

Projecte Final de Carrera

Memòria

Aprenentatge d’idiomes

Alumnes: Capell Brufau, Eduard i Lorca Sans, Salvador

Consultor: Roset Mayals, Roman

Contingut

[1 Introducció 4](#_Toc339438271)

[1.1 Idea 4](#_Toc339438272)

[1.2 Possible funcionament 4](#_Toc339438273)

[1.3 Objectius 5](#_Toc339438274)

[1.3.1 Objectiu general 5](#_Toc339438275)

[1.3.2 Objectius específics 5](#_Toc339438276)

[1.4 Planificació amb fites i temporalització 5](#_Toc339438277)

[2 Estat de l’art 7](#_Toc339438278)

[3 Estudi de mercat 10](#_Toc339438279)

[3.1 Diccionaris 10](#_Toc339438280)

[3.2 *Flashcards* 10](#_Toc339438281)

[3.3 Àudio 11](#_Toc339438282)

[3.4 Pràctica de Traços 12](#_Toc339438283)

[4 Metodologia de treball 14](#_Toc339438284)

[5 Anàlisi i disseny 15](#_Toc339438285)

[5.1 Tecnologia 15](#_Toc339438286)

[5.2 Arquitectura del sistema 16](#_Toc339438287)

[5.3 Disseny preliminar del mòdul client 17](#_Toc339438288)

[5.3.1 Les pantalles del client 17](#_Toc339438289)

[5.3.2 Els objectes del model 24](#_Toc339438290)

[5.3.3 Peticions Client – Client 25](#_Toc339438291)

[5.4 Disseny preliminar del mòdul del servidor 29](#_Toc339438292)

[5.4.1 Avantatges de la implementació de les accions en servidor 30](#_Toc339438293)

[5.4.2 Inconvenients de la implementació de les accions en servidor 30](#_Toc339438294)

[5.4.3 El controlador 30](#_Toc339438295)

[5.4.4 El model 30](#_Toc339438296)

[5.4.5 Els objectes del model 31](#_Toc339438297)

[5.4.6 Emmagatzematge de dades 32](#_Toc339438298)

[5.4.7 Comunicacions client – servidor 32](#_Toc339438299)

[5.4.8 Peticions Client – Servidor 33](#_Toc339438300)

[Il·lustració 1. Diagrama Gantt de planificació 6](#_Toc339438247)

[Il·lustració 2. Selecció d’imatges kanji en el QVE 7](#_Toc339438248)

[Il·lustració 3. Pantalla d’informació d’un kanji en el QVE 7](#_Toc339438249)

[Il·lustració 4. Pantalles de pràctica d’escriptura 8](#_Toc339438250)

[Il·lustració 5. Pantalles de consulta de definició (esquerra) i cerca d’una paraula (**Japanese for iOS**: http://japaneseapp.com/). 10](#_Toc339438251)

[Il·lustració 6. Pantalles de pràctica amb flashcards. Captures corresponents a **Japanese Flip** (primera captura, https://itunes.apple.com/us/app/japanese-flip/id289263209?mt=8) i **Sticky Study Japanese** (segona i tercera captures, https://itunes.apple.com/us/app/japanese-flip/id289263209?mt=8). 11](#_Toc339438252)

[Il·lustració 7. Pantalles de pràctica amb àudio. Inclouen tant la possibilitat d’escoltar com sona una paraula, com la possibilitat de gravar la pròpia veu i re-escoltar-la posteriorment. Les captures mostrades corresponen a l’aplicació **Learn Japanese Vocabulary – Gengo** (https://itunes.apple.com/us/app/learn-japanese-vocabulary/id294770805?mt=8). 12](#_Toc339438253)

[Il·lustració 8. Pantalles Per la pràctica amb els traços del llenguatge japonès. En aquest cas es l’aplicació es limita a ensenyar l’ordre i manera d’execució dels traços. Captures extretes de l’aplicació **Kana Writing** (https://itunes.apple.com/us/app/kana-writing/id451464932?mt=8). 13](#_Toc339438254)

[Il·lustració 9. Pantalles Per la pràctica amb els traços del llenguatge japonès. Aquesta aplicació va una mica més enllà, i és més interactiva, de manera que avalua la pràctica de l’usuari, posant nota a l’execució dels traços d’un caràcter. Les captures s’han extret de l’aplicació **Kana Strokes** (https://itunes.apple.com/us/app/kana-strokes-japanese-hiragana/id318485239?mt=8). 13](#_Toc339438255)

[Il·lustració 10. Exemple de panell Kanban 14](#_Toc339438256)

[Il·lustració 11. Framework escollit, el Sencha Touch 2 15](#_Toc339438257)

[Il·lustració 12. Llenguatges usats per a l’elaboració del projecte 15](#_Toc339438258)

[Il·lustració 13. Servidor escollit, pur JavaScript 16](#_Toc339438259)

[Il·lustració 14. Sistema de control de versions del nostre projecte 16](#_Toc339438260)

[Il·lustració 15. Model Vista Controlador (MVC) del framework Sencha Touch 17](#_Toc339438261)

[Il·lustració 16. Pantalla d’inici de l’aplicació, llistes d’estudi, i pantalla detall de la llista 18](#_Toc339438262)

[Il·lustració 17. Pantalles d’edició i d’esborrat d’una llista 19](#_Toc339438263)

[Il·lustració 18. Pantalla per veure el detall del concepte d’una llista 20](#_Toc339438264)

[Il·lustració 19. Pantalles llistat i detall del diccionari 21](#_Toc339438265)

[Il·lustració 20. Pantalles d’edició i de diàleg d’un concepte 22](#_Toc339438266)

[Il·lustració 21. Pantalla prèvia a l’inici de l’exercici per a escollir una llista 23](#_Toc339438267)

[Il·lustració 22. Pantalles de l’exercici, anvers i revers de la targeta amb el concepte escrit 24](#_Toc339438268)

[Il·lustració 23. Diagrama de classes del model del client 24](#_Toc339438269)

[Il·lustració 24. Diagrama de classes del model del servidor. Diagrama fet amb l’eina ArgoUML (http://argouml.tigris.org/). 31](#_Toc339438270)

# Introducció

## Idea

Partim d’una idea de projecte que es basa en la creació d’un sistema d’aprenentatge de l’idioma japonès mitjançant dues utilitats:

* Llistes d’estudi
* Targetes didàctiques

La primera utilitat es basa en una plataforma per a definir llistes d’estudi. Una llista d’estudi és un conjunt de fitxes o elements que contenen una paraula en japonès (amb símbols *kanji*) i que estan organitzades de forma que un estudiant pot anar aprenent els símbols i la seva pronunciació navegant per les llistes d’estudi. Per exemple, podem definir una llista anomenada *colors*, i que les fitxes t’ensenyin com s’escriu i es pronuncia el *color* en qüestió.

La segona utilitat té com a objectiu la consolidació de l’aprenentatge anterior. Les targetes didàctiques (en anglès *flashcards*) són un conjunt de targetes que contenen informació, com a paraules i nombres, en un o tots dos costats usades per adquirir diversos coneixements a través de la relectura del conjunt de targetes. En un costat de la targeta apareixeria el símbol d’un dels elements en la llengua que volem aprendre (en aquest cas la japonesa) i en l'altre la pronunciació en japonès i el seu significat en català. Les *flashcards* s'usen àmpliament com un exercici d'aprenentatge per ajudar a la memorització per mitjà de la repetició espaiada.

## Possible funcionament

L’aplicació en mòbils permet la gestió de les llistes d’estudi i les paraules, més la visualització de les *flashcards*. Les característiques principals són les següents:

* Personalització de les llistes: Possibilitat de crear noves llistes d’estudi i d’afegir/modificar/esborrar elements a aquestes llistes. Per exemple: Dies de la setmana, mesos de l’any, colors, números, coses d'un hotel, aliments, etc.
* Gestió d’un diccionari de paraules: Creació, modificació i esborrat de paraules que proveiran les llistes d’estudi.
* Visualització de les *flashcards*: Desplaçament entre fitxes amb opció per escoltar com es pronuncia la paraula que conté i presentació de les fitxes usant la metodologia del sistema Leitner[[1]](#footnote-1).

## Objectius

### Objectiu general

El nostre objectiu principal és el disseny i implementació d’una aplicació web adaptada als dispositius mòbils (*tablets*, *smartphones*) que disposin de navegador.

### Objectius específics

Ens agradaria complir els següents objectius específics:

* Lliurar una eina d’aprenentatge que s’adapti a les necessitats dels estudiants que s’inicien en l’aprenentatge de la llengua japonesa.
* Assimilar tots els conceptes i nocions nous que aniran sortint durant el cicle de vida del programari, i més en concret la metodologia que ens portarà a aconseguir l’èxit.

I de manera més precisa:

* Programar la part servidor (mitjançant una API) i la part client del dispositiu mòbil.
* Col·laborar amb un altre company de titulació i realitzar un projecte comú més ambiciós.

## Planificació amb fites i temporalització

A continuació es mostra la planificació i temporització de les tasques que hem definit es en aquest diagrama *Gantt*:



1. Diagrama Gantt de planificació

# Estat de l’art

Actualment existeix una aplicació que té com a finalitat l’aprenentatge i l’avaluació personal de l’escriptura japonesa. El Quadern Virtual d’Escriptura (QVE) és una aplicació que es pot fer servir amb un ordinador o un tablet Android, i conté, per una banda, un assistent d’escriptura virtual que permet l'aprenentatge i pràctica del traç de caràcters, i per altra banda, una part teòrica sobre els caràcters. Aquesta part teòrica és la que menys profunditat presenta i de la que volem treure profit.



1. Selecció d’imatges kanji en el QVE

L’eina treballa amb 10 símbols *kanji* que representen 10 paraules en el nostre alfabet, i quan sel·lecciones un símbol et porta a la seva pantalla d’informació.



1. Pantalla d’informació d’un kanji en el QVE

En aquesta pantalla es mostren explicacions sobre el *kanji*, significats, àudios *on-yomi* i *kun-yomi,* i alguns exemples d’ús. També es pot observar una animació que indica l’ordre de traços i reproduïble mitjançant uns botons.

L’àmbit de l’aplicació és practicar l’escriptura dels símbols japonesos, i es fa amb les pantalles “Escriptura”, “Estil lliure” i “Exercicis”.







1. Pantalles de pràctica d’escriptura

Aquesta eina de simulació de l’escriptura del *kanji* permet a l’estudiant fer un traçat en l’ordre correcte amb el dit, llàpis tàctil o ratolí (si està usant un ordinador). El sistema indica a l’usuari si l’ordre que segueix és correcte o no mitjançant un flash verd o vermell respectivament.

# Estudi de mercat

Tot seguit analitzarem les aplicacions existents actualment per a dispositius mòbils que tenen com a objectiu l’aprenentatge del japonès.

Dividim les aplicacions existents en les següents categories: diccionaris, *flashcards*, àudio, pràctica dels traços. Algunes de les aplicacions encaixen en més d’una de les categories proposades (per exemple, hi ha aplicacions que fan la funció de diccionari i tenen un mòdul d’àudio per saber la pronunciació d’una paraula).

## Diccionaris

És la categoria més freqüent, probablement per la facilitat d’implementació. La característica bàsica és la possibilitat de cerca de paraules, amb el seu significat.

Característiques més interessants de les aplicacions en aquesta categoria:

* Cerca en japonès o en anglès.
* Cerca per radicals.
* Cerca per nombre de traços.
* Possibilitat d’introducció de caràcters escrits “a mà”, la qual cosa implica que hi ha implementat un sistema de reconeixement dels traços dels caràcters.
* Creació de llistes de paraules preferides.

A continuació podem veure alguns exemples d’aquest tipus d’aplicacions:

 **

1. Pantalles de consulta de definició (esquerra) i cerca d’una paraula (**Japanese for iOS**: <http://japaneseapp.com/>).

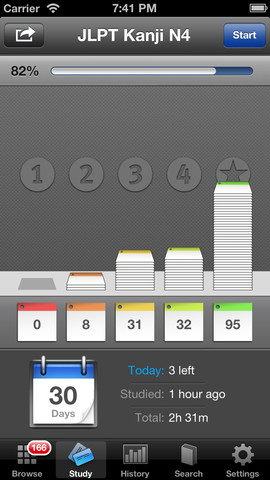
## *Flashcards*

Aquesta és una categoria molt freqüent en les aplicacions per l’estudi d’idiomes. Consisteix en una simulació de targetes didàctiques. A una banda de la targeta hi ha una pregunta, i a l’altra banda hi ha la resposta. Les plataformes mòbils són un entorn ideal per l’ús d’aquesta metodologia, per la qual cosa han triomfat molt.

Aquestes aplicacions, a més de replicar la funcionalitat de *flashcards*, també permeten d’aportar valor addicional, de la següent forma:

* Estadístiques
* Repetició de les preguntes on l’estudiant ha fallat més
* Possibilitat d’afegir targetes personalitzades
* Interacció amb d’altres usuaris o aplicacions (per exemple, correu electrònic o missatgeria instantània)

Exemples de pantalles d’aplicacions centrades en l’estudi amb flashcards:

1. Pantalles de pràctica amb flashcards. Captures corresponents a **Japanese Flip** (primera captura, <https://itunes.apple.com/us/app/japanese-flip/id289263209?mt=8>) i **Sticky Study Japanese** (segona i tercera captures, <https://itunes.apple.com/us/app/japanese-flip/id289263209?mt=8>).

## Àudio

Aquestes aplicacions tenen una funcionalitat molt simple: donat un caràcter o una paraula, permeten la reproducció de la pronunciació de la mateixa.

Algunes aplicacions van més enllà i també permeten que l’usuari gravi la seva versió de la paraula, per poder-la comparar amb la pronunciació correcta.

Podem veure’n alguns exemples:

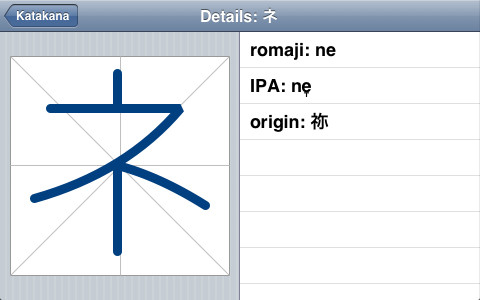
1. Pantalles de pràctica amb àudio. Inclouen tant la possibilitat d’escoltar com sona una paraula, com la possibilitat de gravar la pròpia veu i re-escoltar-la posteriorment. Les captures mostrades corresponen a l’aplicació **Learn Japanese Vocabulary – Gengo** (<https://itunes.apple.com/us/app/learn-japanese-vocabulary/id294770805?mt=8>).

## Pràctica de Traços

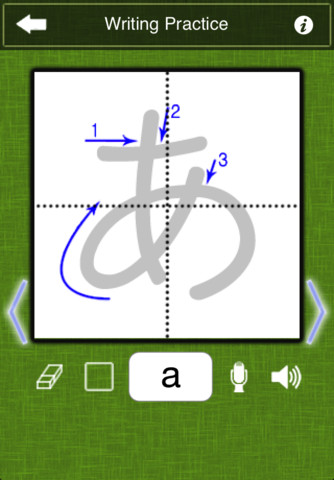
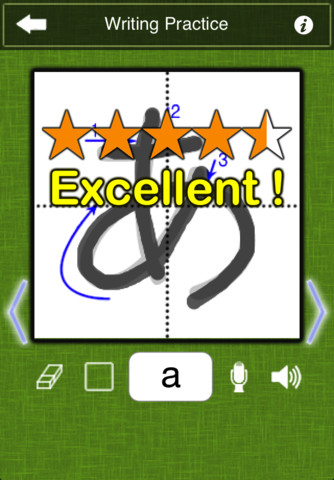
Una característica distintiva de l’idioma japonès és els traços que formen els caràcters. Donat que aquest tret constitueix un àmbit d’estudi en si mateix, és lògic que hi hagi moltes aplicacions que permeten la pràctica dels traços.

La mecànica és simple: normalment les aplicacions mostren l’ordre dels traços en una animació, i llavors és l’estudiant el que ha de reproduir els traços. Un cop acabada la introducció, algunes aplicacions reconeixen aquests traços i avaluen el rendiment de l’usuari, mentre que d’altres aplicacions es limiten a deixar en mans de l’estudiant la comprovació de la correcció dels traços introduïts.

Exemples en aquest camp:

1. Pantalles Per la pràctica amb els traços del llenguatge japonès. En aquest cas es l’aplicació es limita a ensenyar l’ordre i manera d’execució dels traços. Captures extretes de l’aplicació **Kana Writing** (<https://itunes.apple.com/us/app/kana-writing/id451464932?mt=8>).

1. Pantalles Per la pràctica amb els traços del llenguatge japonès. Aquesta aplicació va una mica més enllà, i és més interactiva, de manera que avalua la pràctica de l’usuari, posant nota a l’execució dels traços d’un caràcter. Les captures s’han extret de l’aplicació **Kana Strokes** (<https://itunes.apple.com/us/app/kana-strokes-japanese-hiragana/id318485239?mt=8>).

# Metodologia de treball

Per fer aquest treball ens hem basat en una metodologia que comparteix origen amb el llenguatge dels *kanjis*, el japonès. Es diu *Kanban[[2]](#footnote-2)*, i és un sistema de producció que dispara treball només quan existeix capacitat per processar-ho. El “disparador de treball” és representat per targetes *Kanban* de les quals es disposa d'una quantitat limitada. Cada targeta *Kanban* acompanya a un ítem durant tot el procés de producció, fins que aquest, és empès fora del sistema, alliberant una targeta. Un nou ítem de treball, solament podrà ser ingressat/acceptat si es disposa d'una targeta *Kanban* lliure.

No obstant això, en la pràctica, *Kanban* no es limita a una etiqueta (targeta). Aquesta targeta no serviria de molt si no s'apliqués d'acord a certs principis i regles. Amb tan sol tres simples regles, *Kanban* demostra ser una de les metodologies adaptatives que menys resistència al canvi presenta. Aquestes són les tres regles que hem seguit durant el desenvolupament del projecte final de carrera:

1. Visualització en tot moment dels processos que feim els dos companys.
2. Limitar el treball en curs, és a dir, acordar prèviament el nombre d’ítems que es poden abordar en cada moment.
3. Optimitzar el flux de treball.



1. Exemple de panell Kanban

# Anàlisi i disseny

## Tecnologia

Aquest projecte l’hem realitzat amb el *framework* **Sencha Touch 2**. És un gran marc de treball que permet al desenvolupador construir aplicacions que funcionin en els sistemes operatius iOS o Android, i en dispositius tan variats com BlackBerry, Kindle Fire, iPad, Nexus 7, etc.



1. Framework escollit, el Sencha Touch 2

Per a la part client hem escollit l’entorn de treball **Sencha Architect 2** i utilitzarem els nous estàndards HTML5 i CSS3, que seran utilitzats sobre un servidor web.



1. Llenguatges usats per a l’elaboració del projecte

La part del servidor l’hem fet amb **Apache Tomcat**. Aquest és un sistema creat en Java, basat en l’especificació J2EE. És una implementació oberta i lliure de J2EE, que s’utilitza per moltes aplicacions d’àmbit empresarial, per la seva robustesa i velocitat.

Durant el període de decisió de la tecnologia de servidor a utilitzar, també vam avaluar l’alternativa de **Node.js**. És molt diferent de Tomcat, però podia ser interessant, tenint en compte que està completament basat en Javascript, i el paradigma de programació és funcional. De tota manera, finalment ens hem decantat per l’opció d’Apache Tomcat, per raons principalment de velocitat de desenvolupament, ja que tenim més experiència en aquest darrer, mentre que en el cas de Node.js no tenim els coneixements necessaris per afrontar amb garanties la implementació en el temps que requereix aquest projecte.



1. Servidor escollit, basat en Java

La base de dades que utilitzarem per guardar certs objectes serà MongoDB.

Per últim, a l’hora de centralitzar el treball fet hem decidit tirar d’un repositori centralitzat públic per guardar el nostre codi *open source*. És un dels sistemes de controls de versions més usats i ofereix hospedatge i altres serveis socials. El seu nom és **GitHub**, i el nostre projecte es pot trobar al següent enllaç:

<https://github.com/salvinha/UOCPFC_Eduard_Salva.git>



1. Sistema de control de versions del nostre projecte

## Arquitectura del sistema

L’arquitectura de l’aplicació web seguirà el model vista controlador (MVC) . Aquest patró és aplicat per l’arquitectura del *framework* Sencha Touch.

Bàsicament, aquest model es comporta de la següent forma:



1. Model Vista Controlador (MVC) del framework Sencha Touch

* Una capa **Model** que s’encarrega de la representació especifica de la informació amb la que el sistema opera. Es composa per la lògica de negoci i pel sistema de gestió de base de dades.
* Una capa **Vista** que s’encarrega de la representació del model en un format adequat per interactuar, i que es coneix com interfície d’usuari.
* Una capa **Controlador** que s’encarrega d’accedir al model per tal de consultar les dades que calguin representar a la vista. És a dir, el controlador és la capa que s’encarrega de la comunicació entre la vista i el model.

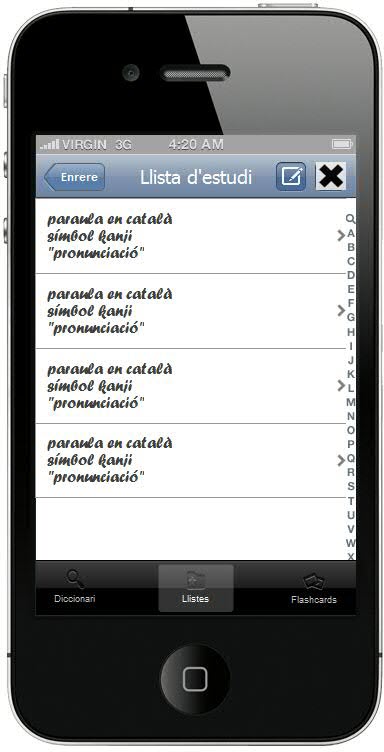
El patró MVC és molt utilitzat en entorns web on l’usuari interactua amb la pàgina, plana o vista; el controlador rep la notificació de l’acció sol·licitada per l’usuari, accedeix al model si convé i executa l’acció. Després la vista s’encarrega de rebre les dades del model per visualitzar el resultat a l’usuari.

## Disseny preliminar del mòdul client

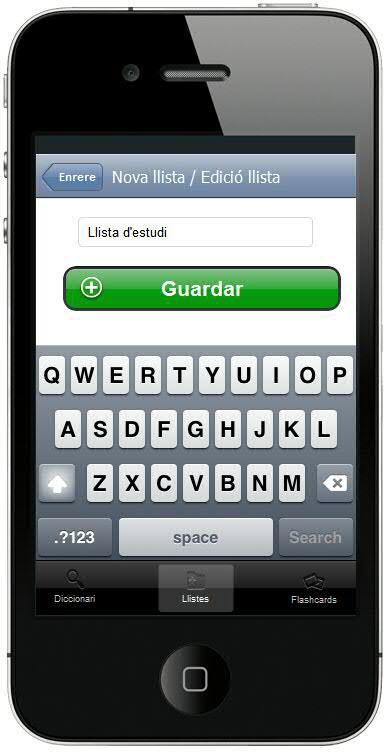
Hem realitzat una primera versió de les pantalles amb l’ajuda de l’eina *Prototyper Free* la qual ens ha ajudat a fer una maqueta digital sense funcionalitat però amb un flux de navegació.

### Les pantalles del client

#### Llistes d’estudi



1. Pantalla d’inici de l’aplicació, llistes d’estudi, i pantalla detall de la llista

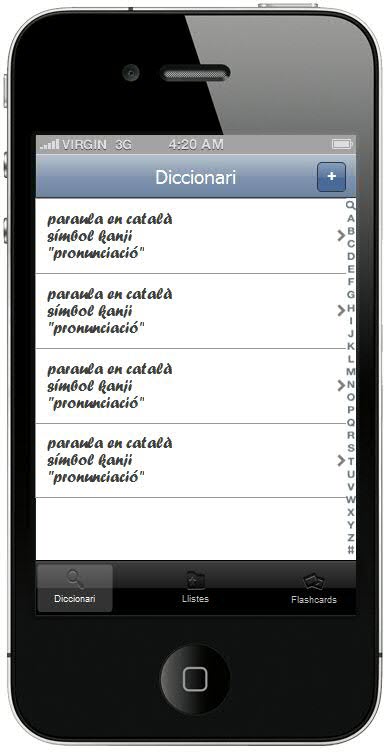


1. Pantalles d’edició i d’esborrat d’una llista

**

1. Pantalla per veure el detall del concepte d’una llista

#### Diccionari



1. Pantalles llistat i detall del diccionari

**

1. Pantalles d’edició i de diàleg d’un concepte

#### Flashcards

**

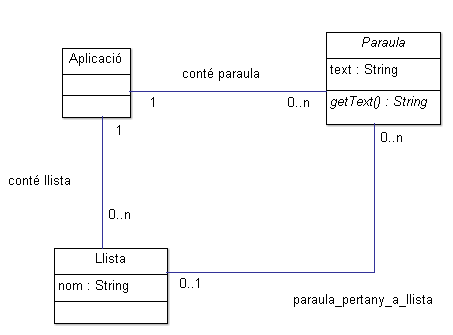
1. Pantalla prèvia a l’inici de l’exercici per a escollir una llista

**

1. Pantalles de l’exercici, anvers i revers de la targeta amb el concepte escrit

### Els objectes del model

En el següent diagrama hi podem veure els objectes bàsics del model que es faran servir per la implementació del client:



1. Diagrama de classes del model del client

Descripció breu de cada classe:

* **Aplicació**: és una entitat que engloba tota la resta d’objectes que s’utilitzaran en la aplicació.
* **Paraula**: és una classe abstracta que denota el que coneixem com a *concepte\_paraula*, és a dir: text en català i japonès, so en català i japonès.
* **Llista**: és un conjunt de paraules agrupades sota un mateix nom.

Explicació de les relacions que veiem al diagrama:

* **conté\_llista**: és una relació **1 – n**, indicant que l’aplicació pot contenir qualsevol nombre de llistes, incloent-hi cap llista.
* **conté\_paraula**: és una relació **1 – n**, indicant que l’aplicació pot contenir qualsevol nombre de paraules, incloent-hi cap paraula.
* **paraula\_pertany\_a\_llista**: cloent-hi cap paraula. ificat per parny a la llista indicada.' és una relació **n – 0,1**, indicant que una llista pot contenir qualsevol nombre de paraules (incloent-hi 0), i una paraula pot estar en una o cap llista.

### Peticions Client – Client

Les peticions Client – Client són aquelles que no surten del dispositiu de l’usuari. En aquest cas no és necessari un motor de servidor, sinó que n’hi haurà prou de programar un mòdul dins del Sencha que atengui aquestes peticions i retorni les respostes apropiades.

En tot cas, l’estructura segueix sent la mateixa que si fossin peticions Client – Servidor: hi ha una petició original, amb un nom i uns paràmetres, es fa la petició, es duu a terme la tasca corresponent, i es retornen uns resultats.

#### Peticions relacionades amb les llistes

En el cas de les llistes, les peticions que conté l’aplicació són:

* **crear\_llista**, per crear una nova llista.
* **get\_llista**, per obtenir una llista, amb les corresponents paraules.
* **get\_llistes**, per obtenir una llista amb totes les llistes creades per l’usuari.
* **editar\_camp\_llista**, per canviar alguna propietat de la llista.
* **afegir\_concepte\_llista**, per afegir paraules a una llista.
* **esborrar\_llista**, per eliminar una llista existent.
* **esborrar\_concepte\_llista**, per treure una paraula d’una llista.

|  |  |  |
| --- | --- | --- |
| **Nom** | **crear\_llista** | |
| **Descripció** | Aquesta funció permet de crear una nova llista buida, amb el nom especificat per paràmetre. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **nom\_llista** | | El nom de la nova llista |
| **Precondicions** | | |
| * No hi ha al sistema una llista amb el mateix nom. | | |
| **Postcondicions** | | |
| * S’ha creat una nova llista buida, amb el nom indicat. * Es retornarà el codi de la nova llista creada. | | |
| **Excepcions** | | |
| * Si no hi ha el paràmetre necessari, es retornarà un codi d’error (KO-1). * Si una llista amb el mateix nom ja existeix, es retornarà un codi d’error (KO-2). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **get\_llista** | |
| **Descripció** | Aquesta funció retornarà una llista de paraules, a partir d’un ID de llista especificat per paràmetre. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_llista** | | El codi de la llista que vol obtenir l’usuari. |
| **Precondicions** | | |
| * La llista amb l’ID especificat ja existeix al sistema. | | |
| **Postcondicions** | | |
| * Es retorna un objecte de tipus Llista, que conté les paraules que hi pertanyen. * No s’ha modificat cap objecte (paraules o llistes). | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si no hi ha cap llista amb l’ID especificat, es retornarà un codi d’error (KO-2). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **get\_llistes** | |
| **Descripció** | Aquesta funció retornarà el conjunt de llistes que hi ha al dispositiu de l’usuari. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| No hi ha paràmetres | |  |
| **Precondicions** | | |
| * Hi ha una o més llistes definides al sistema. | | |
| **Postcondicions** | | |
| * Es retorna un conjunt de llistes. * No s’ha modificat cap objecte (paraules o llistes). | | |
| **Excepcions** | | |
| * Si la crida conté paràmetres, es retornarà un codi d’error (KO-1). * Si no hi ha cap llista, es retornarà un codi d’error (KO-2). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **editar\_camp\_llista** | |
| **Descripció** | Aquesta funció permetrà, donat un camp i un nou valor, modificar alguna de les propietats de la llista.  Tal com s’ha dissenyat l’aplicació, només hi ha un camp rellevant (nom de la llista), però es deixa la funció oberta per tal que pugui suportar més camps en el futur. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_llista** | | El codi de la llista que es vol modificar. |
| **nom\_camp** | | Camp que es vol modificar.  Camps suportats:   * Nom de la llista (*nom\_llista*, de tipus text) |
| **nou\_valor** | | Nou valor que prendrà el camp que s’està modificant. |
| **Precondicions** | | |
| * La llista amb l’ID especificat ja existeix al sistema. * El camp que es vol modificar és vàlid (actualment, l’únic camp admès és *nom\_llista*. * El valor anterior a la modificació és diferent del nou valor. | | |
| **Postcondicions** | | |
| * S’ha modificat el valor del camp de la llista amb l’ID especificat, amb el valor nou substituint l’antic. * Les paraules que pertanyien a la llista hi segueixen associades, sense que s’hagi de fer cap acció addicional. * Si tot ha anat correctament, es retornarà el codi OK. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si el camp especificat no és vàlid, es retornarà un codi d’error (KO-2). * Si el nou valor és igual que el ja existent, es retornarà un codi d’error (KO-3). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **afegir\_concepte\_llista** | |
| **Descripció** | Aquesta funció permetrà, donat un codi de concepte\_paraula i una llista, associar la paraula i la llista en qüestió. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_llista** | | El codi de la llista que es vol modificar. |
| **id\_concepte\_paraula** | | Camp que es vol modificar.  Camps suportats:   * Nom de la llista (*nom\_llista*, de tipus text) |
| **Precondicions** | | |
| * La llista amb l’ID especificat ja existeix al sistema. * La paraula amb el codi de concepte indicat existeix al sistema. * La llista i la paraula indicades no estan prèviament associades. | | |
| **Postcondicions** | | |
| * S’ha creat una associació entre la llista i la paraula, de manera que ara aquesta paraula pertany a la llista indicada. * Si tot ha anat correctament, es retornarà el codi OK. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si la llista no existeix, es retornarà un codi d’error (KO-2). * Si la paraula no existeix, es retornarà un codi d’error (KO-3). * Si la paraula i la llista ja estan prèviament associades, es retornarà un codi d’error (KO-4). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **esborrar\_llista** | |
| **Descripció** | Aquesta funció permetrà, a partir d’un codi de llista indicat, esborrar aquesta llista del dispositiu de l’usuari. Les paraules que hi pertanyin deixaran d’estar-hi associades, però seguiran existint, sense estar associades a la llista esborrada (podran seguir pertanyent a d’altres llistes o bé no pertànyer a cap). | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_llista** | | El codi de la llista que es vol esborrar. |
| **Precondicions** | | |
| * La llista amb l’ID especificat ja existeix al sistema. | | |
| **Postcondicions** | | |
| * S’ha esborrat la llista del sistema de l’usuari. * Les paraules que pertanyien a aquesta llista segueixen existint. * Si tot ha anat bé, es retornarà un codi OK. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si no hi ha cap llista amb el paràmetre indicat, es retornarà un codi d’error (KO-2). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **esborrar\_concepte\_llista** | |
| **Descripció** | Aquesta funció permetrà, donada una llista i un concepte\_paraula, desfer l’associació entre l’una i l’altre. D’aquesta manera, la paraula en qüestió deixarà d’estar associada a la llista.  Un cop finalitzada l’operació, tant la llista com la paraula continuen existint al sistema, simplement s’ha desfet l’associació entre elles. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_llista** | | El codi de la llista que es vol modificar. |
| **id\_concepte\_paraula** | | El codi de la paraula que es vol esborrar de la llista. |
| **Precondicions** | | |
| * La llista amb l’ID especificat ja existeix al sistema. * El concepte\_paraula amb l’ID especificat ja existeix al sistema. * Hi ha una associació entre la paraula i la llista indicades. | | |
| **Postcondicions** | | |
| * S’ha desfet l’associació entre la llista i la paraula indicades. * La paraula i la llista continuen existint, però ja no estan relacionades. * Si tot ha anat correctament, es retornarà el codi OK. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si la llista especificada no existeix, es retornarà un codi d’error (KO-2). * Si la paraula especificada no existeix, es retornarà un codi d’error (KO-3). * Si la paraula i la llista no estaven associades, es retornarà un codi d’error (KO-4). | | |

#### Peticions relacionades amb Flashcards

En el cas de les flashcards, les peticions que hi intervenen són les següents:

* **init\_boxes**, per inicialitzar les caixes.
* **get\_card**, per obtenir la propera targeta d’una caixa.
* **go\_on**, per mostrar la següent targeta quan l’usuari n’encerta una.
* **go\_back**, per mostrar la targeta anterior quan l’usuari n’erra una.

|  |  |  |
| --- | --- | --- |
| **Nom** | **init\_boxes** | |
| **Descripció** | Aquesta petició s’encarrega de posar tots els conceptes d’una llista passada per paràmetre a la caixa inicial, i també d’inicialitzar les altres dues caixes (deixar-les sense conceptes o targetes). Després de fer aquesta inicialització es farà una petició al mètode get\_card per obtenir la propera targeta de la caixa inicial (serà la primera de totes). | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_llista** | | El codi de la llista que volem usar. |
| **Precondicions** | | |
| * La llista amb l’ID especificat ja existeix al sistema. | | |
| **Postcondicions** | | |
| * Els conceptes de la llista es col·loquen en la caixa inicial, la de l’esquerra, i s’extreu la primera targeta de la caixa. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si no hi ha cap llista amb el paràmetre indicat, es retornarà un codi d’error (KO-2). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **get\_card** | |
| **Descripció** | Aquesta petició s’encarrega d’obtenir la targeta disponible de la caixa amb paràmetre id\_box. En el cas que no hagin targetes disponibles a la caixa s’haurà de fer una altra petició a la caixa posterior/anterior, depenent de si ens trobem a la caixa inicial o a una caixa intermèdia. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_box** | | El codi de la caixa de la qual es vol obtenir la següent targeta. |
| **Precondicions** | | |
| * La caixa amb ID igual al paràmetre ha de ser una caixa existent. | | |
| **Postcondicions** | | |
| * Visualitzem la targeta disponible de la caixa. * Si les caixes inicials i intermèdies no contenen targetes s’haurà acabat el joc. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si no hi ha cap caixa amb el paràmetre indicat, es retornarà un codi d’error (KO-2). * Si no es troben targetes a qualsevol caixa que no sigui la final s’acaba el joc (OK) | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **go\_on** | |
| **Descripció** | L’usuari quan clica el botó “He encertat” el que fa es llançar la següent petició per a què el sistema mogui la targeta actual a la següent caixa i després faci un get\_card de la caixa actual. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_concepte\_paraula** | | La paraula (flashcard) que l’usuari ha encertat. |
| **Precondicions** | | |
| * El concepte\_paraula indicat al paràmetre existeix al sistema * El concepte\_paraula indicat al paràmetre no està a la caixa final. | | |
| **Postcondicions** | | |
| * La targeta ocupa un lloc a la següent caixa. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **go\_back** | |
| **Descripció** | L’usuari quan clica el botó “He fallat” el que fa es llançar la següent petició per a què el sistema mogui la targeta actual a la caixa inicial i després faci un get\_card de la caixa actual. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_concepte\_paraula** | | La paraula (flashcard) que l’usuari ha fallat. |
| **Precondicions** | | |
| * El concepte\_paraula indicat al paràmetre existeix al sistema * El concepte\_paraula indicat al paràmetre no està a la caixa final. | | |
| **Postcondicions** | | |
| * La targeta ocupa un lloc a la caixa inicial. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). | | |

## Disseny preliminar del mòdul del servidor

Seguint amb el model MVC exposat anteriorment, ara ens centrarem en el disseny de la part corresponent al servidor.

La idea és fer disponibles un conjunt de serveis, que seran els que invocaran certes accions del client. D’aquesta manera, sempre que es vulgui realitzar alguna operació des de les pantalles, aquesta operació serà gestionada pel servidor. Aquesta arquitectura té avantatges i inconvenients, que mostrem a continuació.

### Avantatges de la implementació de les accions en servidor

* Centralitat: totes les respostes a qualsevol acció estan sota el nostre control. Si hi ha un error, o bé si hi ha alguna millora a fer, n’hi ha prou d’actualitzar el servidor per tal de fer disponible la millora a tots els clients, sense que faci falta que els clients s’actualitzin.
* Lleugeresa: menys pes de l’aplicació client. Si volguéssim fer disponible tota la funcionalitat en els clients, això implicaria que aquests s’haurien de baixar totes les dades necessàries per fer funcionar el sistema. Això té rellevància tant en el consum de dades de l’usuari, com en l’ocupació d’espai en la memòria del dispositiu de l’usuari.
* Potència: les accions a realitzar les fa el servidor, i no pas el client. Això implica que la potència de càlcul resideix en el servidor, i el client es despreocupa de tota aquesta lògica. És un factor molt important, perquè si posem més funcionalitat al dispositiu mòbil això implica que pot suposar un consum de bateria més elevat i l’experiència d’usuari se’n pot ressentir.
* Manteniment: el disseny client – servidor ens obliga a mantenir un servei en el llarg termini, perquè els usuaris confiaran que l’aplicació els ha de funcionar durant un llarg període de temps. Seran necessaris mecanismes de monitorització i de backup per tal d’assegurar la màxima disponibilitat possible.

### Inconvenients de la implementació de les accions en servidor

* Connectivitat: el fet que les accions s’hagin de realitzar sempre contra el servidor obliga a estar connectats permanentment. Això vol dir que certes accions només es podran fer quan el dispositiu estigui connectat a la xarxa.
* Dependència: si les accions són al servidor, el programador segueix controlant l’aplicació encara que l’usuari se l’hagi baixat. Això implica que si el servidor cau o hi ha problemes en el mòdul del servidor, el client és inoperatiu, malgrat que tingui connectivitat i tingui la versió més recent de l’aplicació.

### El controlador

El Controlador serà un únic punt d’accés a qualsevol funcionalitat del servidor. Totes les peticions arribaran al Controlador, que les gestionarà de la manera convenient. Les responsabilitats del Controlador són les següents:

* Rebre totes les peticions.
* Donada una petició, esbrinar a qui correspon la tasca de dur-la a terme, i delegar-la-hi.
* Recollir els resultats de la petició.
* Retornar els resultats de la petició al client.

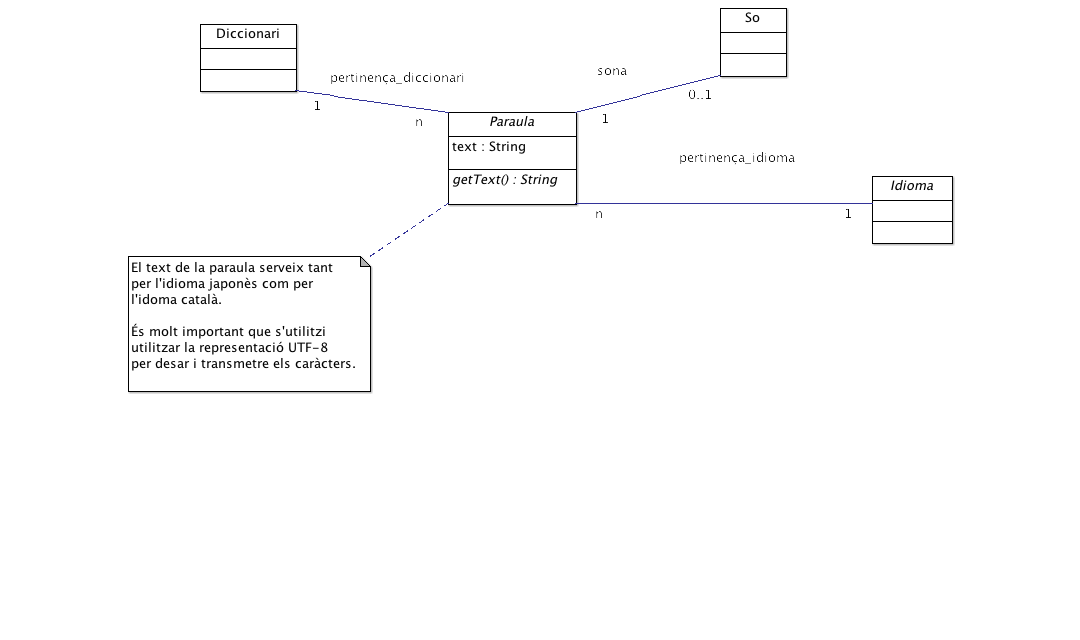
### El model

El Model, en el paradigma MVC, és el conjunt de components que representen la realitat que estem modelant. En el cas concret de la nostra aplicació això suposarà:

* Codi de servidor amb les classes que representen els objectes que prenen part en el sistema.
* Base de dades relacional, amb la informació desada que l’aplicació necessita per a funcionar.

### Els objectes del model

Per representar la realitat que ens ocupa, necessitem un conjunt d’objectes, que veiem a continuació:



1. Diagrama de classes del model del servidor. Diagrama fet amb l’eina ArgoUML (<http://argouml.tigris.org/>).

Descripció breu de cada classe:

* **Paraula**: una paraula, en l’idioma que sigui.
  + S’ha representat com una classe abstracta, de la qual heretaran les paraules concretes en els idiomes que sigui.
  + Conté el mètode abstracte *getText():String*. Aquest mètode retornarà el text de la paraula (variable *text*). Com s’indica al requadre del comentari del diagrama, el text es pot guardar en aquesta variable, tenint sempre en compte d’utilitzar la representació UTF-8.
* **Idioma**: idioma en què està una paraula.
  + S’ha representat com una classe abstracta, de manera que posteriorment s’implementin els idiomes concrets que es necessitin, en el nostre cas Català i Japonès.
* **So**: una representació àudio d’una paraula, en l’idioma que sigui.
  + S’utilitza aquesta classe com una representació a alt nivell. Posteriorment es decidirà per quina implementació s’opta per representar l’àudio, tenint en compte que s’ha d’emmagatzemar i transmetre per la xarxa de manera òptima.
* **Diccionari**: el conjunt de paraules del sistema.

Explicació de les relacions que veiem al diagrama:

* Relació **Idioma – Paraula**: és una relació **1 – n**. Una paraula està sempre en un idioma; un idioma pot tenir qualsevol nombre, positiu o 0, de paraules.
* Relació **Diccionari – Paraula**: és una relació **1 – n**. Una paraula és a un diccionari; un diccionari pot contenir qualsevol nombre, positiu o 0, de paraules.
* Relació **So – Paraula**: és una relació **1 – 0..1**. Una paraula pot, o no, tenir un àudio associat; un àudio correspon sempre a una paraula.

### Emmagatzematge de dades

Pel que fa a l’emmagatzematge de les dades, hem de tenir en compte que hem de guardar dos tipus d’informacions:

* D’una banda, la informació textual de la paraula (representació de text, pertinença al diccionari, idioma).
* De l’altra, la representació en format àudio de la paraula.

La nostra proposta consisteix a utilitzar dos mecanismes diferents per tal d’emmagatzemar aquesta informació:

* Es proposa de desar la informació textual a una base de dades No SQL. Aquesta tria s’ha fet un cop avaluat l’alternativa de fer-ho en una base de dades relacional (tipus MySQL o SQLite). Hem optat per MongoDB (<http://www.mongodb.org>), perquè és un sistema de programari lliure, que compleix perfectament les funcions que necessitem, és escalable i també ens permetrà de començar a explorar el terreny de les bases de dades No SQL, que abans d’aquest projecte ens era desconegut.
* Pel que fa a la informació d’àudio, es desarà en el sistema de fitxers. Això proporciona més flexibilitat pel que fa a l’emmagatzematge. Desar fitxers d’àudio a la base de dades pot ser problemàtic, sobre tot en el cas en què vulguem fer migracions a d’altres sistemes, o bé còpies de seguretat. A la base de dades hi guardarem, associada a cada paraula, el nom del fitxer d’àudio i la ruta del sistema de fitxers on s’ha desat l’àudio de la paraula.

Informació que es desarà a la base de dades:

* **Paraules**:
  + Desarem la paraula en l’idioma japonès i també la seva traducció en català.
  + La paraula en japonès es desarà en l’alfabet japonès, amb els seus *kanji* corresponents.
  + Els usuaris podran afegir paraules pròpies, de manera que la base de dades tindrà unes paraules predeterminades, i unes altres que cada usuari afegirà.
  + Desarem un enllaç a la representació àudio per saber on es troba ubicada en el sistema de fitxers del servidor.

No s’inclou cap diagrama Entitat / Relació, perquè l’entitat **Paraula** serà l’única que es desarà a la base de dades.

### Comunicacions client – servidor

Cada comunicació entre client i servidor es farà mitjançant peticions JSON[[3]](#footnote-3). Totes les peticions tenen la següent estructura:

* Nom de la petició
* Paràmetres (alguns obligatoris, d’altres d’opcionals)
* Resposta del servidor, que prèviament haurà realitzat les accions oportunes, si és necessari.

D’ara en endavant, parlarem d’un nou terme, **Concepte\_Paraula**. Aquest terme es refereix a tota la informació que envolta una paraula:

* Text en català de la paraula.
* So en català de la paraula.
* Text en japonès de la paraula.
* So en japonès de la paraula.

Les peticions de què disposarà la nostra aplicació al servidor seran les següents:

* **crear\_concepte\_paraula**, per donar d’alta una nova paraula al sistema
* **editar\_camp\_concepte\_paraula**, per canviar de valor algun dels camps d’un concepte\_paraula.
* **get\_concepte\_paraula**, per obtenir un concepte\_paraula concret.
* **search\_concepte\_paraula**, per buscar en la llista d’elements concepte\_paraula.
* **get\_sound**, per obtenir el so associat a una paraula

### Peticions Client – Servidor

|  |  |  |
| --- | --- | --- |
| **Nom** | **crear\_concepte\_paraula** | |
| **Descripció** | Acció que s’executa quan un usuari vol donar d’alta una nova paraula al sistema. L’usuari introduirà el text en català i en japonès, i també gravarà (si el dispositiu ho permet) l’àudio associat a cada un dels idiomes.  Tota aquesta informació s’enviarà al servidor, que la desarà si és possible (per exemple, si la paraula ja existeix en català o bé en japonès, no es podrà desar, i es retornarà un codi d’error). | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **text\_catala** | | Text de la paraula en català |
| **text\_japones** | | Text de la paraula en japonès (els *kanji* corresponents) |
| so\_catala | | Àudio de la paraula en català. Opcional. |
| so\_japones | | Àudio de la paraula en japonès. Opcional. |
| **Precondicions** | | |
| * L’usuari ha entrat tots els camps obligatoris. * La paraula no existeix prèviament al diccionari del servidor. | | |
| **Postcondicions** | | |
| * La paraula ha estat donada d’alta al sistema, en els dos idiomes. Si l’usuari també ha enviat la representació en àudio en qualsevol dels dos idiomes (o en els dos) es desarà en forma de fitxer al sistema de fitxers, i la ruta es guardarà juntament amb la paraula a la base de dades. * El servidor retorna un codi de concepte\_paraula (ID) amb el qual s’identificarà a partir d’ara l’entitat creada. | | |
| **Excepcions** | | |
| * Si no tots els camps han estat emplenats, es retornarà un codi d’error al client (KO-1). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **editar\_camp\_concepte\_paraula** | |
| **Descripció** | Acció que s’executa quan un usuari vol canviar el valor d’algun dels camps d’un concepte paraula.  Els camps que es poden canviar amb aquesta acció són: text de la paraula (en català o en japonès); so de la paraula (en català o en japonès). | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_concepte\_paraula** | | El codi del concepte\_paraula que es vol modificar. |
| text\_catala | | Text de la paraula en català |
| text\_japones | | Text de la paraula en japonès (els *kanji* corresponents) |
| so\_catala | | Àudio de la paraula en català. Opcional. |
| so\_japones | | Àudio de la paraula en japonès. Opcional. |
| **Precondicions** | | |
| * La paraula amb l’ID especificat ja existeix al sistema. * Almenys un dels següents paràmetres és diferent de null:   + text\_catala   + text\_japones   + so\_catala   + so\_japones | | |
| **Postcondicions** | | |
| * La paraula ha estat modificada en qualsevol dels seus camps. Si no hi havia cap valor diferent dels ja existents, no es modificarà res. | | |
| **Excepcions** | | |
| * Si no hi ha almenys un dels paràmetres necessaris (textos o sons), es retornarà un codi d’error (KO-1). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **get\_concepte\_paraula** | |
| **Descripció** | Acció que s’executa quan un usuari vol recuperar un concepte\_paraula a partir de l’ID.  Serveix per obtenir tota la informació associada a un concepte\_paraula, és a dir: textos (català i japonès) i sons (català i japonès). | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_concepte\_paraula** | | El codi del concepte\_paraula que es vol modificar. |
| **Precondicions** | | |
| * La paraula amb l’ID especificat ja existeix al sistema. | | |
| **Postcondicions** | | |
| * El concepte\_paraula especificat per l’ID del paràmetre és retornat al client. * No s’ha modificat cap informació al servidor. | | |
| **Excepcions** | | |
| * Si no hi ha el paràmetre necessari (id\_concepte\_paraula), es retornarà un codi d’error (KO-1). * Si no hi ha cap paraula amb l’ID indicat, es retornarà un codi d’error (KO-2). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **search\_concepte\_paraula** | |
| **Descripció** | Acció que s’executa quan un usuari vol cercar paraules. El sistema cerca un text a la base de dades, en un idioma especificat, i retornarà (si és possible) una paraula amb el text que s’està cercant. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **text\_paraula** | | El text de la paraula, en l’idioma concret que indiqui l’usuari, que es vol cercar. |
| **idioma** | | El codi de l’idioma en el qual està escrita la paraula que s’està cercant. |
| **Precondicions** | | |
| * S’han omplert tots els camps necessaris en la petició. * Hi ha una paraula a la base de dades, en l’idioma indicat, que té un text coincident amb el text de cerca introduït per l’usuari. No es tindran en compte majúscules/minúscules. Sí que es tindran en compte els accents, en el cas de l’idioma català. | | |
| **Postcondicions** | | |
| * Es retorna un objecte concepte\_paraula, quan el text d’aquell concepte en l’idioma indicat per l’usuari coincideixi. * No es fa cap modificació a la base de dades. | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si no es troba cap paraula coincident, es retornarà un codi d’error (KO-2). | | |

|  |  |  |
| --- | --- | --- |
| **Nom** | **get\_sound** | |
| **Descripció** | Acció que s’executa quan un usuari vol recuperar una representació àudio d’una paraula, en un idioma especificat.  En aquest punt no s’especifica cap format de representació, sinó que ens limitem a indicar la funcionalitat a alt nivell. Deixem per a la fase d’implementació l’elecció del format de les dades àudio. | |
| **Paràmetres (els obligatoris estan en negreta)** | | |
| **id\_concepte\_paraula** | | El codi del concepte\_paraula pel qual l’usuari vol descarregar la representació àudio. |
| **idioma** | | El codi de l’idioma pel so que s’intenta descarregar. |
| **Precondicions** | | |
| * La paraula amb l’ID especificat ja existeix al sistema. | | |
| **Postcondicions** | | |
| * Una representació àudio de la paraula, en l’idioma indicat per l’usuari, és retornat al client. * No s’ha fet cap modificació al servidor (ni a la base de dades, ni al sistema de fitxers). | | |
| **Excepcions** | | |
| * Si no hi ha els paràmetres necessaris, es retornarà un codi d’error (KO-1). * Si no hi ha cap paraula amb l’ID indicat, es retornarà un codi d’error (KO-2). * Si no hi ha representació àudio per la paraula sol·licitada, es retornarà un codi d’error (KO-3). | | |

## Implementació de les accions de servidor

### Introducció

Per la implementació de les accions del servidor, s’ha fet servir un entorn de desenvolupament integrat (IDE), Eclipse (<http://www.eclipse.org>).

Com s’ha dit anteriorment, el servidor s’ha implementat sobre la plataforma J2EE. El motor triat ha estat Apache Tomcat (<http://tomcat.apache.org>), en la seva versió 7.0.25, sobre una màquina virtual Java versió 7.

Continuant amb el model MVC, s’ha implementat un servlet (*ControllerServlet*), que centralitza totes les peticions.

Un cop el ControllerServlet rep una petició, aquesta és delegada a un dels *Workers* (un per cada tipus de petició definida a la llista de peticions client – servidor), d’aquesta manera, la feina a fer per cada petició està molt localitzada i és fàcil de fer-hi millores o correccions.

Tots els *Worker* hereten d’una classe abstracta *AbstractWorker*, que té una funció principal, *processRequest()*. És en aquesta funció on hi ha la implementació de tota la lògica necessària per portar a terme l’acció que correspongui.

Per tal d’accedir a la base de dades, cada un dels *Worker* delega en la classe *DBController*, que centralitza tota la interacció amb MongoDB.

Per tal d’assegurar que totes les peticions són correctes, hi ha un parell de Filtres, *EncodingFilter* i *ServerActionFilter*. Són classes que hereten de *javax.servlet.Filter*, i es limiten a realitzar operacions sobre les peticions i les respostes HTTP i asseguren que el controlador no rebrà cap petició que no pugui processar. Concretament, *EncodingFilter* assegura que tot el trànsit d’informació es codifica correctament en el joc de caràcters apropiat, i a més estableix el *Content Type* adequat en les respostes. D’altra banda, *ServerActionFilter* verifica que la petició és vàlida (tipus de petició, nom de la petició, paràmetres obligatoris i opcionals), de manera que si no compleix amb els requisits el *ServletController* ja ni tan sols arribarà a rebre-la.

Finalment, hi ha un sistema de configuració de l’aplicació, que s’estableix a través de fitxers *properties*. Això s’utilitza pel sistema de logging, pels paràmetres de connexió a MongoDB, i pels paràmetres generals de l’aplicació.

### Convencions en aquest apartat

En aquest apartat, assumirem les següents convencions:

* El servidor Apache Tomcat estarà instal·lat a la ruta **$TOMCAT\_HOME** del servidor. També farem referència a la mateixa ruta quan parlem de **$CATALINA\_HOME** o bé de **$CATALINA\_BASE**.
* El servidor de bases de dades MongoDB està instal·lat a la ruta **$MONGODB\_HOME** del servidor. Estrictament parlant, no faria falta ni tan sols que estigués instal·lat al mateix servidor on hi ha el Tomcat. De tota manera, per simplicitat assumirem que Tomcat i MongoDB són al mateix servidor.

### Estructura de les peticions

Totes les peticions que rebrà el servidor seran peticions de tipus HTTP. L’estructura bàsica d’una petició serà aquesta:

http://host:port/accio?parametre\_1=valor\_1&parametre2\_valor\_2

En aquest cas podem veure diferents elements que analitzem tot seguit:

* Protocol: totes les peticions seran fetes amb el protocol HTTP. El filtre *ServerActionFilter* s’encarregarà de verificar-ho, i si una petició no està feta amb el protocol HTTP, serà rebutjada.
* Host: aquí s’indica la màquina on s’allotja el servidor.
* Port: per defecte, el protocol HTTP utilitza el port 80. De tota manera, es pot indicar un port alternatiu, en funció de la instal·lació concreta del servidor.
* Acció: l’acció s’indica just després del host, i a partir d’aquesta acció es com es podrà distingir quin component de software haurà de fer la feina concreta. El filtre *ServerActionFilter* serà l’encarregat de verificar que l’acció és una de les permeses, que són les següents:
  + /crear\_concepte\_paraula
  + /search\_concepte\_paraula
  + /editar\_concepte\_paraula
  + /get\_concepte\_paraula
  + /get\_sound
* Paràmetres: cada acció necessita certs paràmetres. Aquests paràmetres són diferents en funció de l’acció de què es tracti. També és el *ServerActionFilter* el qui s’encarrega de validar que els paràmetres d’una petició corresponen a l’acció demanada.

Les peticions seran sempre amb el mètode POST. Si alguna petició arriba amb un mètode HTTP incorrecte (generalment GET), el filtre *ServerActionFilter* la rebutjarà.

Quan una petició és apropiada, aquesta arribarà al controlador, que la gestionarà de la manera apropiada.

### TRACTAMENT DELS FITXERS DE SO

Els fitxers de so que s’utilitzen a l’aplicació tenen el format MP3. Sempre que es vulgui transmetre un fitxer de so entre client i servidor, primer s’haurà de codificar amb el format Base64 per tal que sigui possible de transmetre’l sense problemes a través del protocol HTTP.

El servidor, en rebre un fitxer de so, el descodificarà, i el desarà al servidor. Quan sigui el servidor el qui enviï un fitxer de so al client, el procés serà el contrari: es codificarà el fitxer i s’enviarà en la resposta.

### configuració del servidoR Tomcat

El servidor Tomcat ha d’estar correctament configurat per tal que pugui funcionar sense problemes. A la nostra aplicació això és especialment crític, tenint en compte que hi ha intercanvi d’informació d’uns tipus especials que cal considerar:

* Caràcters *kanji* japonesos.
* Fitxers d’àudio

Per defecte, Apache Tomcat assumeix que la codificació de caràcters de la URI serà ISO-8859-1. Per tal de suportar els caràcters japonesos, hem d’utilitzar UTF-8.

El paràmetre que hem de modificar és **URIEncoding**.

En les peticions POST que farem, les dades binàries (so) seran part de la capçalera de la petició.

En aquest cas, cal tenir en compte que, per defecte, Apache Tomcat té definit un tamany màxim de 8KB. Modificarem aquest valor per tal de suportar fins a 250KB, suposant que aquest és el pitjor cas.

El paràmetre a modificar és **maxHttpHeaderSize**.

Per tal que la petició POST suporti els fitxers de so sense problemes, cal establir un tamany màxim. Per defecte, Apache Tomcat suporta fins a 2MB, però és recomanable que establim el valor de manera explícita, així tenim control sobre aquesta variable.

El paràmetre que hem de modificar és **maxPostSize**.

Per tant, amb aquests canvis, el fitxer a modificar seria **$TOMCAT\_HOME/conf/server.xml**, en l’apartat del *Connector* HTTP. La configuració del connector ha de quedar així:

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443"

**maxPostSize="0"**

**maxHttpHeaderSize="250000"**

**URIEncoding="UTF-8"**

/>

### Configuració del servidor MongoDB

Una de les característiques que té el servidor de base de dades MongoDB és la poca configuració que cal fer per tenir una instal·lació estàndard funcionant en pocs minuts.

En el nostre cas no hem fet un *tuning* específic de la base de dades, perquè en la configuració per defecte és suficient per suportar la càrrega de dades que demana el sistema.

Un cop instal·lat el servidor, hem utilitzat una base de dades, que anomenem **uocpfc**, i dins d’aquesta base de dades fem servir una únic col·lecció, **words**.

Les dades que emmagatzemarem a **words** són les següents:

* **\_id**: id assignat pel servidor de la base de dades. De tipus **ObjectId**.
* **text\_catala**: el text de la paraula en català.
* **text\_japones**: el text de la paraula en japonès.
* **so\_catala**: ruta al fitxer amb el so de la paraula en català.
* **so\_japones**: ruta al fitxer amb el so de la paraula en japonès.

Un exemple de consulta d’una col·lecció amb un registre seria el següent:

lion:bin edu$ ./mongo localhost/uocpfc

MongoDB shell version: 2.2.1

connecting to: localhost/uocpfc

> db.words.find();

{ "**\_id**" : ObjectId("50c1998c5ec39f14bc014016"), "**text\_catala**" : "blau", "**text\_japones**" : "日本", "**so\_catala**" : "/tmp/ca/ca.mp3", "**so\_japones**" : "/tmp/jp/jp.mp3" }

Podem veure-hi tots els atributs d’una paraula, tal com hem comentat anteriorment.

### COnfiguració de l’aplicació web

Un cop configurats els dos servidors (Tomcat i MongoDB) és el moment de veure com es configura la pròpia aplicació.

Per tal de parametritzar l’aplicació, s’han utilitzat fitxers *properties* on s’hi indiquen els paràmetres que s’utilitzaran.

Els fitxers de *properties* són fitxers de text que l’aplicació llegeix per tal de saber quins són els valors dels paràmetres de la configuració. Un fitxer d’aquest tipus és una llista de parelles clau-valor. Cada clau (*property*) té associat un valor que serà el que l’aplicació utilitzarà quan sigui necessari.

Podem veure la configuració des de tres vessants:

* Configuració d’accés al servidor de les dades.
* Configuració de paràmetres generals de l’aplicació.
* Configuració del *logging* de l’aplicació.

#### Accés al servidor de dades

En aquest cas, el fitxer de configuració serà de la forma següent:

db.mongo.host = clone.vps

db.mongo.port = 27017

db.mongo.username = test

db.mongo.password = test

db.mongo.dbName = uocpfc

db.mongo.collectionName = words

Anem a veure amb més detall cada una de les claus especificades en aquest fitxer:

* **db.mongo.host**: El host on estarà corrent el servidor de base de dades MongoDB.
* **db.mongo.port**: El port per on serà necessari connectar-se a la base de dades, en el host indicat a la *property* anterior.
* **db.mongo.username**: L’usuari amb què l’aplicació s’autenticarà al servidor de MongoDB.
* **db.mongo.dbName**: El nom de la base de dades.
* **db.mongo.collectionName**: La col·lecció que utilitzarem per desar/consultar la informació.

#### Paràmetres generals de l’aplicació

Al fitxer de configuració de paràmetres generals de l’aplicació s’hi han de posar aquells valors que puguin canviar en algun moment al llarg de la vida de l’aplicació, sense que faci falta modificar codi font.

Actualment, aquest fitxer té aquest aspecte:

server.mp3.root = /tmp/mp3

Com es pot veure, només hi ha una parella clau-valor:

* **server.mp3.root**: ruta on es deixaran els fitxers d’àudio MP3, dins del mateix servidor on hi hagi el Tomcat, que serà el procés que els escriurà/esborrarà/llegirà.

#### *Logging* de l’aplicació

En tota aplicació és molt important de fer un *logging* adequat. El *logging* consisteix en l’escriptura de missatges per tal que els qui desenvolupen o mantenen l’aplicació puguin conèixer què passa durant l’execució.

Per tal de fer el *logging* en aquesta aplicació, hem recorregut a una solució àmpliament coneguda i provada, com és Log4J. Aquest és un *Logging Framework* lliure, que és utilitzat per un gran nombre de projectes a tots els nivells i escales. És fàcilment configurable, i permet la separació ben clara entre entorns de desenvolupament, proves i producció.

L’aspecte del fitxer és el següent:

log4j.rootLogger=DEBUG, PFC2012

# Define all the appenders

log4j.appender.PFC2012=org.apache.log4j.DailyRollingFileAppender

log4j.appender.PFC2012.File=${catalina.base}/logs/pfc2012

log4j.appender.PFC2012.Append=true

log4j.appender.PFC2012.Encoding=UTF-8

# Roll-over the log once per day

log4j.appender.PFC2012.DatePattern='.'yyyy-MM-dd'.log'

log4j.appender.PFC2012.layout = org.apache.log4j.PatternLayout

log4j.appender.PFC2012.layout.ConversionPattern = %d [%t] %-5p %c- %m%n

En aquest cas es pot veure la configuració, molt simple. El contingut que hem adjuntat és el d’un fitxer de *logging* durant el període de desenvolupament (nivell *DEBUG*).

Durant l’execució, el sistema anirà afegint línies a un fitxer (**${catalina.base}/logs/pfc2012**). Al final del dia aquest fitxer serà desat, amb la data del dia que correspongui, i se’n crearà un de nou amb el nom curt (**pfc2012**), que serà sempre el fitxer “viu” on s’escriuran totes les peticions de *logging* que corresponguin.

### Entorn(s) de desenvolupament

Per al desenvolupament del servidor s’han fet servir 3 entorns: desenvolupament, test, producció. A partir d’ara els anomenarem DEV, TEST i PROD, respectivament.

L’entorn de DEV correspon a la màquina on es fa l’escriptura de codi font, i les proves bàsiques de la funcionalitat. La base de programari en aquesta màquina és la següent:

<Característiques màquina desenvolupament>

L’entorn de TEST és una màquina virtual, sobre VirtualBox (<http://www.virtualbox.org>), amb les següents característiques:

<Característiques de la màquina de test>

L’entorn de PROD és una màquina amb les següents característiques:

<Característiques de la màquina de prod>

Per tal d’escriure el codi font del servidor, l’eina utilitzada ha estat Eclipse (<http://www.eclipse.org>).

La manera d’accedir a la base de dades MongoDB ha estat directament a través de terminal, sense fer servir entorns GUI d’accés a les dades.

### Estructura del codi del servidor

Com s’ha dit anteriorment, hem fet servir una estructura MVC (Model – View – Controller) per la implementació del servidor. En aquest apartat anem a aprofundir en l’estructura que hem utilitzat, per tal de veure cada un dels components de programari del servidor.

#### Classe PFCConstants

La interface **edu.uoc.pfc2012.edusalva.utils.PFCConstants** és una classe que conté totes les constants que s’utilitzen al llarg de l’aplicació web.

Exemples de constants definides aquí:

* Codificació de les respostes HTTP (**HTTP\_RESPONSE\_ENCODING**).
* Tipus de contingut de les respostes HTTP (**HTTP\_RESPONSE\_CONTENT\_TYPE**).
* Rutes de les peticions (**PATH\_CREATE\_CONCEPTE\_PARAULA**).
* ...

Aquestes constants podrien estar definides a un fitxer de *properties*, però pensem que té més sentit que siguin dins del codi font, perquè no és probable que canviïn en el futur.

En canvi, si la propietat és probable que canviï en el futur (per exemple, el host del servidor de base de dades), llavors aquesta propietat serà a un fitxer *properties*.

#### Els filtres

Cada cop que arriba una petició, aquesta és inspeccionada pels filtres.

Segons el fitxer *web.xml*, de configuració de l’aplicació web, l’ordre dels filtres és el següent:

<filter>

<filter-name>EncodingFilter</filter-name>

<filter-class>

edu.uoc.pfc2012.edusalva.filter.EncodingFilter

</filter-class>

</filter>

<filter>

<filter-name>ServerActionFilter</filter-name>

<filter-class>

edu.uoc.pfc2012.edusalva.filter.ServerActionFilter

</filter-class>

</filter>

El primer filtre que inspecciona la petició és el **EncodingFilter**. Aquest filtre té la tasca d’establir la codificació de les respostes HTTP. Fixem-nos que aquest filtre no fa cap operació sobre la petició, simplement assegura que la resposta està en condicions òptimes. El codi rellevant d’aquest filtre, al mètode **doFilter()**, és el següent:

res.setCharacterEncoding(PFCConstants.*HTTP\_RESPONSE\_ENCODING*);

res.setContentType(PFCConstants.*HTTP\_RESPONSE\_CONTENT\_TYPE*);

Com podem veure, aquest filtre es limita a posar el valor apropiat al joc de caràcters de la resposta, i també hi posa el **content-type** apropiat per tal que qui fa la petició tingui més informació sobre la resposta.

Els valors de les dues constants que s’estableixen són:

* **Response encoding**: UTF-8
* **Response content type**: application/json;charset=UTF-8

A continuació, entra el següent filtre, **ServerActionFilter**. Aquest filtre és l’encarregat de vigilar que les peticions que entren compleixen els requisits necessaris. Si una petició no és apropiada, no la deixa passar cap endavant. D’aquesta manera garantim que qualsevol petició que arriba al Servlet hi arriba en condicions de ser tractada de manera correcta (amb els paràmetres necessaris, amb l’acció necessària).

Les accions que executa el mètode **doFilter()** del **ServletActionFilter** són aquestes:

1. Comprovar que la petició és de tipus HTTP. Podria venir una petició no HTTP d’algun client no controlat o desconegut.
2. Comprovar que la petició s’ha fet amb el mètode HTTP correcte. La nostra aplicació del servidor només accepta peticions **POST**.
3. Comprovar el *path*. Cada acció que pot fer el servidor es detecta amb un *path* (**/crear\_concepte\_paraula**, per exemple). Si una petició arriba sense un path correcte, el filtre no la deixarà passar. El path és clau per saber quina classe Java farà la feina concreta, per la qual cosa aquesta comprovació és essencial.
4. Comprovar els paràmetres de la petició. Cada acció té uns paràmetres concrets. Es comprovarà que, en funció del *path*, els paràmetres (obligatoris i/o opcionals) siguin els que han de ser per tal que la petició es pugui atendre correctament.

Si els quatre punts de comprovació esmentats no han trobat problemes, la petició segueix endavant i arriba al Servlet.

#### El Servlet ControllerServlet

Aquest Servlet és el nucli del Controlador, en el paradigma MVC. La seva funció és rebre les peticions, encarregar la feina concreta de cada petició al mecanisme especialitzat que correspongui, i servir la resposta.

El codi d’aquest Servlet és molt simple, com podem veure a continuació:

@Override

**protected** **void** service(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

processRequest(req, res);

}

**private** **void** processRequest(HttpServletRequest req, HttpServletResponse res) {

AbstractWorker worker = WorkerFactory.*getWorker*(req, res);

worker.processRequest();

}

Podem comprovar que només hi ha dos mètodes. El primer mètode, **service()** rep les peticions que arriben. Es limita a cridar un segon mètode, **processRequest()**, que, com el seu nom indica, processa la petició.

El segon mètode obté el **Worker** apropiat per la petició de què es tracti, i crida el mètode **processRequest()** per tal que la petició sigui atesa.

#### Els Workers

En aquest punt veurem com s’estructuren els Workers, que són les classes encarregades de fer la feina concreta per cada tipus de petició.

L’estructura és molt simple, i es pot resumir en aquests punts:

* Un *worker* genèric, **AbstractWorker**, al cim de la jerarquia. Té un mètode abstracte, **processRequest()**, que és el que haurà d’implementar cada *worker* específic.
* Una *factory*, que s’encarrega, en base al *path* i els paràmetres de la petició, de crear el worker específic.

L’*AbstractWorker* té accés als objectes següents:

* Petició HTTP
* Resposta HTTP
* Path de la petició
* Paràmetres de la petició

El procés de creació d’un *worker*, per part de la *WorkerFactory* és el següent: es comprova el path de la petició, i es delega la creació a un mètode concret:

**if** (PFCConstants.*PATH\_CREATE\_CONCEPTE\_PARAULA*.equals(path)) {

**return** *createKoncept*(req, res, path, params);

} **else** **if** (PFCConstants.*PATH\_SEARCH\_CONCEPTE\_PARAULA*.equals(path)) {

**return** *searchConcept*(req, res, path, params);

} **else** **if** (PFCConstants.*PATH\_EDIT\_CONCEPTE\_PARAULA*.equals(path)) {

**return** *editKoncept*(req, res, path, params);

} **else** **if** (PFCConstants.*PATH\_GET\_CONCEPTE\_PARAULA*.equals(path)) {

**return** *getKoncept*(req, res, path, params);

} **else** **if** (PFCConstants.*PATH\_GET\_SOUND*.equals(path)) {

**return** *getSound*(req, res, path, params);

}

Es pot veure que és tan senzill com examinar el path, i en funció d’aquest, cridar el mètode apropiat de *WorkerFactory* per retornar l’objecte *worker* específic apropiat.

En el cas del mètode per la creació del *worker* de cerca de conceptes, el mètode seria així:

**private** **static** SearchKonceptWorker searchConcept(HttpServletRequest req, HttpServletResponse res, String path, Map<String, String[]> params) {

SearchKonceptWorker w = **new** SearchKonceptWorker(req, res, path, params);

**return** w;

}

Cada un dels *worker* específics tindran un constructor que inicialitzarà els valors de la petició i els paràmetres segons convingui. El constructor de **SearchKonceptWorker**, per exemple, és aquest:

**public** SearchKonceptWorker(HttpServletRequest req, HttpServletResponse res, String path, Map<String, String[]> params) {

setReq(req);

setRes(res);

setPath(path);

setParams(params);

}

En aquest cas, podem veure que simplement s’estableixen els valors dels atributs del *worker* (petició HTTP, resposta HTTP, *path*, paràmetres).

Un cop es cridi al mètode **processRequest()** d’aquest *worker*, es farà la feina realment efectiva, que consistirà en la cerca a la base de dades.

1. El sistema Leitner és un sistema de preguntes i respostes basat en fitxes inventat per Sebastian Leitner. [↑](#footnote-ref-1)
2. Kanban és un terme que pot traduir-se com a etiqueta o tiquet d'instrucció. [↑](#footnote-ref-2)
3. *JavaScript Object Notation*, sistema lleuger d’intercanvi de dades. [↑](#footnote-ref-3)