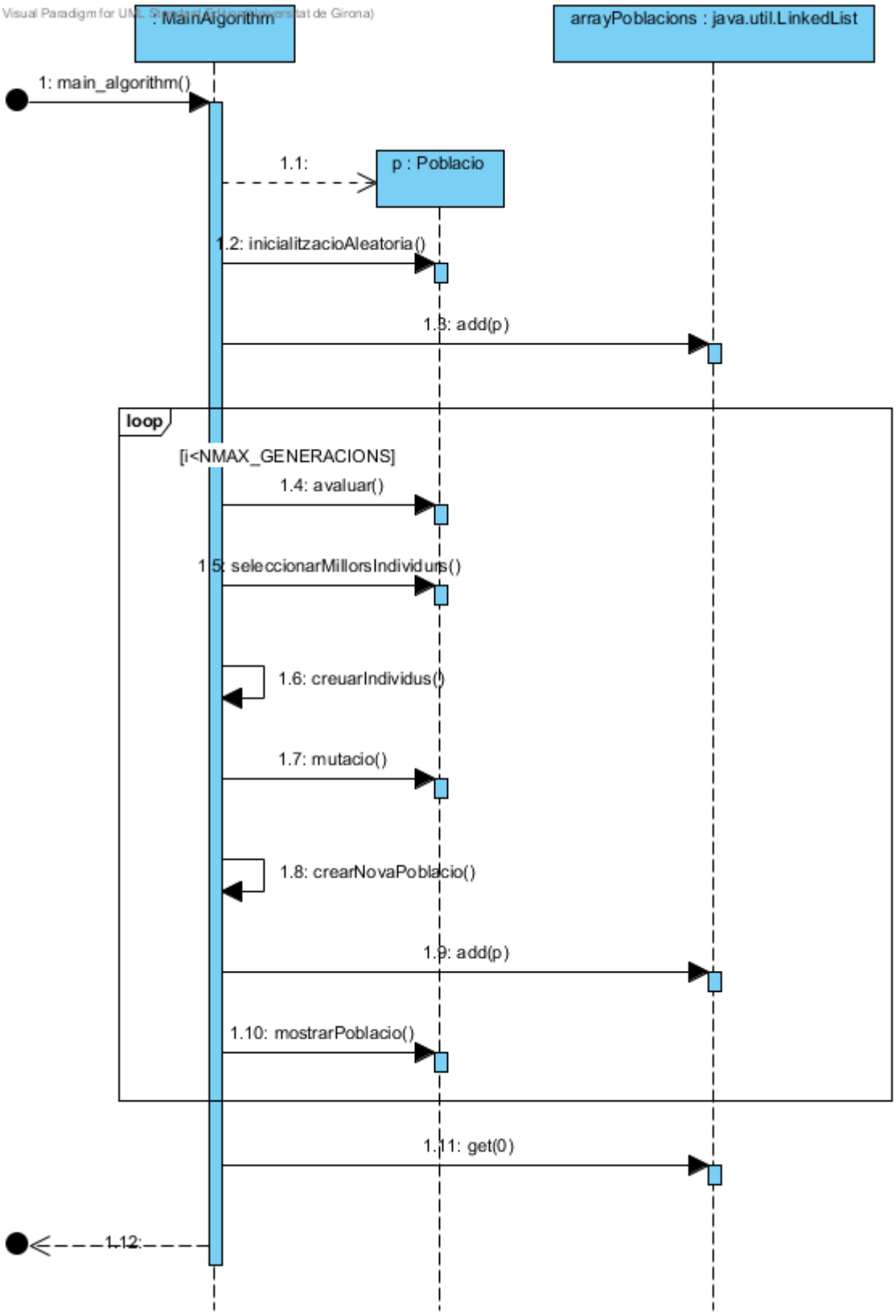
8.2.2 Diagrames de seqüència

A continuació s'exposaran dos diagrames de seqüència. Només s'han fet dos degut a que són les operacions més complexes que s'han hagut de realitzar. Les altres operacions són senzilles i seria poc útil posar el seu diagrama de seqüència.

8.2.2.1 Procés matricula



8.2.2.2 Algoritme Genètic



8.2.3 Disseny API

En aquest apartat s'explicarà primer uns principis que s'ha seguit per tal de dissenyar l'API del projecte, i s'exposarà el resultat final.

8.2.3.1 Filosofia de disseny.

Hi ha una sèrie de convencions que s'han agafat com a bones pràctiques alhora de fer un bon disseny de les URIs que serviran als clients per comunicar-se amb l'API de servidor.

Es important però entendre el concepte de **Recurs**. En termes de aplicacions **RESTful**, seria allò que s'ha de poder accedir a través de la web. Cal tenir-los identificats, donat que en base aquests recursos es podran definir unes URIs que facin una correcta referència a aquests.

Una vegada definits els recursos, es poden aplicar certs criteris referents al disseny de URIs, per facilitar la programació i la integració de possibles clients a una API. Alhora de dissenyar l'API d'aquest projecte s'han tingut en compte els següents punts:

1. Hauria de ser curta, per tal que sigui senzilla d'escriure i recordar.
2. 'Up-the-tree': Significa que si una URI esta composta per, per exemple: http://localhost:8080/pfg/rest/personal/1, si s'esborra 1, s'obtindrà tot el personal donat d'alta. S'ha de veure una URI com una estructura d'arbre, a mesura que obres amb "/" crees un altre nivell, que filtra els resultats, que obtindries abans.
3. Ha de descriure el recurs, i també ha de ser predictible.
4. Noms, res de verbs.
5. Qualsevol cosa que vingui després del ? serveix per consultar/buscar recursos (exclusivament)

8.2.3.2 API RESTful del Servidor

La part servidor de l'aplicació es pot dividir en 4 grans blocs REST, corresponents al personal de l'aplicació, la part acadèmica,, l'alumnat (amb les seves corresponents matricules i expedients), i els horaris. Aquests 4 blocs es correspondrien als 4 principals recursos que poden ser accedits.

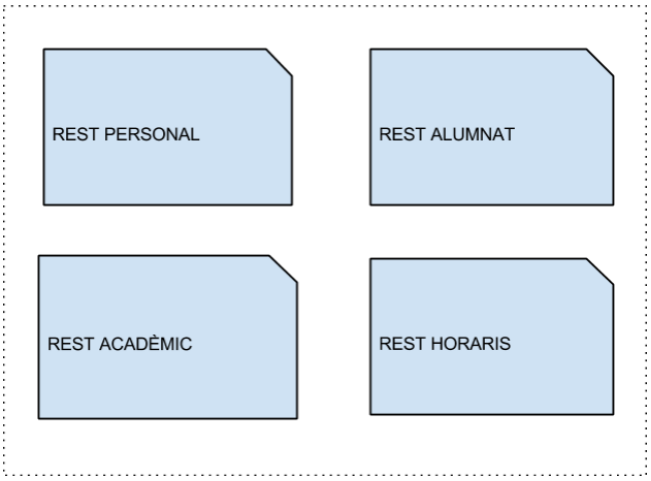


Figura 24: Blocs REST

Les URI del bloc REST referent a la gestió del personal són les següents:

HTTP	URI	Descripció
GET	/personal	Obté tots el personal donat d'alta.
POST	/personal	Dona d'alta un personal. Accepta per paràmetre el nom, cognoms, 2 telèfons de contacte (requereix mínim 1), adreça i email (es requereix el email).
GET	/personal/{id}	Obté la informació del personal amb identificador ID.
POST	/personal/{id}	Actualitza la informació del personal amb identificador ID.
DELETE	/personal/{id}	Esborra el personal amb identificador ID.
GET	/personal/{id}/could	Obté els permisos d'un usuari amb identificador ID.
POST	/personal/{id}/could	Assigna permisos a un usuari.
POST	/personal/{id}/could/{id2}	Modificar el permís amb identificador ID2 si s'ha especificat, de l'usuari amb identificador ID.
GET	/personal/{id}/events	Retorna una llista d'identificadors del recurs Event assignat a un usuari.

Les URI del bloc REST referent a la gestió del acadèmica són les següents:

HTTP	URI	Descripció
POST	/academia/curs	Crea un curs.
POST	/academia/assignatura	Crea una assignatura.
POST	/academia/assignatura/{id}/teacher	Assigna un professor a una assignatura amb identificador ID.
POST	/academia/curs/{id}/assignatura	Assigna una o varies assignatures a un curs.
GET	/academia/curs/	Obté un/s cursos, depenent d'uns paràmetres.
GET	/academia/curs/assignatura	Obté els cursos que contenen una assignatura amb un codi que es passa per paràmetre.
GET	/academia/curs/assignatura/{id}	Obté els cursos que contenen una assignatura amb un identificador ID que es passa per paràmetre.
GET	/academia/assignatura/	Obté una/es assignatures, depenent d'uns paràmetres.
POST	/academia/curs/{id}	Modificar la informació d'un curs.
POST	/academia/assignatura/{id}	Modificar la informació d'una assignatura.
DELETE	/acadèmia/curs/{id}	Elimina un curs.
DELETE	/academia/assignatura/{id}	Elimina una assignatura.

Les URI del bloc REST referent a la gestió dels alumnes, les matricules i els expedients són les següents:

HTTP	URI	Descripció
GET	/alumnat	Obté tots els alumnes donats d'alta.
GET	/alumnat/{id}/components	Obté els cursos o assignatures als que esta matriculat.
GET	/alumnat/{id}/matricula	Obté les matricules de l'alumne.
GET	/alumnat/{id}/historial	Obté l'expedient de l'alumne.
POST	/alumnat	Crea un alumne.
POST	/alumnat/{id}/matricula	Crea una matricula per l'alumne amb identificador ID.
POST	/alumnat/{id}/historial/	Crea historial per l'alumne si no en té cap.
POST	/alumnat/{id}/historial/notes	Crea una nova entrada a l'historial, assignant una nota a una assignatura, a més d'altres dades.
POST	/alumnat/{id}	Modifica informació d'un alumne.
DELETE	/alumnat/{id}	Dóna de baixa un alumne.

Les URI del bloc REST referent a la gestió dels horaris són:

HTTP	URI	Descripció
GET	/events	Obté tots els events. Es poden obtenir per una assignatura o un alumne o una aula en concret,
GET	/events/{id}	Obté informació d'un Event en concret.
POST	/events/	Crea un Event que relaciona alumne, assignatura i aula. L'assignatura ha de ser de tipus individual.
POST	/events/grupals	Per totes els assignatures grupals, crea un horari a partir d'un algoritme genètic, i ho retorna al usuari.
POST	/events/{id}	Modifica la informació d'un Event.
DELETE	/events/{id}	Elimina un Event.

A més, hi ha dues URIs més que permeten la autentificació d'usuaris:

HTTP	URI	Descripció
POST	/auth/0	Login dels usuaris.
POST	/auth/1	Logout dels usuaris.

9 Implementació i proves

9.1 Client

La part client s'ha implementat bàsicament amb HTML i JavaScript. Ara bé, el fet de utilitzar AngularJS ha fet que els fitxers HTML acabessin tenint un conjunt de directives que normalment no hi serien.

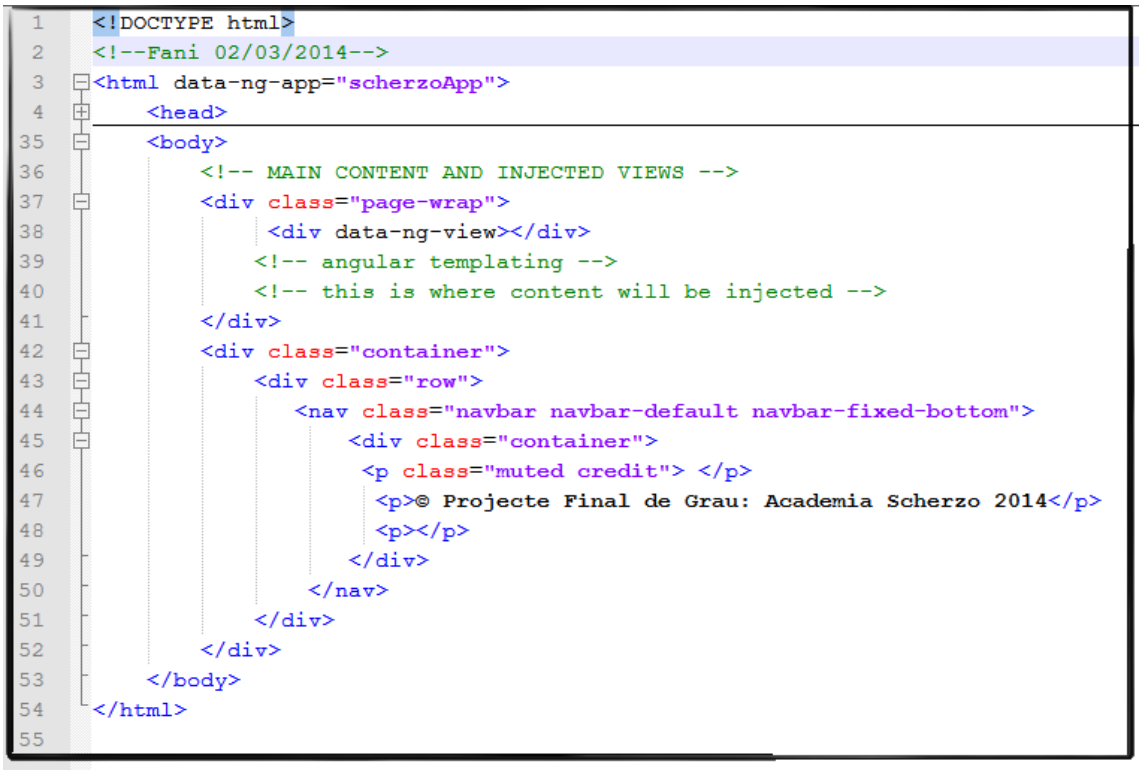
A continuació s'explicarà les directives més importants utilitzades, les seves funcions, i un exemple de com esta implementat en aquest projecte.

Els conceptes més importants a tenir en compte que es tractaran a continuació

Concepte	Descripció
Template	HTML amb marques addicionals.
Directives	Extendre el llenguatge HTML amb atributs i elements personalitzats.
Model	La informació mostrada al usuari en una vista, sobre la qual l'usuari interactua.
***Scope* o àmbit**	Context on el model és guardat de manera que els controladors, les directives i les expressions poden accedir a ell.
Expressions	Variables d'accés i funcions del àmbit.
***Filter* o filtres**	Formateja el valor d'una expressió per mostrar-ho al usuari.
***View* o vista**	Allò que l'usuari veu.
Data Binding	Data sincronitzada entre el model i la vista.
***Controller* o controlador**	La lògica de negoci que hi ha darrera les vistes.
Module	Un contenidor per diferents parts d'una aplicació, incloent controladors, serveis, filtres, directives, etc.

9.1.1 Vistes

Les vistes de l'aplicació s'injecten dins un fitxer que es diu *index.html*. Aquest fitxer és el següent:



```
<!DOCTYPE html>
<!--Fani 02/03/2014-->
<html data-ng-app="scherzoApp">
    <head>
    <body>
        <!-- MAIN CONTENT AND INJECTED VIEWS -->
        <div class="page-wrap">
            <div data-ng-view></div>
            <!-- angular templating -->
            <!-- this is where content will be injected -->
        </div>
        <div class="container">
            <div class="row">
                <nav class="navbar navbar-default navbar-fixed-bottom">
                    <div class="container">
                     <p class="muted credit"> </p>
                      <p>© Projecte Final de Grau: Academia Scherzo 2014</p>
                      <p></p>
                    </div>
                 </nav>
            </div>
        </div>
    </body>
</html>
```

Figura 25: Fitxer index.html

Com es pot observar, en aquest fitxer ja apareixen les primeres directives angular: `ng-app` i `ng-view`[14].

La directiva `ng-app` serveix per definir el context de l'aplicació; crea un contenidor anomenat `scherzoApp` dintre el qual es guardarà tota la informació dels controladors, les vistes, i les directives (entre altres).

Això s'aconsegueix gràcies a una prèvia configuració:

```
//Define an angular module for our app
var scherzoApp = angular.module('scherzoApp', [
    'ngRoute',
    'ui.calendar', 'ui.bootstrap',
    'scherzoControllers'
]);
```

Figura 26: Configuració d'un mòdul.

[14] Cal comentar que per temes de configuració i integració entre IDEs i angular, s'ha agafat el costum general d'afegir la paraula data a la majoria de les directives.

Com es pot veure, es defineix un mòdul que es diu `scherzoApp`, on assigna diverses opcions: `ngRoute`, `ui.calendar`, `ui.Bootstrap` i `scherzoControllers`.

D'aquestes opcions en parlarem més endavant.

L'estructura dels projectes que es fan amb angular situen aquesta configuració en un fitxer que es diu *app.js* (veure estructura de directori per situar-se). Però no és més que una convenció. Per tal de seguir al màxim aquesta estructura i facilitar l'aprenentatge d'aquesta nova forma de programar per web s'ha decidit utilitzar-ho també.

L'altra directiva `ng-view` serveix per injectar les vistes corresponents. Al projecte hi ha les següents vistes:

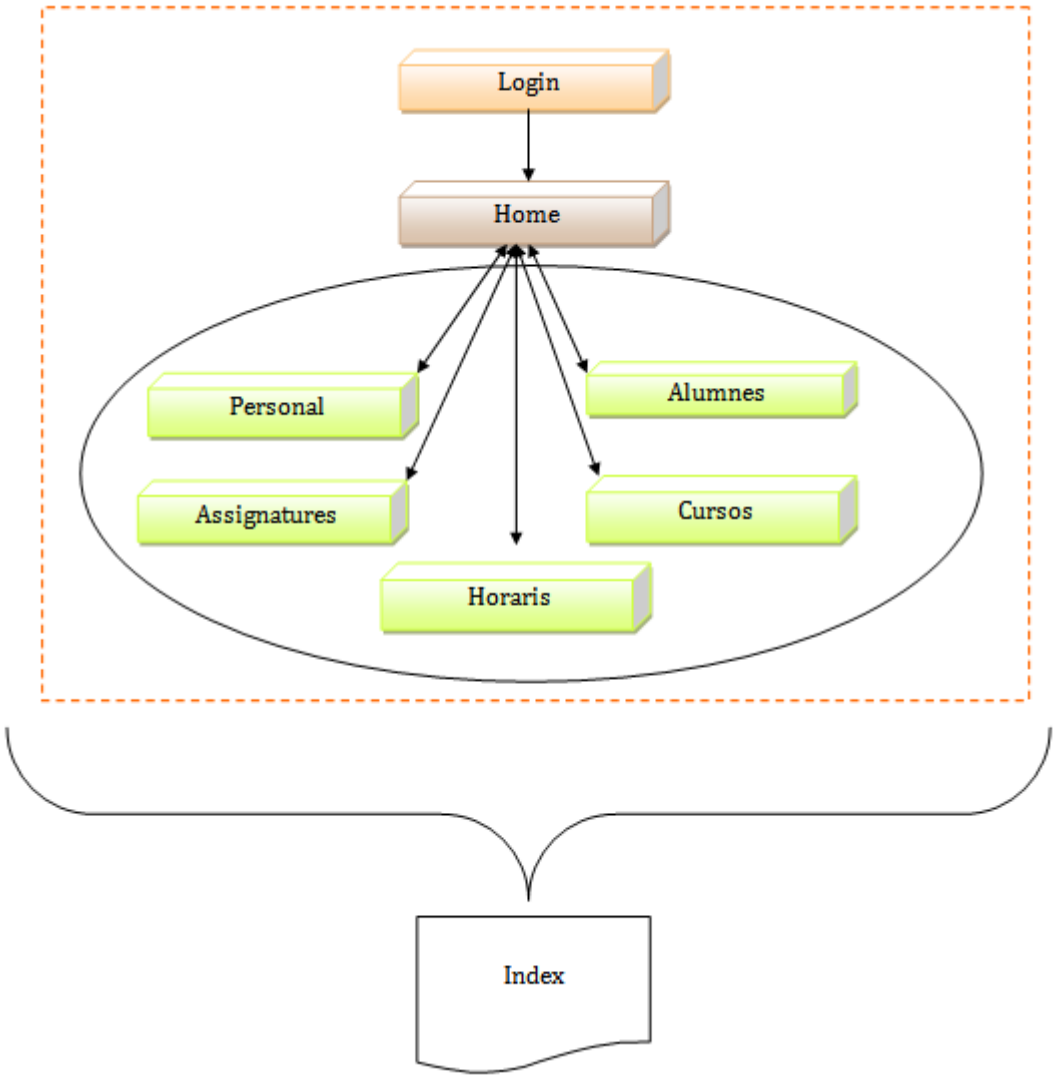


Figura 27: Gràfic amb les vistes del projecte.

Des de la vista *Login*, es pot accedir directament a la Home, des de el *Home* es pot accedir a través d'un menú a les altres 5 vistes que es veuen en el gràfic anterior.

La pregunta ara deu ser, i com s'administra quina vista s'injecta?

Tot es fa per mitjà de la directiva `ngRoute`:

```
scherzoApp.config(["$routeProvider", function($routeProvider)

    $routeProvider
        .when('/',
        {
            templateUrl: 'partials/login.html',
            controller: 'loginController'
        })
        .when('/home',
        {
            templateUrl: 'partials/home.html'
        })
```

Figura 28: Route Provider, injector de vistes

Al module scherzoApp s'assigna un nou parametre de configuració anomenat *RouteProvider*, el qual segons la url que s'escrigui al navegador injectarà una vista o una altre. Hi ha la opció d'assignar el controlador en aquest context.

De manera que, quan s'entra a la web el primer que es veu es la vista del *login.html* injectada al *index.html,* i tot el que hi ha dintre aquesta vista tindrà la lògica de negoci definida al loginController.

El següent fragment prové de la vista que proporciona la interficie que gestiona el personal de l'aplicació:

```
<div id="personal" data-ng-controller="personalController">
    <div class="row">
        <nav class="navbar navbar-default navbar-static-top" role="navigation">
            <div class="container">
                <div class="navbar-header">
                    <a class="navbar-brand" href="#">Personal</a>
                </div>
                <div class="navbar-collapse collapse">
                    <ul class="nav navbar-nav">
                        <li data-ng-repeat="opt in opts">
                            <a data-ng-href="#/{{opt.nom | lowercase}}">{{opt.nom}}</a>
                        </li>
                    </ul>
                </div>
```

1 2 3 4

Figura 29: Fragment de codi de *personal.html*

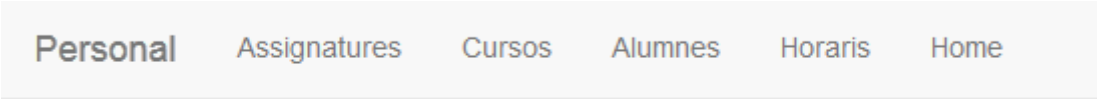
El que aquí es veu es un fragment del codi que s'encarrega de la gestió de les pestanyes de navegació. Aquesta informació prové del `personalController` (a la imatge anterior, pas 1). Els controladors es poden assignar a les vistes de dues maneres, quan s'accedeix a elles amb el *RouteProvider*, o assignant-les directament al div.

Les dues tenen avantatges i inconvenients. Si es fa a través del *RouteProvider*, t'estalvies haver-ho de definir al fitxer HTML, i té efectes a tot el fitxer. En canvi, si es defineix als div, es podrien tenir diversos controladors en un mateix fitxer.

A la part que està marcada amb un 2, es pot veure un exemple de expressió. És molt freqüent veure les expressions acompanyades de la directiva **ng-repeat**. Aquesta directiva fa textualment el que diu: repetir. En concret el que fa es repetir el div que conté les vegades que faci falta. Aquestes "vegades que faci falta" ve donada per l'expressió que segueix a la directiva. La variable opts, que es troba definida al àmbit del `personalController`, és una col·lecció d'informació. L'expressió "opt **in** opts" fa un *loop* pel qual, per cada element de la llista, l'aplica al div que hi ha el `ng-repeat.`

Després es defineix un href que ve seguit d'una altre expressió que aplica un filtre de *lowercase*[15] al element opt.nom (l'atribut nom dintre del opt definit anteriorment, al pas 3), i per últim aplica la propietat de *data binding* (de fet qualsevol cosa que vagi entre {{ }} és *data binding*) i mostra el opt.nom per pantalla (pas 4).

El resultat d'aquest fragment de codi és el següent:



9.1.2 Controladors

Vist el més important de la gestió de les vistes, a continuació s'explicaran els controladors del projecte.

[15] Lowercase: Funció que posa en minúscula totes les lletres d'una paraula.

La pròpia documentació d'Angular explica diferents aspectes a tenir en compte a l'hora de crear i utilitzar controladors.

Els més importants que he tingut en compte són:

1. No definir funcions que representin controladors en l'àmbit global de l'aplicació, és millor fer-ho de la següent manera:

```
angular.module('scherzoControllers').controller("personalController", function($scope,$http) {
    $scope.opts=[
        {nom :"Assignatures", on:"isAssignatures"},
        {nom :"Cursos", on:"isCursos"},
        {nom :"Alumnes", on:"isAlumnes"},
        {nom :"Horaris", on:"isHoraris"},
        {nom :"Home", on:"isInici"}

    ];
    $scope.dies="";
    $scope.personalId="";
    $scope.people=[];
    $scope.personal=[];
    $scope.addPersonal=function(){
```

Figura 30: Inicialització d'un controller.

   Al mòdul `scherzoControllers` se l'hi assignen els controladors. També es podria fer a través de la variable global `scherzoApp`, però si es fes amb aquesta, llavors el controladors estaria assignat a tot el contenidor de l'aplicació, i per mantenir una estructura modular he preferit que es guardés en un mòdul intern de controladors anomenat `scherzoControllers`. Això és més aviat decisió pròpia que no una pauta de bones pràctiques a seguir.
2. Utilitzar un controlador només per donar comportament a un àmbit en concret.

Els controladors que he necessitat per aquesta aplicació són 7:

- `navbarController`: és el control de pestanyes de la vista *home.html.*
- `personalController`: conté tota la lògica de negoci relacionada amb la vista *personal.html.*
- `assignaturesController`: conté tota la lògica de negoci relacionada amb la vista *assignatures.html.*
- `cursosControllers`: com us podeu imaginar, conté la lògica de negoci relacionada amb la vista dels cursos.
- I el mateix passa amb `alumnesController` i `historialControllers`.

9.1.3 Models

No hi ha en si cap classe o tipus d'objecte que es faci servir exclusivament per guardar les dades dels models relacionades amb les vistes.

Normalment, es defineix al àmbit del controlador que conté la lògica de negoci de la vista. Ara bé, si s'ha de compartir informació d'un model entre dos controladors diferents, s'aconsella utilitzar els **Services**.

Un *service* o servei és un contenidor per el codi que es repeteix al llarg de l'aplicació. D'aquesta manera, es podria utilitzar per compartir els models entre varis controladors diferents o per diferents vistes. Al llarg de l'aplicació s'han utilitzat alguns, però el més important és el que ens proporciona les eines per comunicar-nos amb el servidor: el servei $http (als serveis normalment se'ls diferència per posar "$" al davant).

El serveis $http és el component que facilita la comunicació amb la part servidor via l'objecte XMLHttpRequest[16] dels navegadors o amb JSONP[17].Per utilitzar aquest servei només cal utilitzar la següent estructura:

```
$http({
        url: 'rest/personal/',
        method: 'POST',
        data: { data aquí },
        headers: {
             'Content-Type': 'application/x-www-form-urlencoded'
        }
}).success(function (data) {
// Tot a anat bé
}).error(function(data, status, headers, config) {
// Hi ha hagut algun error
});
```

Figura 31: Estructura d'una petició HTTP.

[16] **XMLHttpRequest**: També se'n diu XMLHTTP, és una interfície que s'utilitza per realitzar peticions HTTP i HTTPS a servidors web. Per transferir les dades s'utilitza qualsevol codificació basada en text, incloent: text pla, XML, JSON, HTML i codificacions particulars específiques.
[17] **JSONP**, JSON amb Padding és una tècnica de comunicació utilitzada en els programes JavaScript per a realitzar crides asíncrones a dominis diferents. JSONP és un mètode concebut per a suplir la limitació de AJAX entre dominis, que únicament permet realitzar peticiones a pàgines que se troben sota el mateix domini i port per raons de seguretat.

La figura 20 exposa a grans trets el que cal per enviar una petició de tipus POST a la URL "`rest/personal`". Degut a que és de tipus post cal especificar al *header* (la capçalera de la petició) que és '`application/x-www-form-urlencoded`'.

9.1.4 Directiva Ui-Calendar i llibreria FullCalendar

La llibreria FullCalendar[18] és un plugin de jQuery molt complet creat per Adam Shaw que proporciona una gestió bastant acurada dels esdeveniments d'un calendari, amb diverses vistes (dia, setmana, més, dia amb hores, setmana amb hores, etc).

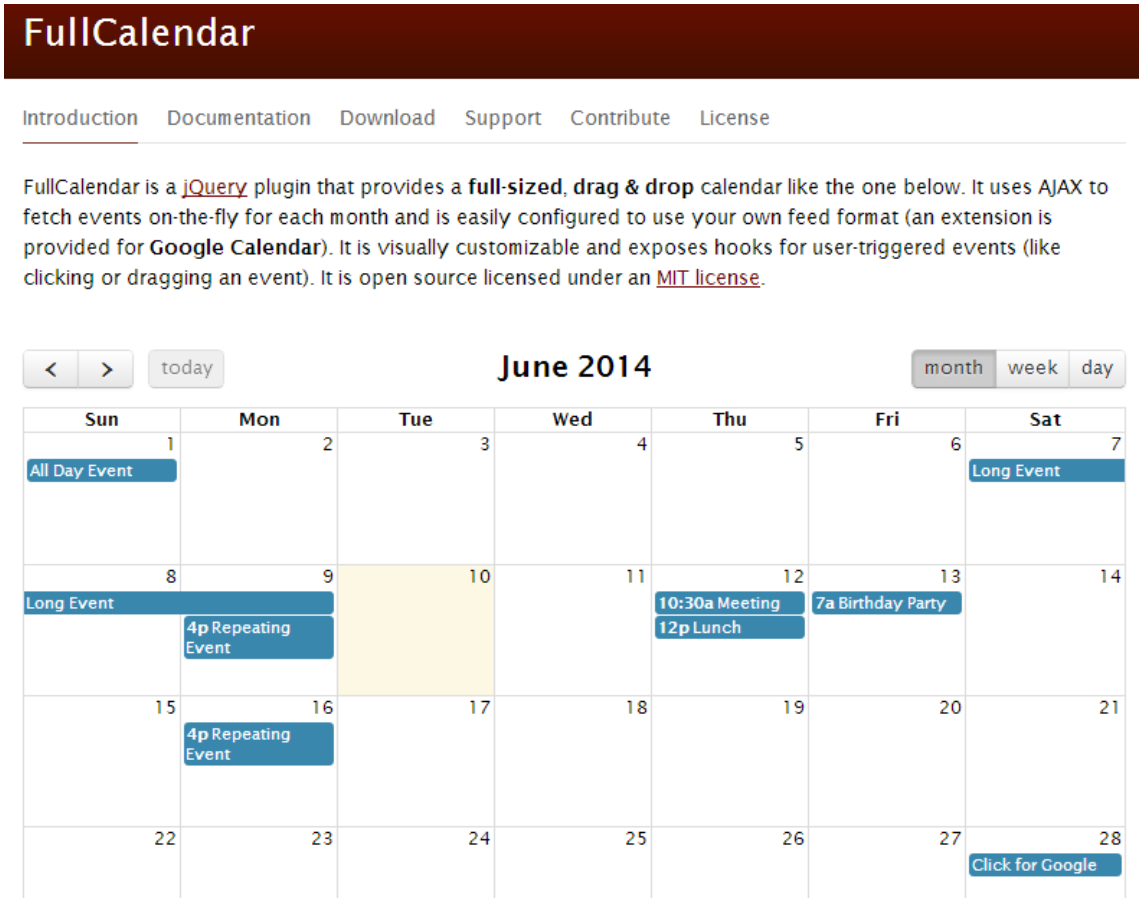


Figura 32: Captura de pantalla de la web de FullCalendar.

[18] Web oficial de FullCalendar: arshaw.com/fullcalendar/

Permet també personalitzar els events, és totalment compatible amb Bootstrap i es possible tenir diferents fonts de dades que proporcionin els esdeveniments.

Aquestes fonts poden ser:

- Una petició AJAX a una URL que conté la direcció d'un Google Calendar pùblic, o bé una URL qualsevol que proporcioni esdeveniments.
- Un fitxer JSON.

Ara bé, Angular necessita de directives per a que funcioni tots els comportaments, de manera que si es vol utilitzar aquest plugin cal adjuntar a les llibreries d'Angular la directiva anomenada Ui-Calendar (existeixen moltes directives que no venen per defecte al framework que es descarrega des de la pàgina oficial de Angular, però que són completament compatibles).

El procediment per crear l'horari amb les assignatures grupals és el següent:

1. A la vista *horaris.html* s'afegeix aquestes línies:

```
<div class="row">
    <div class="col-md-12">
        {{alertMessage}}
        <div ui-calendar="uiConfig.calendar" ng-model="eventSources"></div>
    </div>
</div>
```

2. Es crida el mètode crearEventsGrupals del HorarisController:

```
$scope.crearEventsGrupals=function(){
    $http({
         url: 'rest/events/grupals',
         method: 'POST',
         headers: {
             'Content-Type': 'application/x-www-form-urlencoded'
         }
     }).success(function (data) {
         for(var i = 0; i < data.length; i++) {
           $scope.events.push(data[i]);
           //debugger;
         }
        //$scope.events = data;
     }).error(function(data, status, headers, config) {

     });
};
```

Aquest mètode fa una petició rest a la URL que es veu a la figura, i parseja el resultat guardant-lo en una col·lecció d'events.

3. Els events es guarden en la següent col·lecció:

```
/* event sources array*/
$scope.eventSources = [$scope.events];
```

`eventSources`, és el que proporciona les dades al div del pas 1. Podria haver diferents *sources* (un calendari de Google, un altre URL rest).

4. A part de tot això, hi ha una configuració:

```
/* config object */
$scope.uiConfig = {
  calendar:{
    height: 600,
    editable: true,
    header:{
      left: 'month agendaWeek agendaDay',
      center: 'title',
      right: 'today prev,next'
    },
    dayClick: $scope.alertOnEventClick,
    eventDrop: $scope.alertOnDrop,
    eventResize: $scope.alertOnResize,
    firstDay:1,
    weekends:false
  }
};
```

Que permet personalitzar el calendari.

Els events per a que es guardin correctament cal que tinguin el següent format:

```
{
      "id"   : (Opcional) Defineix un identificador de Event.
      "title" : "títol que faci falta",
      "start" : "dia i  hora inicial",
      "end" : "dia i hora final"
}
```

Poden existir varis events amb el mateix id, el que significa que és un event amb repeticions.

9.1.5 Estructura de directori:

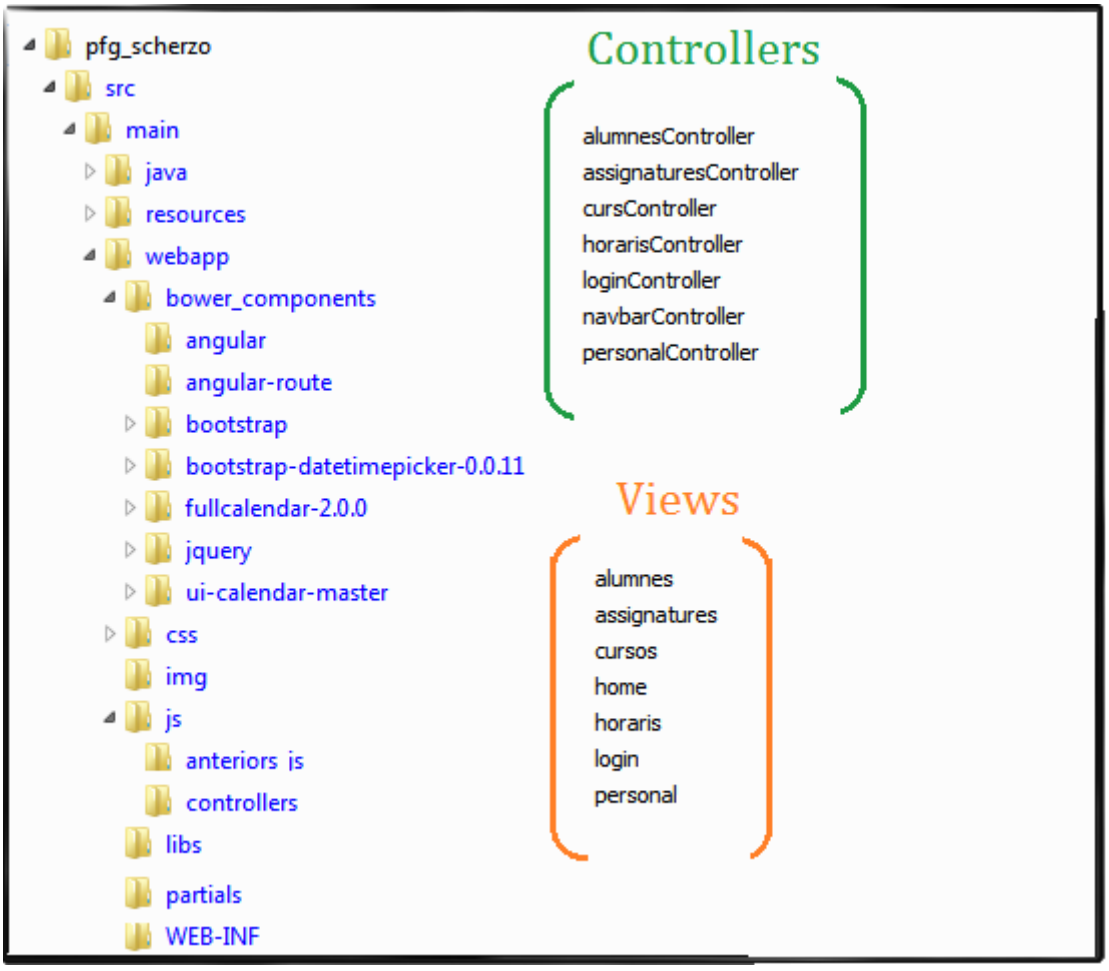


Figura 33: Estructura directoris part Client

9.2 Servidor

La part servidor és un sistema multicapa amb la següent estructura:

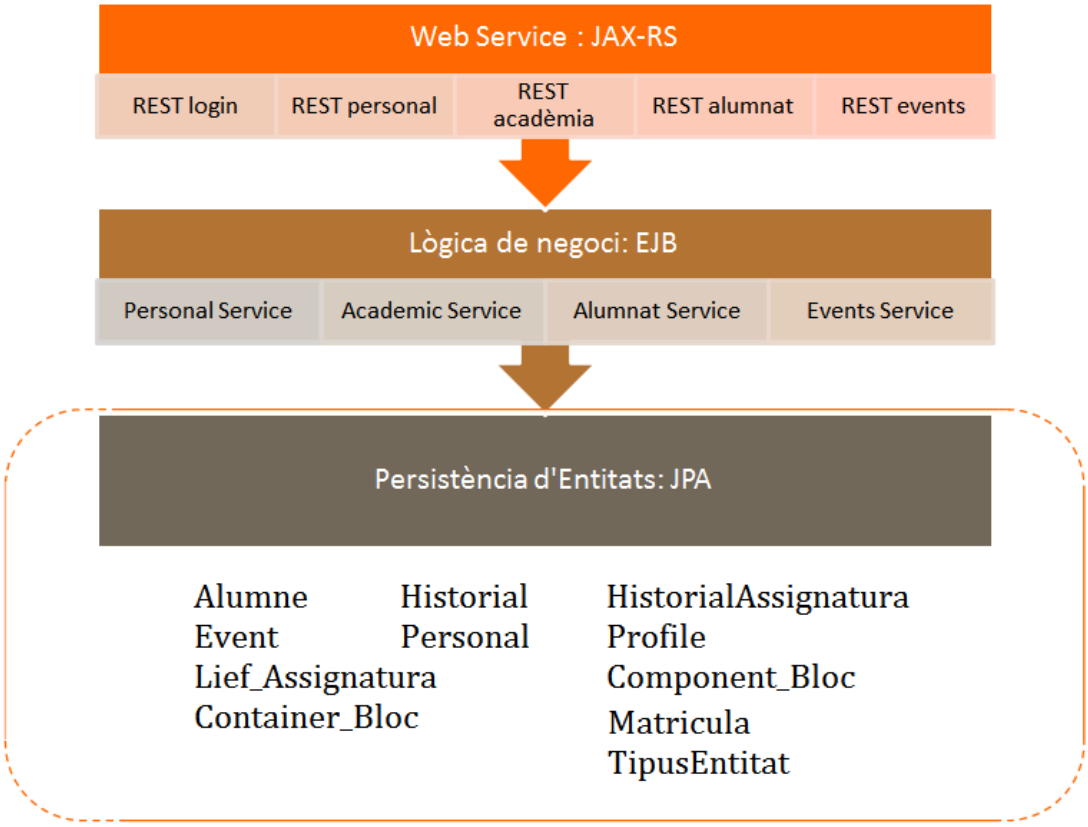


Figura 34: Arquitectura multicapa del Servidor

La primera capa és la que rep les peticions HTTP del client en format `XMLHttpRequest`. Aquesta capa està estructurada segons l'arquitectura REST que hem estat fent referència al llarg d'aquest projecte.

Hi ha 5 tipus diferents de "punts d'entrada" al servidor, que es corresponen als 4 blocs principals que comentàvem al disseny de la API, i a més un cinquè que conté la lògica de autentificació i control d'accés dels usuaris de l'aplicació.

L'únic objectiu d'aquestes classes del web service és rebre peticions i enviar respostes a aquestes peticions.

9.2.1 Capa de web service

Hi ha diferents aspectes a tenir en compte per realitzar una classe d'aquesta capa.

Cal reunir tots els possibles "punts d'entrada" que es comentava abans en una sola classe:

```
/**
 *
 * @author Fani
 */
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

/**
 * A class extending {@link Application} and annotated with @ApplicationPath is the Java EE 6
 * "no XML" approach to activating JAX-RS.
 *
 * <p>
 * Resources are served relative to the servlet path specified in the {@link ApplicationPath}
 * annotation.
 * </p>
 */
@ApplicationPath("/rest")
public class JaxRsActivator extends Application {
    /* class body intentionally left blank */
}
```

Figura 35: JaxRsActivator

JaxRsActivator.java és una classe molt simple, però necessària per configurar el JAX-RS (que ens permet aplicar REST a les classes del projecte). Simplement es defineix el *path* de l'aplicació amb l'anotació `@ApplicationPath`.

Aquest *path*, que en aquest projecte és "rest" serveix per definir la URL base de tots els recursos URI proporcionats per `@Path`.

Una vegada creada aquesta classe, es procedeix a crear les altres 5.

El següent fragment de codi correspon al fitxer *LoginRESTService.java.*

```
/**
 *
 * @author Fani
 */
@Path("/auth")
@RequestScoped
public class LoginRESTService {
    @EJB
    private PersonalService userService;
```

Figura 36: Definició de la classe LoginRESTService

Veiem varis aspectes importants en aquestes poques línies. Primer de tot, l'anotació `@Path`, que serveix per definir el següent camí per accedir a un recurs determinat.

Llavors, ara si volguéssim accedir als mètodes de la classe `LoginRESTService` caldria fer una petició a la URL : "`rest/auth`".

Després hi ha la definició del EJB `PersonalService`. El concepte EJB ha aparegut moltes vegades i potser és confús, però no és res més que el diminutiu per *Enterprise JavaBeans*, que és una de les moltes API que formen part del estàndar de construcció d'aplicacions empresarials *Java EE d'Oracle*. La seva especificació detalla com els servidors d'aplicacions proveeixen objectes des de la part servidor.

En altres paraules, els EJB ens permeten encapsular uns Objectes que s'anomenen Entitats, que són persistents i per tant es guarden a la base de dades. Les entitats no són res més que els recursos de l'aplicació. Els EJB retornen aquests objectes, els tracten, o els guarden, depèn de l'acció que vulgui realitzar l'usuari.

Vist això, el pròxim dubte a resoldre és: com s'accedeixen als mètodes REST?

Observem el pròxim fragment de codi, que no és res més que la continuació de la mateixa classe `LoginRESTService` que estàvem mirant:

```
@POST
@Path("/0")
@Produces(MediaType.APPLICATION_JSON)
public String auth(
        @Context HttpServletRequest req,
        @FormParam("username") String username,
        @FormParam("password") String password) {
```

Figura 37: Mètode d'autentificació.

En aquesta imatge podem apreciar com definim una funció anomenada auth, que rep per paràmetre 3 objectes: un `HttpServletRequest`, i dos `FormParams`, *Username* i *Password* respectivament.

Aquesta funció és un mètode que només s'hi podrà accedir si es fa una petició POST al path "`rest/auth/0`". Això es pot saber gràcies a les anotacions `@POST` i `@Path`.

També es pot saber el que retornarà aquest mètode, un `JSON`, degut a que així es defineix amb l'anotació `@Produces` i amb el tipus `MediaType.APPLICATION_JSON.`

També hi ha la opció de definir `@Consumes`, que determina quin tipus d'informació pot rebre. Pel que fa als paràmetres:

- `HttpServletRequest`: és un objecte que encapsula la petició que arriba del client, fent així possible accedir a la informació. Això es pot fer servir per guardar la sessió del usuari, donat que també es pot guardar informació dins i això es rebrà sempre a la part servidor (el client no ho podrà modificar).
- `@FormParam`: és una anotació a nivell de paràmetre i serveix per lligar paràmetres entre un formulari HTML amb variables del servei REST.

Juntament amb l'anotació `@POST` hi ha altres tipus d'anotacions que també serveixen per definir diferents mètodes HTTP, com ara `@GET`, `@PUT` i `@DELETE`.

Pel que fa als paràmetres dels mètodes, hi ha més a part del `@FormParam`:

- `PathParam`: és una anotació a nivell de paràmetre i serveix per lligar els paràmetres d'una petició clàssica REST amb les variables del servei.

  Un exemple, aquest cop de la classe `PersonalRESTService`:

```
@DELETE
@Path("/{id}")
public String deleteUser(
        @Context HttpServletRequest ui,
        @PathParam("id") Long id
) {
```

Figura 38: Mètode deleteUser( ) de PersonalRESTService

  En el *path*, veiem {id}, això defineix una variable amb nom id, que es rep per paràmetre al mètode *deleteUser*. Si es fes la següent petició: DELETE "`rest/personal/1`", s'accedirà al mètode *deleteUser*, i el paràmetre id que es de tipus Long tindrà valor 1.
- `QueryParam`: Aquesta anotació permet llegir paràmetres que vinguin a través d'una petició clàssica GET, com per exemple:

  Si es fa una petició GET a "*`rest/acadèmia/curs/assignatura?codi=AS000001W`*
  es rebrà per paràmetre un codi AS000001 que es guardarà en la seva respectiva variable.

```
@GET
@Path("/curs/assignatura/")
@Produces(MediaType.APPLICATION_JSON)
public String findCoursesByCodiAssig(
    @QueryParam("codi") String codi
){
```

Figura 39: Mètode findCoursesByCodiAssig( ) de AcademicRESTService

9.2.2 Capa de EJB

Les classes EJB són més senzilles que les anteriors, però tenen més feina. S'encarreguen de consultar les dades de les entitats, tractar-les i retornar-les a la capa superior per a que ho retorni al client.

Tal com abans, hi ha 3 anotacions importants a tenir en compte.

```
@Stateless
@LocalBean
public class PersonalService  {
    @PersistenceContext
    private EntityManager em;
```

Figura 40: @Stateless i @LocalBean

L' anotació *Stateless* marca si la lògica de negoci ha de guardar (`stateful`) o no (`stateless`) informació del client. Per definició, el REST vol evitar guardar l'estat actual, de manera que per això he definit totes les classes d'aquesta capa com a `@Stateless.`

Els EJB es poden accedir per clients Locals o clients Remots:

- Els clients locals són aquells que s'executen en la mateixa maquina virtual (JVM) que el EJB que invoquen.
- Els clients remots poden executar-se en màquines virtuals diferents a la del EJB que invoquen.

D'aquesta forma, amb anotacions `@Local` i `@Remot` podríem definir diferents interfícies, per diferenciar els mètodes que executen els clients locals dels clients remots.

L'anotació `LocalBean` serveix per evitar haver de definir l'anotació `@Local` per la interfície dels clients locals.

Per últim, es defineix un `@PersistenceContext`. Tant la API de *org.hibernate.session* com la API *javax.persistence.EntityManager* representen un context per tractar amb les dades persistents. Aquest concepte s'anomena context de persistència (persistence context). Les dades persistents tenen un estat en

relació entre el persistence context i la base de dades on es troben guardades les dades.

Es pot accedir a les dades persistents de diferents formes, de les quals les principals són:

- A partir del objecte *EntityManager*.
- A partir d'un objecte *Query*.

En el següent exemple es poden observar els dos casos:

```
public Personal addUser(String username, String password) {
    Personal nu = new Personal();
    try{
        System.out.println("UserService: ADD USER "+username);
        Query q = em.createQuery("select u from Personal u where u.username=:username");
        q.setParameter("username", username);
        try{
            Personal u = (Personal) q.getSingleResult();
        }catch(NoResultException ex){
            nu.setUsername(username);
            nu.setPassword(password);
            em.persist(nu);
            em.flush();
            return nu;
        }
    }
```

Figura 41: Mètode addUser( ) de personalRESTService

En aquest mètode es vol afegir un nou personal a l'aplicació. Primer es fa una consulta, amb llenguatge HQL[19] que selecciona de la taula Personal aquell que té al mateix *username* que el que s'especifica amb el mètode `setParameter()`.

En el cas que no trobés cap Personal amb aquest *username*, crearia un nou Personal amb aquest *username* i *password*. Per persistir aquest nou objecte Personal només cal fer `em.persist( objecte ).`

Seguidament, `em.flush( )` serveix per persistir tots els canvis que s'han realitzat en el *entity manager*. Altres operacions del *entity maneger* que he utilitzat molt són:

- ***Merge*** : Actualitzar informació.
- ***Remove***: Esborrar informació.
- ***Find***: Buscar un objecte amb un identificador establert.

[19] HQL: Hibernate Query Language.

9.2.3 Capa de Persistència

La següent imatge il·lustra a quines entitats accedeixen tots els EJB's definits en el projecte.

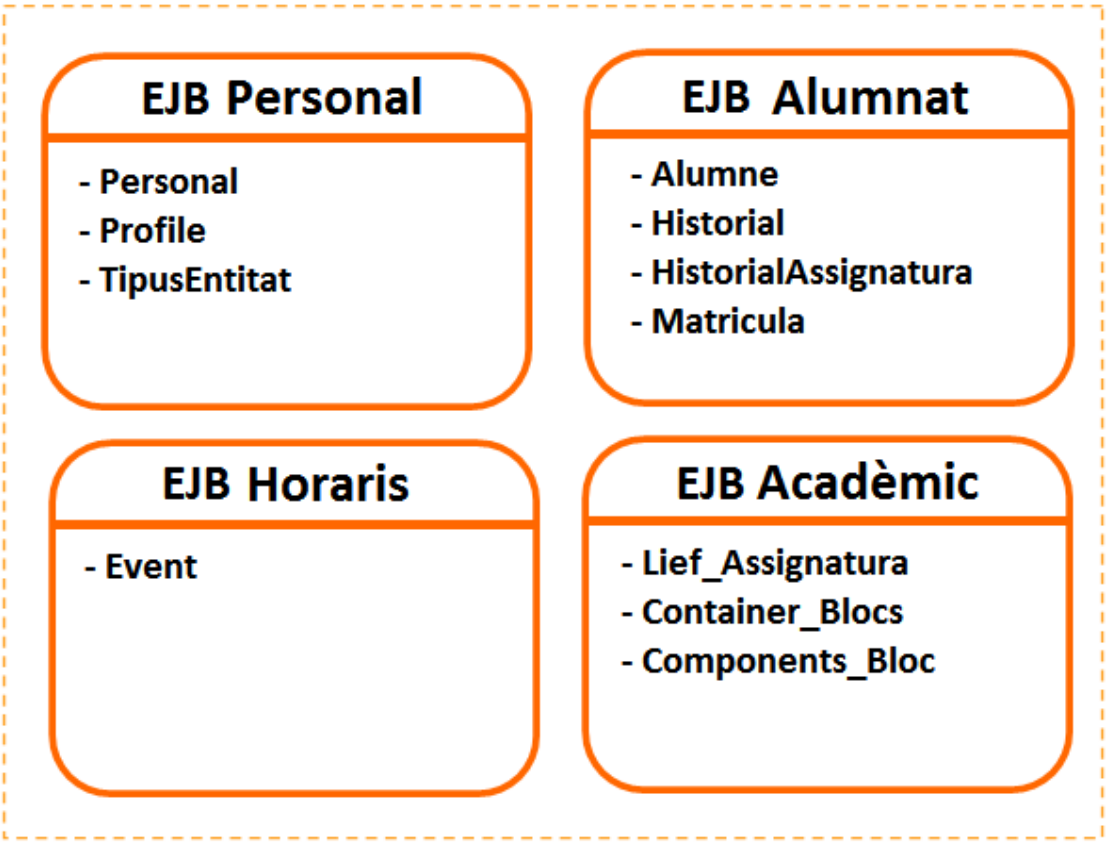


Figura 42: EJB i Entitats del projecte.

Aquestes entitats a vegades són accedides per altres EJB, com per exemple en el cas del EJB Horaris, però només per consultar, mai per modificar o crear.

Les entitats també tenen un conjunt de anotacions que cal comentar, i es farà com en els apartats anteriors, agafant una entitat del projecte i comentant el que es necessari per entendre el context.

```
@Entity
public class Alumne implements Serializable {
    private static final long serialVersionUID = 7L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Expose
    @Column(name = "ALUMNE_ID", unique = true, nullable = false)
    private Long id;
```

Figura 43: Entitat Alumne

La primera anotació que es veu és `@Entity.` Aquesta anotació s'encarrega de marcar la classe *Alumne* com a una `Entity Bean`, que representa una taula en una base de dades i cada instància d'aquesta classe representa un registre de la taula. En altres paraules, amb les `Entity Beans` s'aconsegueix crear un mapeig entre les propietats d'una classe i els camps d'una taula. A més d'aquest mapeig també es pot especificar les relacions que tenen les classes entre elles (un a un, un a molts o molts a un, i molts a molts).

El fet que sigui una implementació de `Serializable`, tot i que no es del tot necessari per la persistència, és una bona pràctica que serveix per poder representar les classes en altres formats o guardar-les en una sessió HTTP. No s'ha utilitzat cap de les dos opcions, però ha sigut una qüestió de bona pràctica el fet de posar-ho.

Un altre aspecte a comentar és la anotació `@Id` que serveix per marcar quin atribut és l'identificador de l'entitat. És totalment imprescindible definir un identificador únic a totes les entitats degut a que representarà la clau primària de la taula a la base de dades.

L'anotació `@generatedValue` determina que l'identificador id serà generat automàticament, de manera que no s'ha de portar un control (posteriorment es defineix l'estratègia que es farà servir, en aquest cas és `identity`).

També es pot especificar el nom que tindrà la columna de la taula Alumne amb l'anotació `@Column,`i assignar també diferents propietats, com ara que no pugui ser null i que hagi de ser única.

L'anotació `@Expose` no forma part del JavaEE ni de res semblant, sinó que prové de la llibreria *Gson*, que permet generar JSON o llegir-los i transformar-los a classes java. Aquesta classe permet exposar o amagar diferents atributs. Si es posa `@Expose` mostrarà l'atribut quan més endavant es vulgui fer un JSON d'una Entitat alumne.

Pel que fa a definir les relacions, la classe Alumne necessita conèixer les seves matricules, les assignatures que cursa i l'historial que té assignat.

Això es defineix de la següent forma:

```
@Expose
@OneToMany(cascade = CascadeType.ALL, fetch=FetchType.LAZY,
        mappedBy = "alumne_matricula", targetEntity = Matricula.class)
private List<Matricula> alumne_matricula;

@Expose
@ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
@JoinTable(
        name = "alumne_componentbloc",
        joinColumns =
        {
            @JoinColumn(name = "ALUMNE_ID", nullable = false, updatable = false)
        },
        inverseJoinColumns =
        {
            @JoinColumn(name = "COMPONENT_BLOC_ID",nullable = false, updatable = false)
        }
)
private List<Component_Bloc> alumne_componentbloc;

@OneToOne(cascade = CascadeType.ALL, fetch=FetchType.LAZY,
        mappedBy = "alumne_historial", targetEntity = Historial.class)
private Historial alumne_historial;
```

Figura 44: Mapeig de les relacions d'Alumne

- `@OnetoOne` : Defineix una relació de Un a Un.
- `@OneToMany` : Defineix una relació de Un a Molts.
- `@ManytoMany` : Defineix una relació de Molts a Molts.

Les més complexes i difícils d'entendre són les dues últimes. En concret, per exemple, un Many to Many crea una nova taula que s'anomena `alumne_componentbloc` que té dos columnes, `Alumne_ID` i `Component_Bloc_Id`. Cap de les dues columnes pot ser nul·la, i tampoc es pot modificar, degut a que aquesta taula es gestiona automàticament i seria molt caòtic permetre que un EJB esborri algun registre de la taula. Tot això es fa realitza mitjançant les anotacions `@JoinColumn` i `@JoinTable.`

Un altre aspecte a comentar és la característica lazy fetch[20] (`fetch.lazy`): és una de les característiques que ofereix JPA (entre altres frameworks de mapeig relacional d'objectes) que permet que, quan existeixen relacions entre diferents entitats, les que són dependents d'altres no siguin inicialitzades amb els seus valors inicialitzats a la base de dades fins que no siguin explícitament accedides (llegides).

[20] Lazy fetch: Lectura demorada.

Hi ha varies formes de forçar que s'actualitzi aquestes dades. En aquest projecte s'ha optat per incorporar el següent mètode (veure la següent figura) a tots els getters dels atributs definits com lazy.

```
public Historial getAlumne_historial() {
    Hibernate.initialize(alumne_historial);
    return alumne_historial;
}
public List<Component_Bloc> getAlumne_componentbloc() {
    Hibernate.initialize(alumne_componentbloc);
    return alumne_componentbloc;
}

public List<Matricula> getAlumne_matricula() {
    Hibernate.initialize(alumne_matricula);
    return alumne_matricula;
}
```

Figura 45: Inicialització d'atributs definits amb Lazy Fetch

D'aquesta forma només s'inicialitzen aquests atributs quan és estrictament necessari.

9.2.4 Estructura de directoris

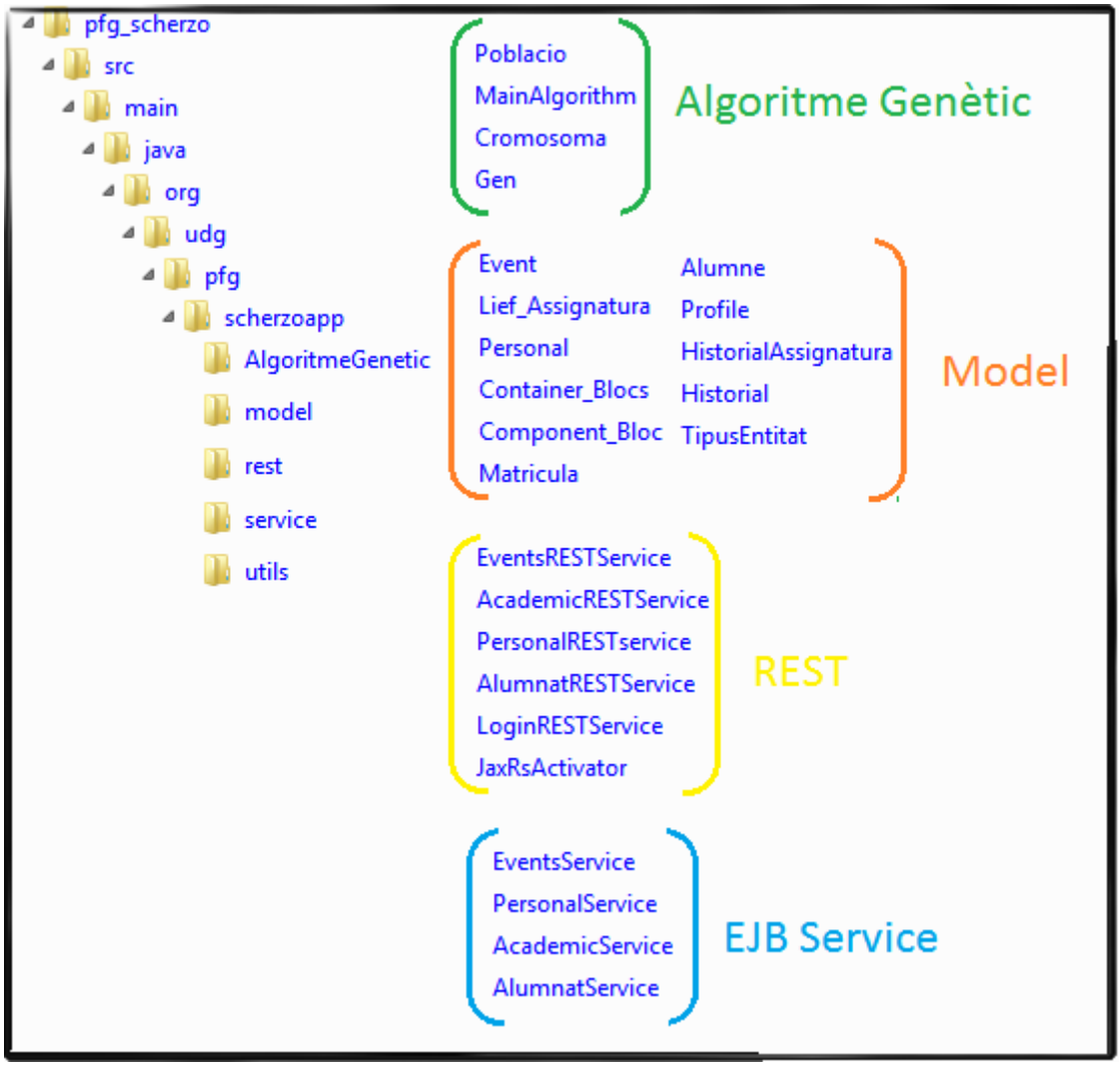


Figura 46: Estructura directoris part Servidor

9.2.5 Algoritme gènetic

Tot i que ja s'ha fet una gran explicació a l'apartat 5 sobre els algoritme genètics, a continuació s'explicarà els punts més importants de l'algoritme.

Abans però, cal introduir uns conceptes previs específics per l'algoritme que s'ha realitzat.

Concepte	Descripció
Restriccions HARD	Són les restriccions que no poden ser violades mai.
Restriccions SOFT	Són les restriccions que determinen el grau de validesa d'una solució.
Conflicte	És allò que no compleix les restriccions hard.

Les restriccions s'han anat recopilant simulant les necessitats d'una acadèmia de música.

Les restriccions hard per un problema d'aquestes característiques són:

- Una assignatura no pot ser impartida dos cops la mateixa setmana.
- No es poden impartir dues assignatures a la mateixa hora, dia i aula.
- Un professor indica els dies i les hores que pot fer classe, fora d'aquí no pot ser assignat a cap assignatura per un dia i hora que no compleixi amb les seves necessitats.
- Es pot impartir dues assignatures a la mateixa hora i dia si es fan en aules diferents i em professors diferents.
- Només es calcularan els events de 12:00 a 21:00.
- Es dona per establert que les assignatures tenen una duració d'una hora.

Aquestes condicions sempre es tenen en compte alhora de manipular els horaris calculats, de manera que si tot esta programat com toca, mai es donarà el cas que apareguin conflictes. En cas que n'apareguin, voldria dir que alguna restricció no està ben controlada.

Les restriccions soft, per entendre-ho millor, són aquelles que es tindran en compta alhora de donar una nota a un horari. Les que s'han tingut en compte són les següents:

- Que hi hagi un cert marge entre les assignatures, per evitar que tot es concentri en un sol dia, es a dir, si hi ha hores lliures millor puntuació.
- Que les assignatures s'imparteixin per la tarda, considerant que el horari de tardes comença a les 15:00 i acaba a les 21:00.
- Que un professor te les hores repartides equitativament durant els dies que pot venir a treballar (sempre que pugui venir més d'un dia).

Aquestes condicions soft proporcionen una puntuació, que servirà per seleccionar els millors horaris i després fer combinacions entre ells per arribar a una solució que es consideri òptima.

10. Implantació i resultats

En aquest apartat s'exposarà com s'ha de configurar el servidor Glassfish i després es comentaran els resultats obtinguts.

10.1 Configuració del servidor Glassfish

Un cop instal·lat el Netbeans i també el servidor Glassfish, calia configurar la base de dades sobre la qual treballaria l'aplicació del projecte. Van haver-hi molts problemes de configuració però, al final, es va aconseguir el que es buscava.

El primer pas, doncs, és iniciar el Servidor Glassfish (següent figura, pas 1), i després anar a la opció *View Domain Admin Console*.

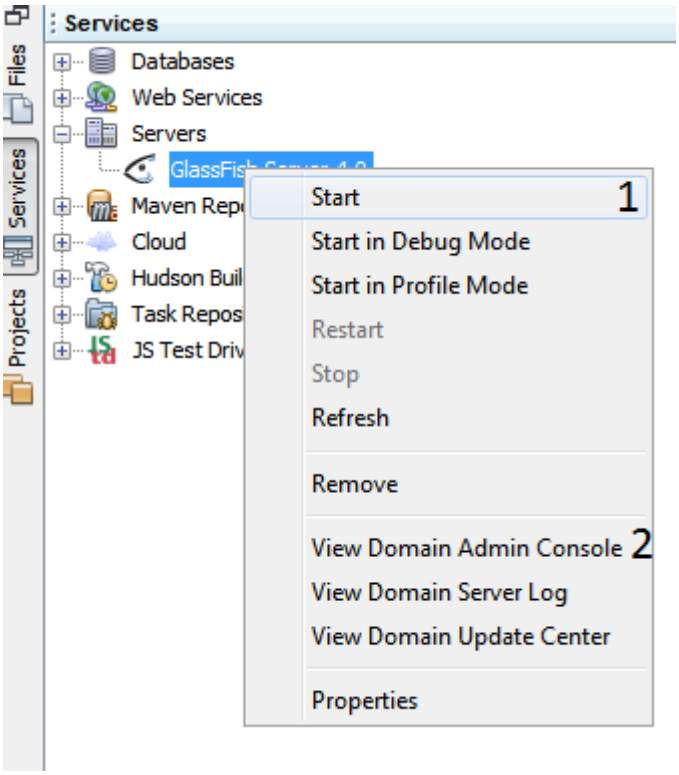


Figura 47: Connexió BD amb servidor Glassfish (1)

Aquesta opció obre una finestra en el navegador, que permet configurar el domini.

Figura 48: Connexió BD amb servidor Glassfish (2)

Quan s'arriba a aquest punt, abans d'alarmar-se, cal anar a les propietats del servidor i allà es veura quin nom d'usuari i contrasenya tenim assignats per defecte.

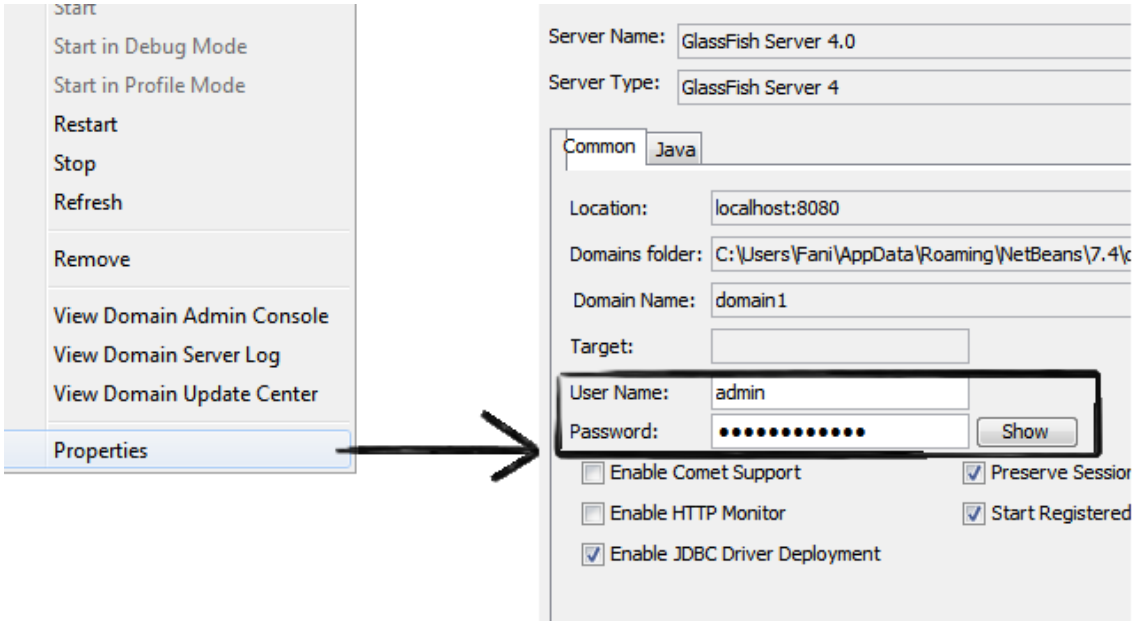


Figura 49: Connexió BD amb servidor Glassfish (3)

Un cop iniciat sessió es pot observar un menú amb vàries opcions:

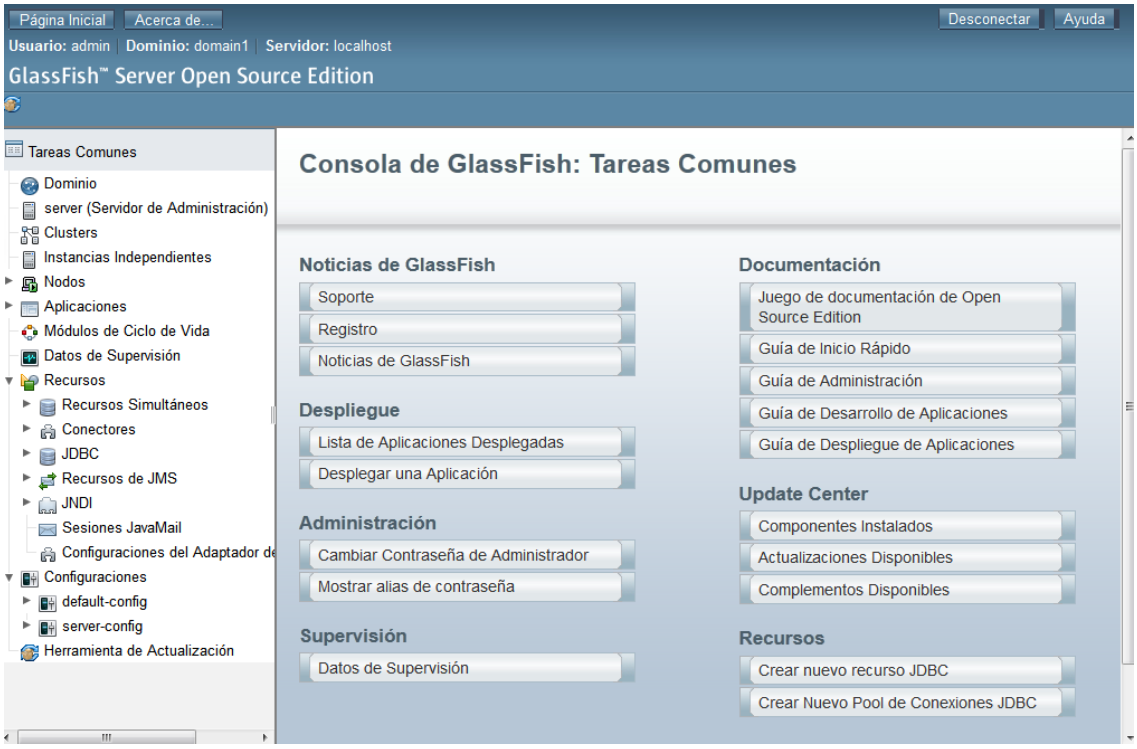


Figura 50: Connexió BD amb servidor Glassfish (4)

Degut a que s'esta configurant el servidor per a que el projecte guardi la informació a la base de dades correcta, cal anar als Pools de Connexions JDBC, que tal com informa la web, serveix per: emmagatzemar, organitzar i recuperar dades, la majoria de les aplicacions que utilitzen bases de dades (com és el cas d'aquest projecte).

També la web diu: Les aplicacions Java EE accedeixen a les bases de dades relacionals a través de la API JDBC. Abans de que una aplicació pugui accedir a una base de dades, cal obtenir una connexió.

En efecte, és el que es busca, així que es prosegueix.

Figura 51: Connexió BD amb servidor Glassfish (5)

Com es pot observar, el projecte té un pool anomenat pfg, que apunta a un recurs SQL, concretament MySql.

Per especificar aquestes caracteristiques, cal premer el boto “Nuevo...”, i arribem a la següent finestra:

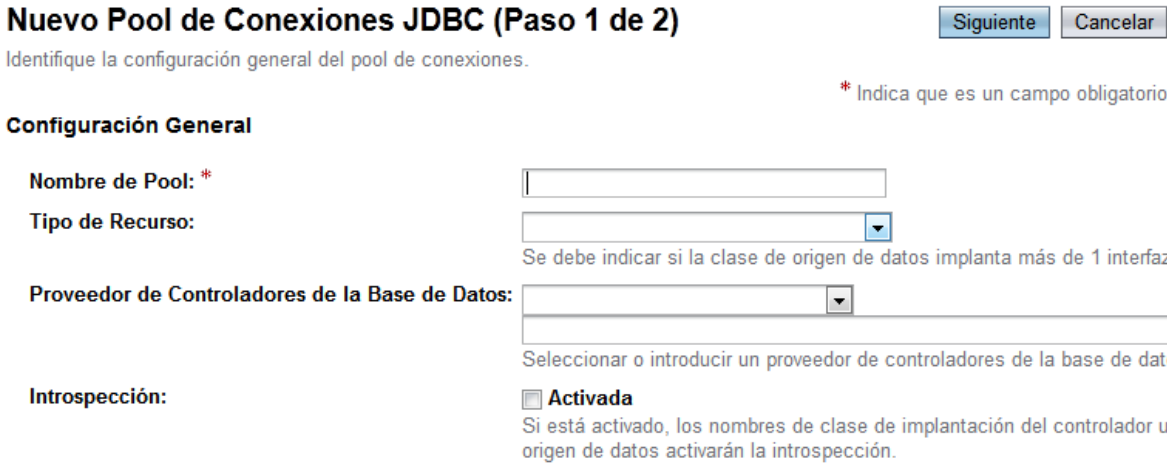


Figura 52: Connexió BD amb servidor Glassfish (6)

Cal emplenar els espais amb la següent informació (si es tracta d'una base de dades MySql):

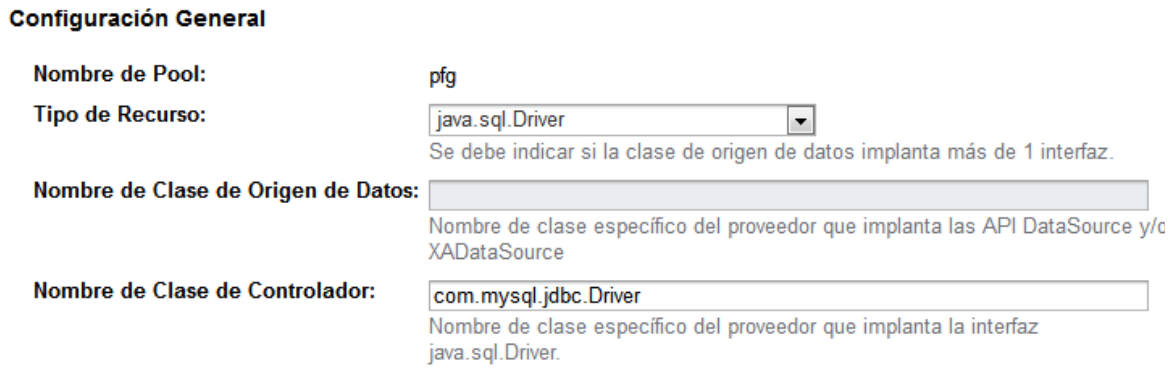


Figura 53: Connexió BD amb servidor Glassfish (7)

Fet això, només cal definir el recurs JDBC que s'utilitzarà. Això es troba en el menú que anteriorment s'ha vist.

Pel projecte he utilitzant el recurs que està senyalat:

Figura 54: Connexió BD amb servidor Glassfish (8)

Per crear-lo es va haver de especificar només el pool que es farà servir:

Figura 55: Connexió BD amb servidor Glassfish (9)

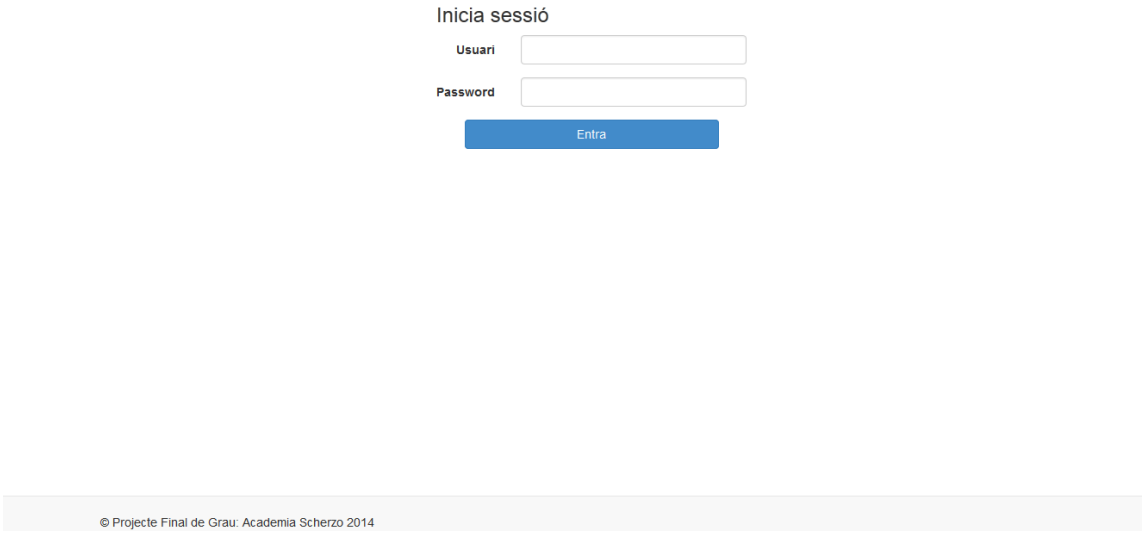
Per últim, cal crear un fitxer persistence.xml que relacionarà el proveidor de la API eclipseLink, amb la connexió JDBC/pfg que s'ha creat anteriorment.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSch
  <persistence-unit name="primary" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <jta-data-source>jdbc/pfg</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
    </properties>
  </persistence-unit>
</persistence>
```

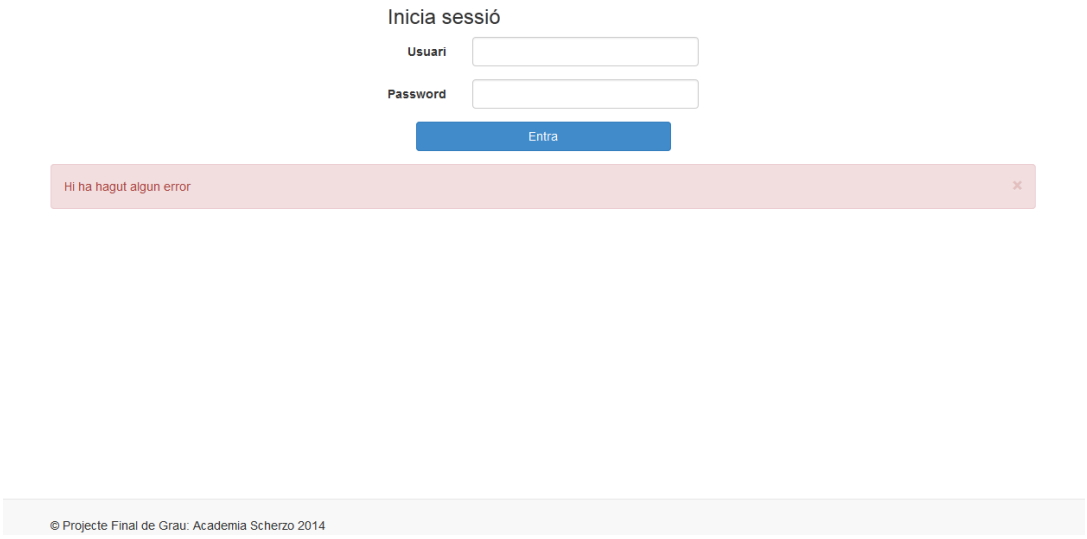
Figura 56: Fitxer persistence.xml

10.2 Resultats obtinguts

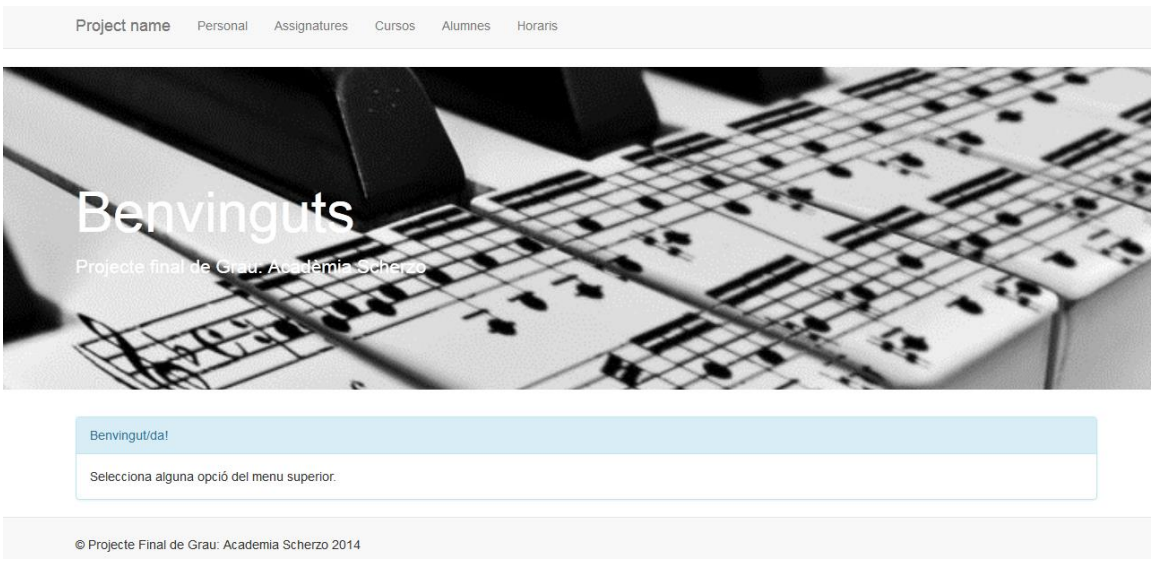
A continuació explicaré alguns dels resultats obtinguts. Donat que alguns són molt semblants, he considerat que seria important definir els següents:



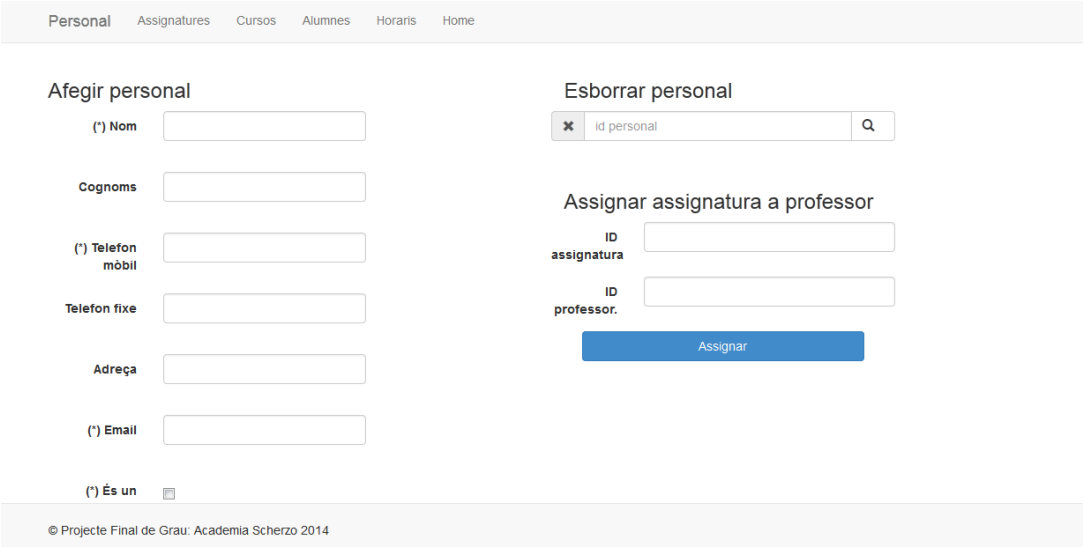
Login, que en el cas que hi hagi algun error indica:



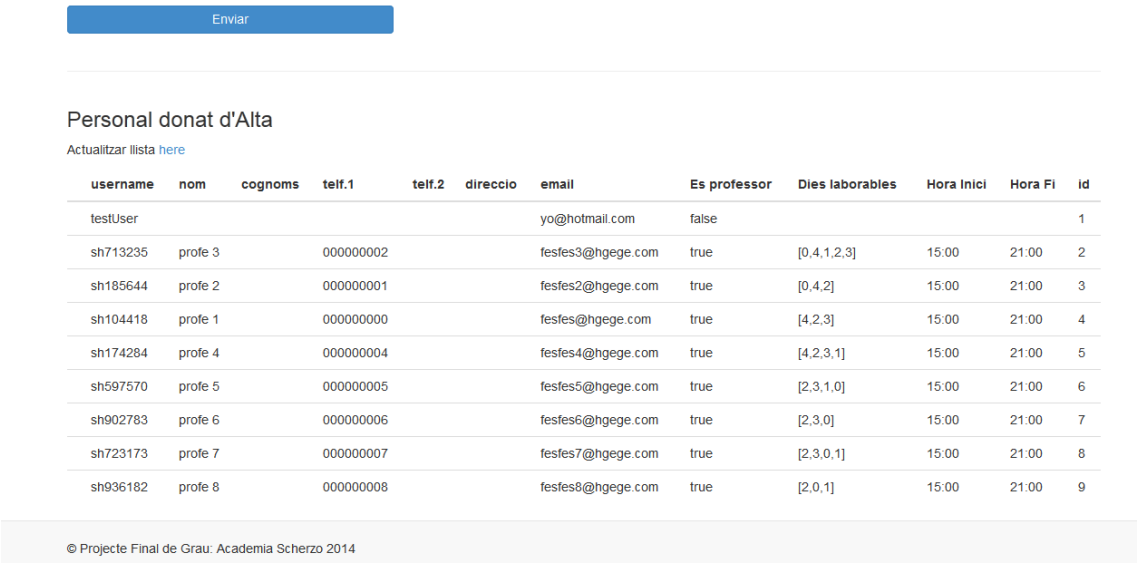
La pàgina de Home, amb una barra de navegació que permet accedir a altres vistes:



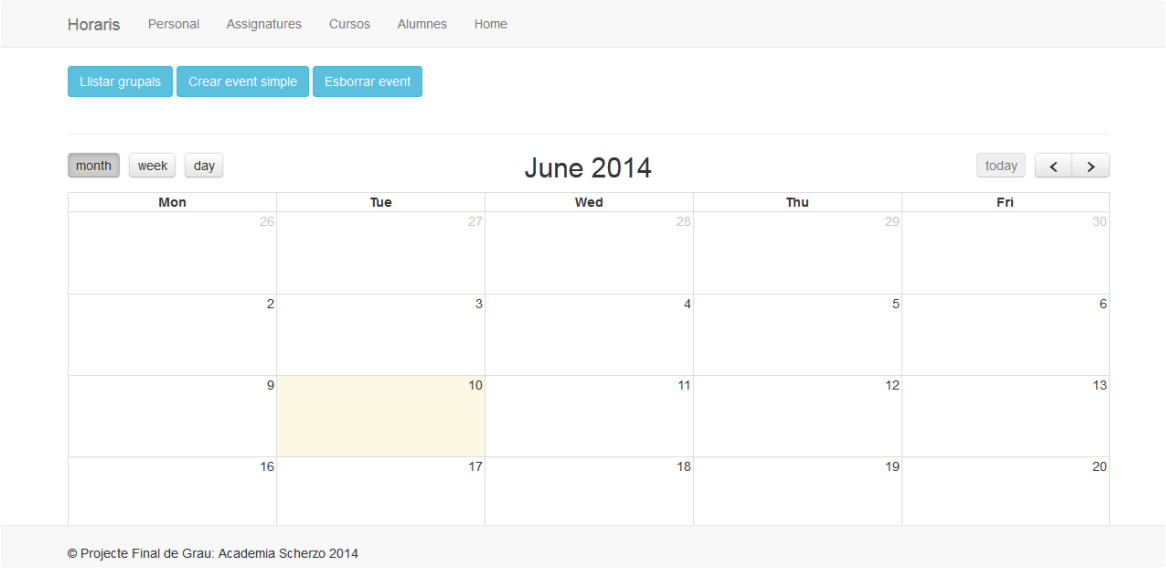
Es pot: administrar el personal, les assignatures, els cursos, els alumnes i els horaris.



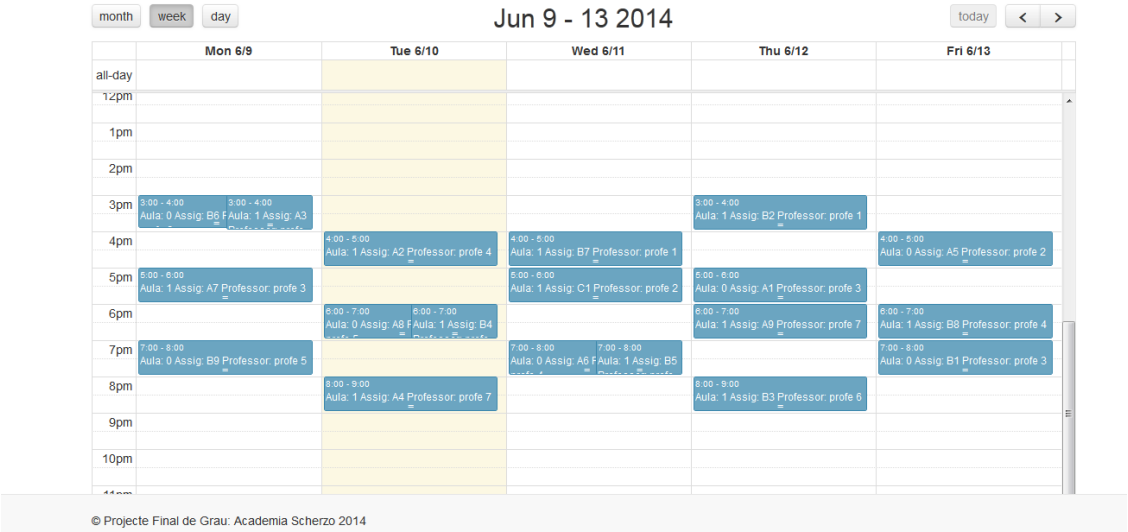
El personal que s'afegeix es mostra en una llista, a sota del afegir:



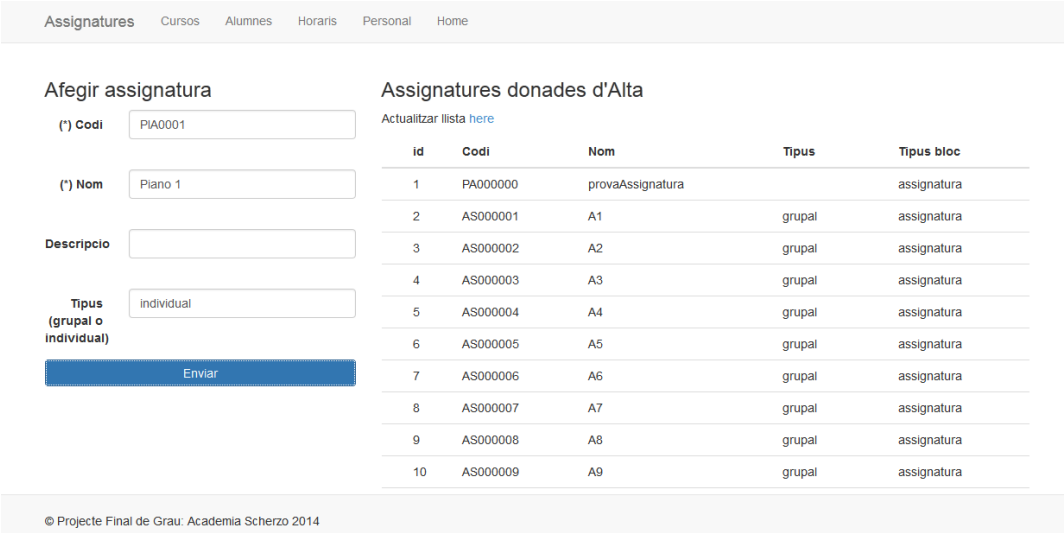
El més important i el repte més gran va ser poder incorporar l'horari a els directives AngularJS i entendre com utilitzar-ho. El resultat va ser l'esperat, i encara més:



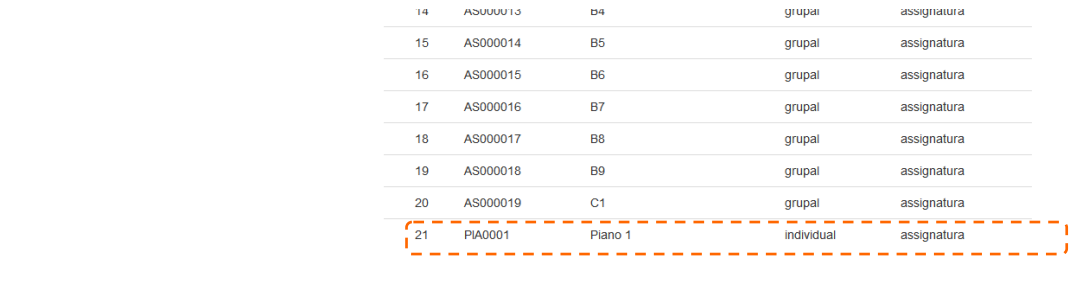
Si fem: llistar events, en cas que no existeixi un calendari per assignatures grupals, en crearà un, com succeeix en la següent figura:



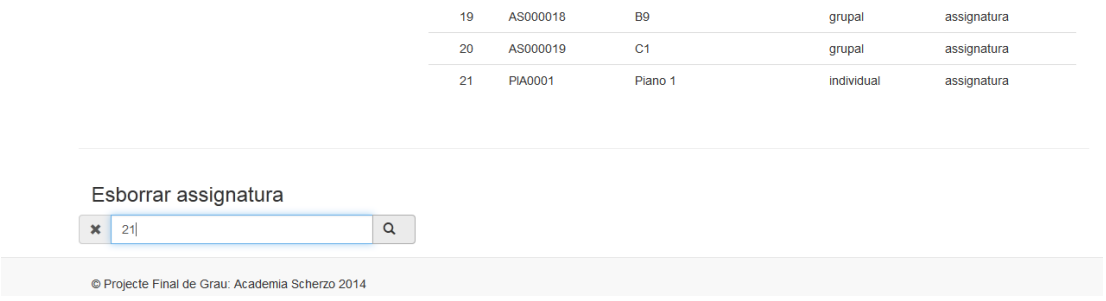
L'algoritme genètic a assignat els events repartits equitativament entre els 5 dies de la setmana, entre les 15:00 i les 21:00, tenint en compte les aules que es disposa (que s'ha establert que són dos per fer aquesta prova), els professors que imparteixen les assignatures, i les seves pròpies restriccions.



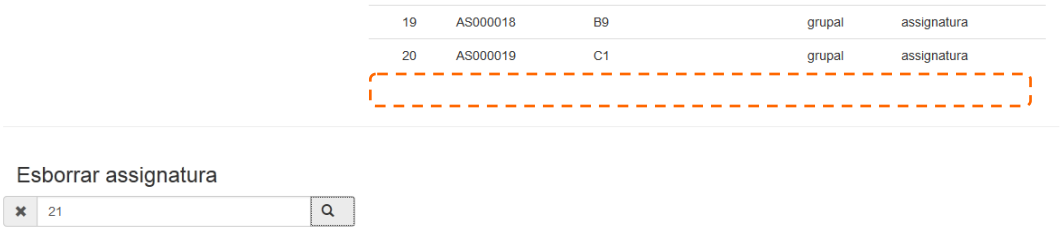
Per veure altres comportaments, aquí a la part d'administració acadèmica d'assignatures. Es vol afegir una nova assignatura, i un cop afegida es mostra a la llista.



A continuació, si ens disposem a esborrar aquesta assignatura:



Efectivament s'ha esborrat i actualitzat la llista gairebé al moment.



11. Conclusions

Els objectius del projecte eren la creació d'una aplicació web per a l'administració d'acadèmies i escoles, mitjançant una SPA amb un disseny simple, que consumeixi una API REST.

Els resultats han sigut els que s'havien establert:

- S'ha programat una SPA amb AngularJS per la gestió d'una acadèmia de música.
- El disseny de la web és simple, i útil. No hi ha informació extra, només tot allò necessari.
- S'ha programat una API REST.

A més de la API REST, s'ha creat un organitzador amb intel·ligència artificial que realitza els horaris de forma satisfactòria i els mostra correctament a través de la web.

Opinions personals

Ha sigut un repte personal, donat que per molt que hagués programat abans pàgines web, o hagués fet algun treball, sempre ha estat amb més gent, sobretot professors o professionals amb anys d'experiència al darrera. Però aquest cop les idees, la confecció, el repartiment del temps, el rendiment, tot havia d'anar al meu càrrec.

El més difícil va ser prendre les decisions corresponents als frameworks que utilitzaria per la part client. Vaig estar dubtant durant varies setmanes si seria millor utilitzar Backbone.js o Angular.js, donat que les dues proporcionaven força facilitat a l'hora de programar. La part de l'Algoritme Genètic va ser complicada també, per confeccionar una estructura de dades que em permetés realitzar tots els càlculs, i al final tot va sortir bé gràcies a la col·laboració d'una professora d'Intel·ligència Artificial, Beatriz Lopez, a qui estic profundament agraïda per aquest cop de llum que em va aportar.

Al final, he acabat recopilant molta experiència, tant amb AngularJS com amb Algoritmes Genètics, i tot i que ja coneixia Java EE, hi havia conceptes que no acabava d'entendre i ara els tinc molt presents.

12. Treball futur

Arribat a aquest punt s'ha de tenir en compte què és el que es pot modificar de la implementació proposada.

Proposta de millores:

- Al client SPA, afegir-li la funcionalitat *LocalStorage*[21], incorporada amb l'HTML5, per a que l'aplicació pugui funcionar sense connexió a Internet.
- Aprofitar les característiques de Bootstrap per adaptar completament la web a dispositius mòbils i tabletes, o al contrari, dispositius amb pantalles molt grans.
- Introduir millores a l'Algoritme Genètic per tal de millorar el temps de resposta.
- Incorporar la gestió de nomines del personal a l'aplicació.
- Creació d'aplicacions per *Android* i *iOS* amb les mateixes funcionalitats que el client web.

[21] Local Storage, explicació molt ben detallada a: http://www.html5rocks.com/es/features/storage

13. Bibliografia

(sense data). Recollit de getbootstrap.com

Angular. (sense data). Recollit de https://angularjs.org/

Backbone.js. (sense data). Recollit de http://backbonejs.org/

Emberjs. (sense data). Recollit de http://emberjs.com/

Gavidia, C. (23 / 06 / 2010). *Arquitectura y diseño de aplicaciones Java EE.* Recollit de http://www.slideshare.net/cptanalatriste/arquitectura-y-diseo-de-aplicaciones-java-ee

Gavin King, Christian Bauer, Steve Ebersole, Max Rydahl Andersen, Emmanuel Bernard, Hardy Ferentschik, Adam Warski, and Gail Badner. (sense data). *Hibernate Developer Guide*. Recollit de http://docs.jboss.org/hibernate/orm/4.0/devguide/en-US/html/index.html

Google. (sense data). *AngularJS by Google, HTML enhanced for web apps!* Recollit de https://angularjs.org/

https://docs.angularjs.org. (sense data). *AngularJS: Tutorial*. Consultat el 2014, a https://docs.angularjs.org/tutorial

Jersey, Getting Started. (sense data). Consultat el 2014, a https://jersey.java.net/documentation/latest/getting-started.html

jQuery. (sense data). Recollit de jquery.com

Learn REST: A Tutorial. (sense data). Consultat el 2014, a http://rest.elkstein.org/

Mark Otto, Jacob Thornton. (sense data). *Bootstrap 3, el manual oficial.* LibrosWeb.

ORACLE. (sense data). *ORACLE GLASSFISH SERVER*. Recollit de http://www.oracle.com/us/products/middleware/application-server/050870.pdf

Osmani, A. (2013). *Developing Backbone.js Applications.* O'Reilly Media.

Santamaría, J. M. (15 / 07 / 2011). *Manifiesto por una Única Página Web (Single Page Interface).* Consultat el 06 / 2014, a http://itsnat.sourceforge.net/php/spim/spi_manifesto_es.php

Single Page Application (SPA). (sense data). Recollit de http://en.wikipedia.org/wiki/Single-page_application

TeamGantt. (sense data). Consultat el 06 / 2014, a https://teamgantt.com/

Wikipedia. (sense data). Recollit de http://es.wikipedia.org/

14. Annexos

14.1 Llistat de figures