

2 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'Espectura (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.



Il·lustració 2. Selecció d'imatges kanji en el QVE

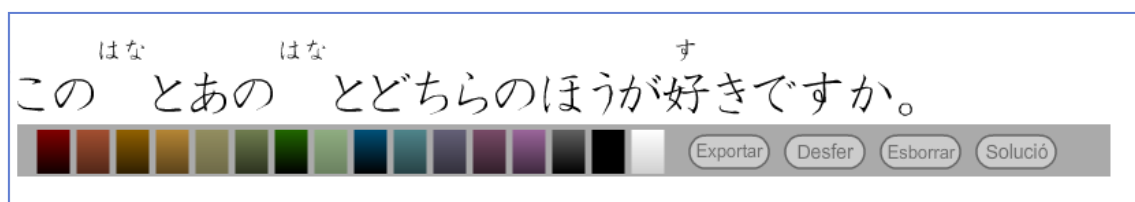
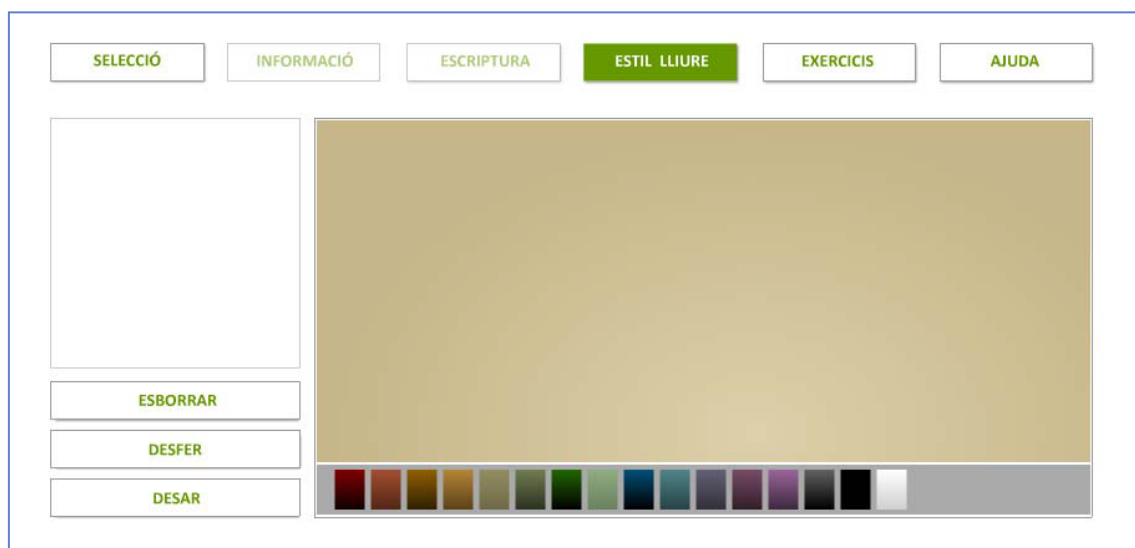
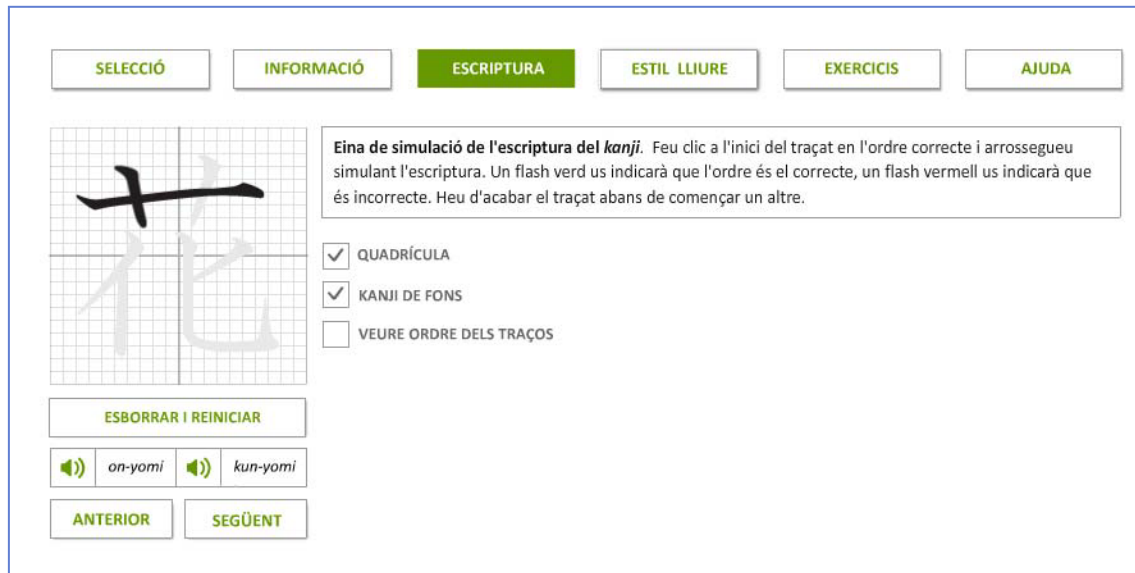
L'eina treballa amb 10 símbols *kanji* que representen 10 paraules en el nostre alfabet, i quan sel·leccionem un símbol et porta a la seva pantalla d'informació.



Il·lustració 3. 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".



Il·lustració 4. 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, llapis tàctil o ratolí (si està usant un ordinador). El sistema

Memòria

Estudiants:

Capell Brufau, Eduard – Lorca Sans, Salvador

UOC. PFC. Gener 2013

Consultor:

Roset Mayals, Roman

indica a l'usuari si l'ordre que segueix és correcte o no mitjançant un flash verd o vermell respectivament.

3 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).

3.1 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:



Il·lustració 5. Pantalles de consulta de definició (esquerra) i cerca d'una paraula (Japanese for iOS: <http://japaneseapp.com/>).

3.2 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:



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>).

3.3 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:



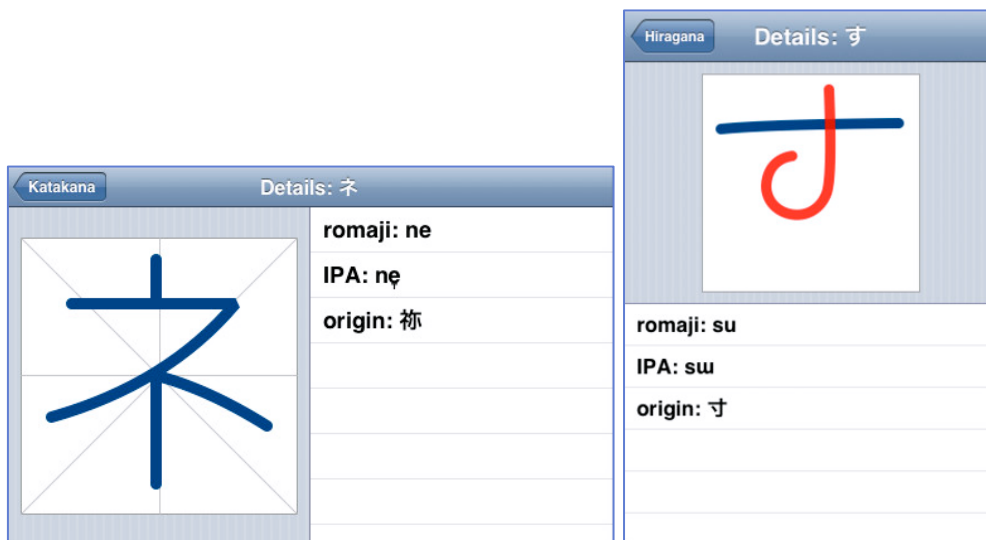
Il·lustració 7. Pantalles de pràctica amb àudio. Inclouen tant la possibilitat d'escoltar com sona una paraula, com la possibilitat de gravar la pròpia veu i re-escoltar-la posteriorment. Les captures mostrades corresponen a l'aplicació **Learn Japanese Vocabulary – Gengo** (<https://itunes.apple.com/us/app/learn-japanese-vocabulary/id294770805?mt=8>).

3.4 PRÀCTICA DE TRAÇOS

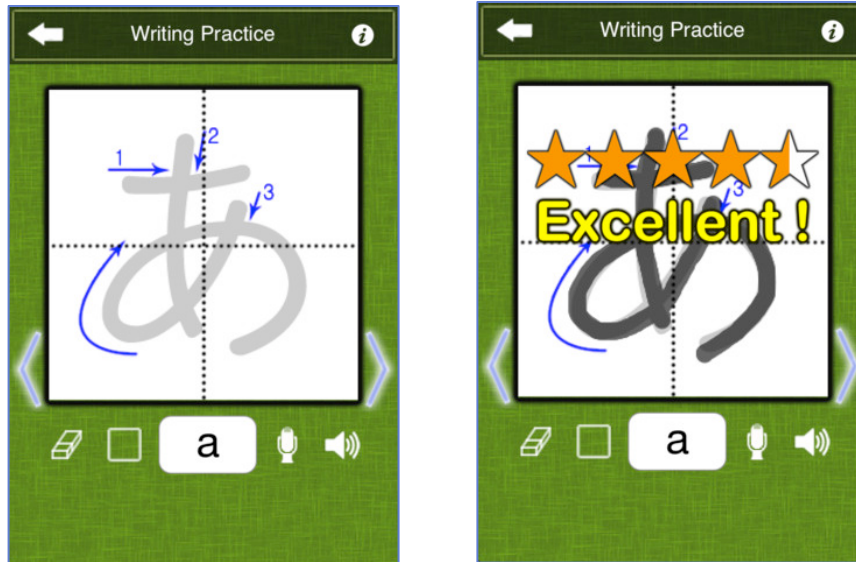
Una característica distintiva de l'idioma japonès és els traços que formen els caràcters. Donat que aquest tret constitueix un àmbit d'estudi en si mateix, és lògic que hi hagi moltes aplicacions que permeten la pràctica dels traços.

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:



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>).



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>).

4 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, 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.



Il·lustració 10. Exemple de panell Kanban

² Kanban és un terme que pot traduir-se com a etiqueta o tiquet d'instrucció.

5 ANÀLISI I DISSENY

5.1 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.



Il·lustració 11. *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.



Il·lustració 12. *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.



Il·lustració 13. *Servidor escollit, basat en Java*

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



Il·lustració 14. *Base de dades escollida*

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

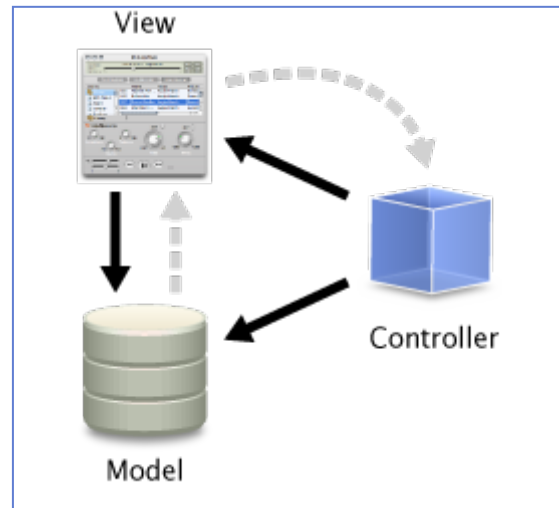


Il·lustració 15. *Sistema de control de versions del nostre projecte*

5.2 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:



Il·lustració 16. *Model Vista Controlador (MVC) del framework Sencha Touch*

- Una capa **Model** que s'encarrega de la representació específica de la informació amb la que el sistema opera. Es compon 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.

5.3 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.

5.3.1 LES PANTALLES DEL CLIENT

5.3.1.1 LLISTES D'ESTUDI



Il·lustració 17. Pantalla d'inici de l'aplicació, llistes d'estudi, i pantalla detall de la llista



Il·lustració 18. Pantalles d'edició i d'esborrat d'una llista



Il·lustració 19. Pantalla per veure el detall del concepte d'una llista

5.3.1.2 DICCIONARI



Il·lustració 20. Pantalles llistat i detall del diccionari



Il·lustració 21. Pantalles d'edició i de diàleg d'un concepte

5.3.1.3 FLASHCARDS



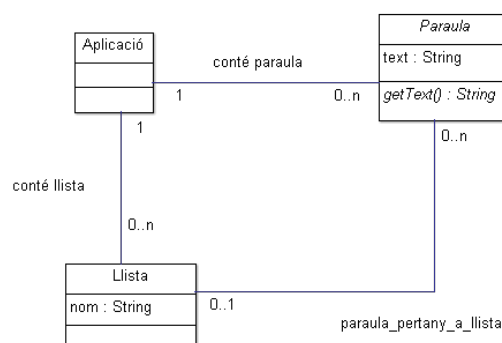
Il·lustració 22. Pantalla prèvia a l'inici de l'exercici per a escollir una llista



Il·lustració 23. Pantalles de l'exercici, anvers i revers de la targeta amb el concepte escrit

5.3.2 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:



Il·lustració 24. 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_paula*, 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_paula:** és una relació **1 - n**, indicant que l'aplicació pot contenir qualsevol nombre de paraules, incloent-hi cap paraula.
- **paraula_pertany_a_llista:** 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.

5.3.3 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.

5.3.3.1 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	
<ul style="list-style-type: none">• No hi ha al sistema una llista amb el mateix nom.	
Postcondicions	
<ul style="list-style-type: none">• S'ha creat una nova llista buida, amb el nom indicat.• Es retornarà el codi de la nova llista creada.	
Excepcions	
<ul style="list-style-type: none">• 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	
<ul style="list-style-type: none"> La llista amb l'ID especificat ja existeix al sistema. 	
Postcondicions	
<ul style="list-style-type: none"> Es retorna un objecte de tipus Llista, que conté les paraules que hi pertanyen. No s'ha modificat cap objecte (paraules o llistes). 	
Excepcions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> Hi ha una o més llistes definides al sistema. 	
Postcondicions	
<ul style="list-style-type: none"> Es retorna un conjunt de llistes. No s'ha modificat cap objecte (paraules o llistes). 	
Excepcions	
<ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Nom de la llista (<i>nom_llista</i>, de tipus text)
nou_valor	Nou valor que prendrà el camp que s'està modificant.
Precondicions	
<ul style="list-style-type: none"> La llista amb l'ID especificat ja existeix al sistema. El camp que es vol modificar és vàlid (actualment, l'únic camp admès és <i>nom_llista</i>). El valor anterior a la modificació és diferent del nou valor. 	
Postcondicions	
<ul style="list-style-type: none"> 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. 	

<ul style="list-style-type: none"> Si tot ha anat correctament, es retornarà el codi OK.
Excepcions
<ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Nom de la llista (<i>nom_llista</i>, de tipus text)
Precondicions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> La llista amb l'ID especificat ja existeix al sistema. 	
Postcondicions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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). 	

5.3.3.2 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	
<ul style="list-style-type: none"> La llista amb l'ID especificat ja existeix al sistema. 	
Postcondicions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> La caixa amb ID igual al paràmetre ha de ser una caixa existent. 	
Postcondicions	
<ul style="list-style-type: none"> Visualitzem la targeta disponible de la caixa. Si les caixes inicials i intermèdies no contenen targetes s'haurà acabat el joc. 	
Excepcions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> El concepte_paraula indicat al paràmetre existeix al sistema El concepte_paraula indicat al paràmetre no està a la caixa final. 	
Postcondicions	
<ul style="list-style-type: none"> La targeta ocupa un lloc a la següent caixa. 	
Excepcions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> El concepte_paraula indicat al paràmetre existeix al sistema El concepte_paraula indicat al paràmetre no està a la caixa final. 	
Postcondicions	
<ul style="list-style-type: none"> La targeta ocupa un lloc a la caixa inicial. 	
Excepcions	
<ul style="list-style-type: none"> Si no hi ha els paràmetres necessaris, es retornarà un codi d'error (KO-1). 	

5.4 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.

5.4.1 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 confien 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.

5.4.2 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.

5.4.3 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.

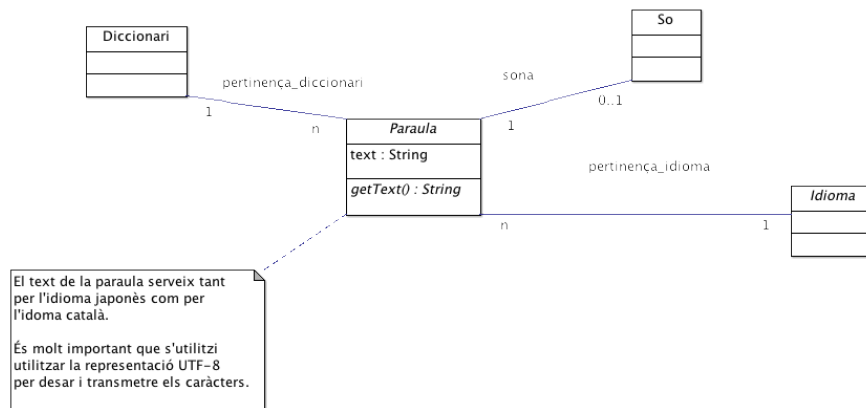
5.4.4 EL MODEL

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.

5.4.5 ELS OBJECTES DEL MODEL

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



Il·lustració 25. 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.

5.4.6 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. La versió que proposem de MySQL suporta aquesta funcionalitat.
 - 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.

5.4.7 COMUNICACIONS CLIENT – SERVIDOR

Cada comunicació entre client i servidor es farà mitjançant peticions JSON³. 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.
- **get_words**, per obtenir les paraules del servidor.
- **delete_word**, per esborrar una paraula.
- **get_paraula**, per obtenir la següent paraula d'una llista.
- **reset_game**, per posar el comptador de paraules que han aparegut a zero.

5.4.8 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 <i>kanji</i> corresponents)	
so_catala	Àudio de la paraula en català. Opcional.	
so_japones	Àudio de la paraula en japonès. Opcional.	
Precondicions		
<ul style="list-style-type: none">• L'usuari ha entrat tots els camps obligatoris.• La paraula no existeix prèviament al diccionari del servidor.		
Postcondicions		
<ul style="list-style-type: none">• La paraula ha estat donada d'alta al sistema, en els dos idiomes. Si l'usuari també		

³ JavaScript Object Notation, sistema lleuger d'intercanvi de dades.

<p>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.</p> <ul style="list-style-type: none"> El servidor retorna un codi de concepte_paraula (ID) amb el qual s'identificarà a partir d'ara l'entitat creada.
Excepcions
<ul style="list-style-type: none"> Si no tots els camps han estat emplenats, es retornarà un codi d'error al client (KO-1).

Nom	editar_camp_concepte_paraula
Descripció	<p>Acció que s'executa quan un usuari vol canviar el valor d'algun dels camps d'un concepte paraula.</p> <p>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).</p>
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 <i>kanji</i> corresponents)
so_catala	Àudio de la paraula en català. Opcional.
so_japones	Àudio de la paraula en japonès. Opcional.
Precondicions	
<ul style="list-style-type: none"> La paraula amb l'ID especificat ja existeix al sistema. Almenys un dels següents paràmetres és diferent de null: <ul style="list-style-type: none"> text_catala text_japones so_catala so_japones 	
Postcondicions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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	<p>Acció que s'executa quan un usuari vol recuperar un concepte_paraula a partir de l'ID.</p> <p>Serveix per obtenir tota la informació associada a un concepte_paraula, és a dir: textos (català i japonès) i sons (català i japonès).</p>
Paràmetres (els obligatoris estan en negreta)	
id_concepte_paraula	El codi del concepte_paraula que es vol modificar.
Precondicions	
<ul style="list-style-type: none"> La paraula amb l'ID especificat ja existeix al sistema. 	
Postcondicions	
<ul style="list-style-type: none"> El concepte_paraula especificat per l'ID del paràmetre és retornat al client. No s'ha modificat cap informació al servidor. 	
Excepcions	
<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> • 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	
<ul style="list-style-type: none"> • 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	
<ul style="list-style-type: none"> • 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	
<ul style="list-style-type: none"> • La paraula amb l'ID especificat ja existeix al sistema. 	
Postcondicions	
<ul style="list-style-type: none"> • 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	
<ul style="list-style-type: none"> • 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). 	

Nom	get_words
Descripció	Acció que s'executa per obtenir una llista de totes les paraules que hi ha al servidor.
Paràmetres (els obligatoris estan en negreta)	
max_results	El nombre màxim de resultats que ha d'enviar el servidor.
Precondicions	
<ul style="list-style-type: none"> Hi ha paraules al servidor. 	
Postcondicions	
<ul style="list-style-type: none"> El client tindrà una llista de paraules que hi ha al servidor, amb el límit del paràmetre del nombre màxim de paraules, si l'ha especificat. No s'ha fet cap modificació al servidor (ni a la base de dades, ni al sistema de fitxers). 	
Excepcions	
<ul style="list-style-type: none"> Si no hi ha cap paraula al servidor, es retornarà un missatge on s'indiqui que s'ha produït aquesta excepció. 	

Nom	delete_word
Descripció	Acció que s'executa per esborrar una paraula del servidor.
Paràmetres (els obligatoris estan en negreta)	
id	L'ID de la paraula a esborrar.
Precondicions	
<ul style="list-style-type: none"> La paraula amb l'ID especificat existeix al servidor. 	
Postcondicions	
<ul style="list-style-type: none"> El client rebrà una notificació indicant que l'operació d'esborrat ha finalitzat amb èxit. La paraula ja no estarà a la base de dades. No es marca com esborrada, sinó que es fa un esborrat físic del document a la base de dades. 	
Excepcions	
<ul style="list-style-type: none"> Si la paraula amb l'ID indicat no és al servidor, es retornarà al client un missatge indicant que no s'ha pogut eliminar amb èxit la paraula sol·licitada. 	

Nom	get_paraula
Descripció	Acció que s'executa per obtenir la següent paraula d'una llista d'estudi
Paràmetres (els obligatoris estan en negreta)	
llista	El codi de la llista d'estudi al qual ha de pertànyer la paraula que es vol obtenir.
Precondicions	
<ul style="list-style-type: none"> Hi ha alguna paraula que pertany a la llista indicada. 	
Postcondicions	
<ul style="list-style-type: none"> El client obtindrà una paraula d'entre les que hi ha al servidor i que pertanyen a la llista especificada al paràmetre, sempre i quan no hagi sortit ja la mateixa paraula en la sessió de joc actual. En el cas que hi hagi més d'una paraula que compleixi la condició anterior, se'n triarà una a l'atzar. La paraula retornada es posarà a una llista de paraules que ja han sortit, per tal que no surti durant la mateixa sessió en una petició subsegüent. 	
Excepcions	
<ul style="list-style-type: none"> Si no hi ha cap paraula al servidor que compleixi el requisit, es notificarà al client indicant-li que no hi ha hagut èxit en l'intent de recuperació de la paraula. 	

Nom	reset_game
Descripció	Acció que s'executa per tal de posar el comptador a zero pel que fa a les paraules que ja han sortit en la sessió actual.
Paràmetres (els obligatoris estan en negreta)	
<No hi ha paràmetres>	
Precondicions	
<ul style="list-style-type: none">No hi ha precondicions	
Postcondicions	
<ul style="list-style-type: none">El client rebrà la notificació que s'ha posat el comptador a zero.Poden tornar a sortir paraules que ja han sortit fins aquest moment.La llista de paraules del servidor s'ha buidat.	
Excepcions	
<ul style="list-style-type: none">Si no hi ha cap paraula al servidor, es retornarà un missatge on s'indiqui que s'ha produït aquesta excepció.	

6 IMPLEMENTACIÓ

6.1 CLIENT

6.1.1 CU_01: LLISTAT DE PARAULES

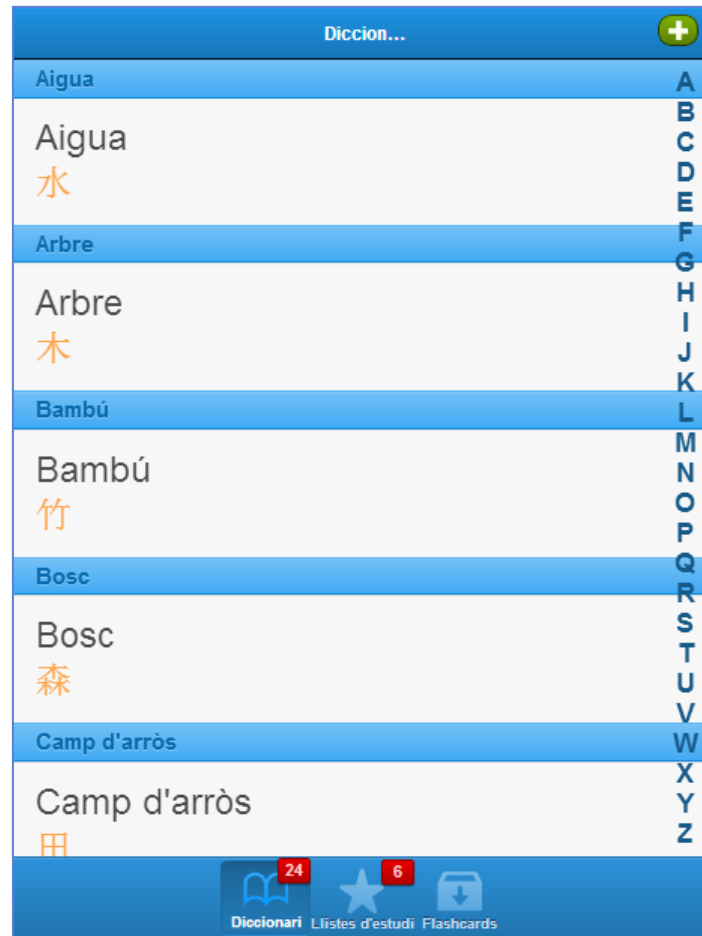
La vista inicial de l'aplicació consisteix en mostrar un llistat de paraules del manteniment "Diccionari". Es carreguen totes les paraules que ens retorna el servidor amb la petició Ajax següent:

http://eduardcapell.com/pfc2012/get_words

El llistat porta una agrupació per text de paraula en català i una barra d'indexació a la part dreta de la pantalla. Aquest component facilita molt la cerca de paraules.

Dues accions es poden portar a terme en aquesta pantalla:

- Veure el detall d'una paraula (CU_02)
- Crear nova paraula (CU_04)



Il·lustració 26. Pantalla inicial de l'aplicació. Llistat de paraules del manteniment Diccionari

6.1.2 CU_02: DETALL D'UNA PARAULA

Si hem fet clic en una paraula del llistat (CU_01) veurem el detall de la mateixa, que es compon dels elements següents:

- Texte en català
- Símbol *Kanji*
- Pronunciació en japonès

Dues accions es poden portar a terme en aquesta pantalla:

- Tornar a la pantalla anterior (CU_01)
- Veure les opcions que es poden fer amb aquesta paraula (CU_03)



Il·lustració 27. *Pantalla detall d'una paraula*

6.1.3 CU_03: OPCIONS D'UNA PARAULA

Arribam a aquest comportament si des del detall d'una paraula premem el botó amb punts suspensius de la barra superior.

Tres accions es poden portar a terme en aquesta pantalla:

- Esborrar paraula (CU_05)
- Editar paraula (CU_06)
- Cancelar (segueix en CU_02)



Il·lustració 28. Pantalla d'opcions que es poden dur a terme amb una paraula

6.1.4 CU_04: CREAR NOVA PARAULA

Si l'usuari vol inserir una paraula ho podrà fer mitjançant aquest formulari d'alta, on l'aplicació li demanarà les dades bàsiques de l'entitat Paraula.

Els tres primers camps són de tipus "text", mentre que el darrer fa aparèixer una llista desplegable amb l'opció d'escollir una llista d'estudi del manteniment Llistes d'Estudi.

El símbol *Kanji* es pot introduir, si fas servir un dispositiu amb el sistema operatiu Android, amb l'ajuda d'una aplicació de teclat que es diu GoKeyboard (veure la il·lustració 30), i permet la captura de traços amb suggeriments.

Dues accions es poden portar a terme en aquesta pantalla:

- Tornar a la pantalla anterior (CU_01)
- Guardar la paraula al diccionari (CU_01)

Il·lustració 29. Pantalla de creació de paraules

Si es prem el botó Guardar es llançarà la petició al servidor següent:

http://eduardcapell.com/pfc2012/crear_concepte_paraula?text_catala=Gos&text_japones= 犬 &pronjap=Inu&llista=2



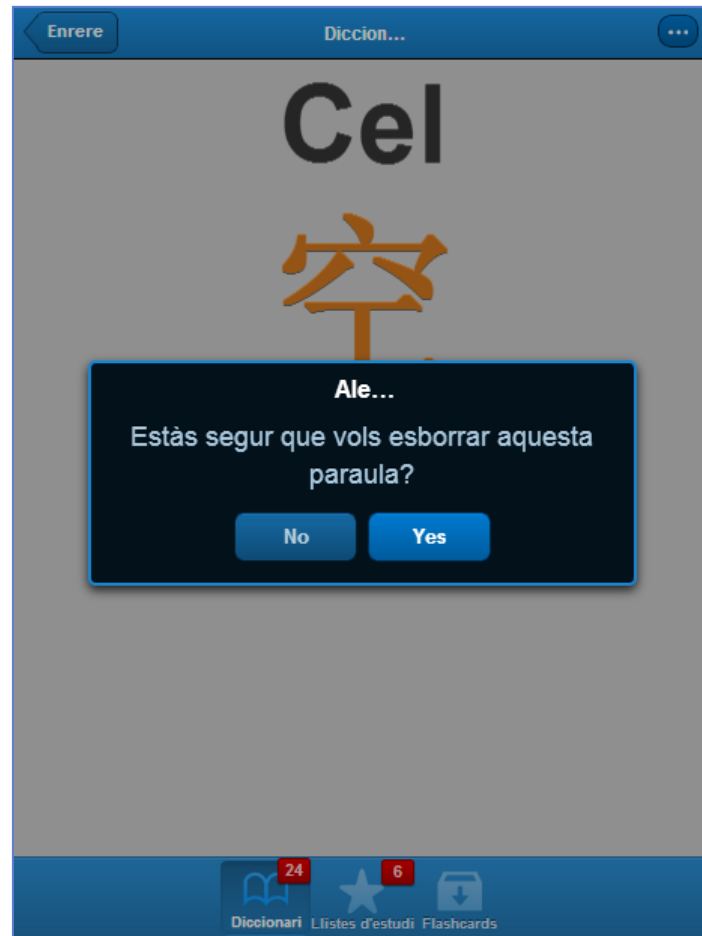
Il·lustració 30. Exemple d'introducció de símbols Kanji amb l'aplicació de teclat [GoKeyboard](#) per Android.

6.1.5 CU_05: ESBORRAR UNA PARAULA

Si al cas d'ús CU_03 hem decidit prémer el botó Esborrar arribarem a aquest diàleg que ens farà confirmar la nostra decisió.

Dues accions es poden portar a terme en aquesta pantalla:

- Confirmar l'esborrat de la paraula (CU_01)
- Cancel·lar l'acció d'esborrat (CU_02)



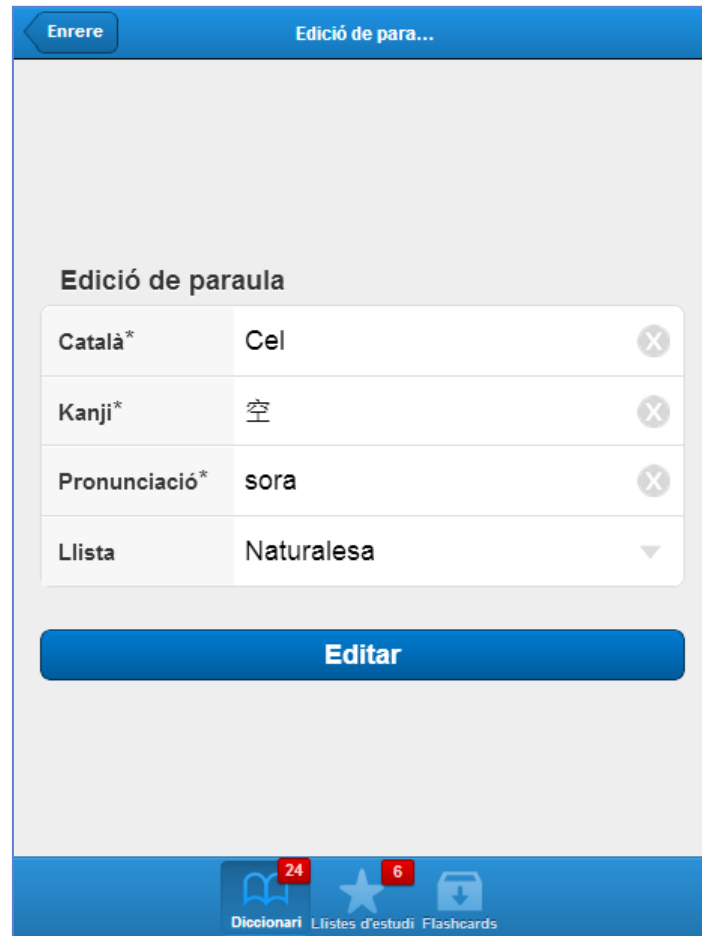
Il·lustració 31. *Pantalla d'esborrat d'una paraula*

6.1.6 CU_06: EDICIÓ D'UNA PARAULA

Si al cas d'ús CU_03 hem decidit prémer el botó Editar arribarem a aquest formulari que ens permetrà canviar algun valor. Té el mateix funcionament que el CU_04.

Dues accions es poden portar a terme en aquesta pantalla:

- Tornar a la pantalla anterior (CU_01)
- Guardar la paraula al diccionari (CU_01)



Il·lustració 32. Pantalla d'edició d'una paraula

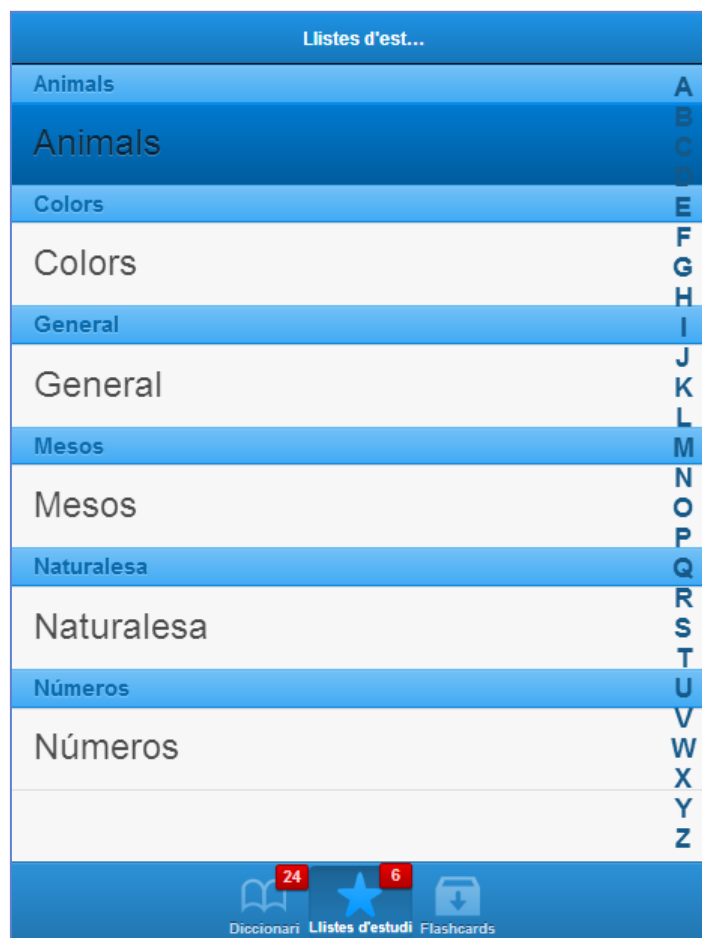
Si es prem el botó Editar es llançarà la petició al servidor següent:

6.1.7 CU_07: LLISTES D'ESTUDI

El segon manteniment de l'aplicació consisteix en mostrar una sèrie de llistes d'estudi guardades en el client. Es carreguen totes les llistes residents en el fitxer `llista.json` de la carpeta "data" del client.

El llistat porta una agrupació per nom de llista d'estudi i una barra d'indexació a la part dreta de la pantalla, exactament com al manteniment de Diccionari.

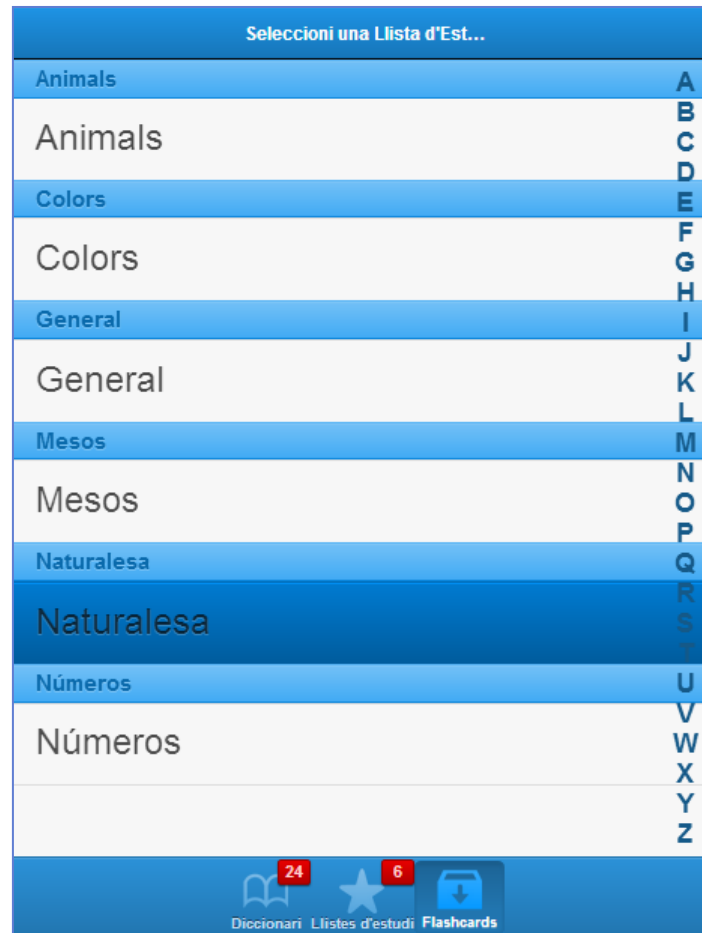
No es poden realitzar accions en aquest manteniment perquè la finalitat del projecte és gestionar paraules amb llistes d'estudi predefinides i aprendre els seus símbols *kanji* a més de la pronunciació *kun'yomi*.



Il·lustració 33. Pantalla amb les llistes d'estudi predefinides

6.1.8 CU_08: INICI DE LES FLASHCARDS

Aquesta és la pantalla d'inici del mòdul *Flashcards*. Només podem portar a terme una acció, seleccionar una llista d'estudi. Seleccionant un element del llistat de llistes d'estudi farà que l'aplicació ens presenti la primera paraula en format pregunta (CU_09).



Il·lustració 34. Pantalla per seleccionar una llista d'estudi

6.1.9 CU_09: PREGUNTA

Les targetes es mostraran en dues cares mitjançant la utilitat “carrusel” que disposa el *framework* Sencha Touch. Aquest cas d'ús contempla la cara de la targeta que conté la pregunta.

Dues accions es poden portar a terme en aquesta pantalla:

- Començar un nou exercici de *Flashcards*, amb la possibilitat d'escollir una llista d'estudi diferent (CU_08)
- Veure la resposta fent el gest *swipe* de dreta a esquerra (CU_10)



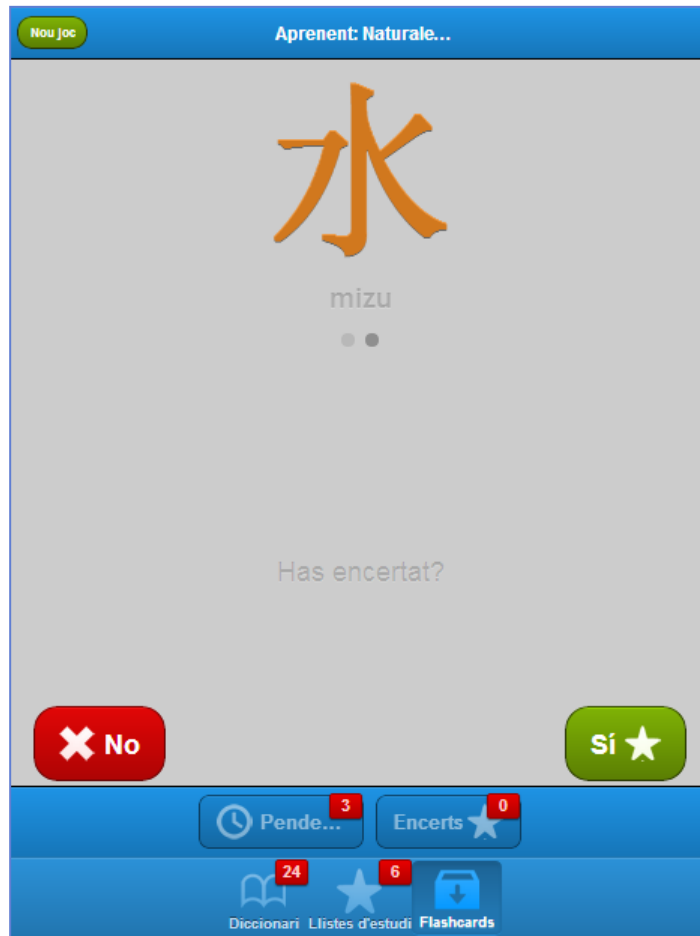
Il·lustració 35. *Pregunta de la targeta*

6.1.10 CU_10: RESPOSTA

La resposta a la pregunta del cas d'ús CU_09 apareix en aquesta cara de la targeta.

També s'habiliten dos botons que realitzaran les accions següents:

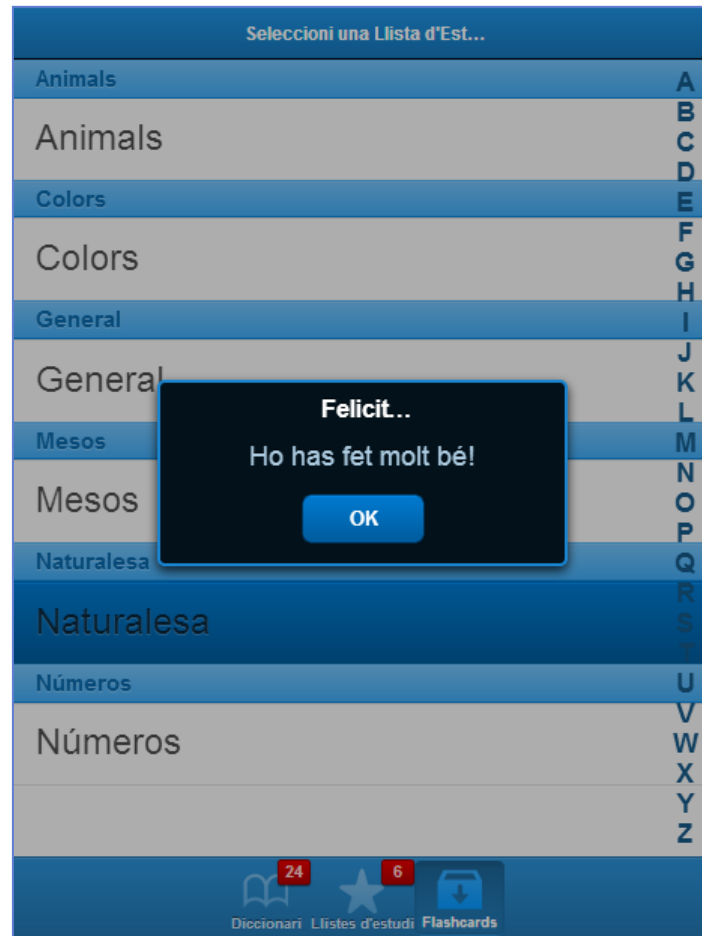
- Resposta fallida (CU_09)
- Resposta encertada (CU_09 o CU_11), incrementant el comptador de encerts i decreixent el comptador de fallades en una unitat.



Il·lustració 36. *Resposta de la targeta*

6.1.11 CU_11: ESTUDI FINALITZAT

Si l'usuari ha encertat totes les paraules de la caixa "Pendants" l'exercici arriba a la seva fi, mostrant un missatge a l'usuari i tornant al cas d'ús CU_08.



Il·lustració 37. Exercici acabat amb èxit

6.2 SERVIDOR

6.2.1 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.

6.2.2 CONVENCIIONS 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.
- Els fitxers d'àudio es desen al mateix servidor on hi ha el servidor Apache Tomcat. Això simplifica les coses de cara al desenvolupament. Si es volgués desar aquests fitxers en d'altres servidors, hi ha solucions que permeten de fer-ho de forma transparent (la nostra proposta seria utilitzar el protocol NFS, que permet desar els fitxers aparentment a una ruta local, encara que físicament són a un servidor diferent).

6.2.3 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¶metre2=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_camp_concepte_paraula
 - /get_concepte_paraula
 - /get_sound
 - /get_words
 - /delete_word
 - /get_paraula
 - /reset_game
- 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 HTTP podran ser POST o GET. Només hauran de ser obligatòriament de tipus POST quan hi hagi enviament de dades binàries des del client, la resta de peticions poden ser GET. Qualsevol altre tipus de petició que no sigui o bé GET o bé POST, serà descartada pel filtre *ServerActionFilter*.

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

6.2.4 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 envii un fitxer de so al client, el procés serà el contrari: es codificarà el fitxer i s'enviarà en la resposta.

6.2.5 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 es rebin, 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"
 />
```

6.2.6 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 única 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.
- **pronjap**: pronunciació de la paraula en idioma japonès
- **proncat**: pronunciació de la paraula en idioma català
- **llista**: codi de la llista d'estudi al qual pertany la paraula

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" :  
  "/tmp7jp/jp.mp3", "pronjap" : "neko", "proncat" : "blau", "llista" : "colors" }
```

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

6.2.7 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.

6.2.7.1 ACCÉS AL SERVIDOR DE DADES

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

```
db.mongo.host = localhost  
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.

6.2.7.2 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  
server.list.maxResults = 10
```

Veiem a continuació l'ús de cada un dels elements del fitxer:

- **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.
- **server.list.maxResults:** és el nombre màxim de resultats que pot tenir una petició de la llista de paraules. Es fa per no retornar una llista massa llarga.

6.2.7.3 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.

6.2.8 ENTORNS DE DESENVOLUPAMENT I PRODUCCIÓ

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:

- Apache Tomcat, 7.0.25
- Servidor web: Apache 2.2.22

L'entorn de TEST és una màquina virtual, sobre VirtualBox (<http://www.virtualbox.org>), amb les següents característiques (s'intenta replicar al màxim la màquina de producció):

- Linux clone 2.6.32-38-server #83-Ubuntu SMP Wed Jan 4 11:26:59 UTC 2012 x86_64 GNU/Linux
- Memòria RAM: 128MB
- Tomcat 7.0.26
- Servidor web: Nginx 0.7.65

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

- Virtual Private Server, al host eduardcapell.com
- Linux vps16530.ovh.net 2.6.38.2-xxxx-std-ipv6-64-vps #2 SMP Fri Sep 2 14:33:43 UTC 2011 x86_64 GNU/Linux
- Memòria RAM: 128MB
- Tomcat 7.0.26
- Servidor web: Nginx 0.7.65

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.

6.2.9 SERVIDOR WEB

En el cas de l'entorn de Producció, tenint en compte que el servidor és a Internet, cal tenir en compte algunes consideracions addicionals de seguretat. La màquina on s'ha fet la instal·lació (eduardcapell.com) té els mínims ports oberts, per tal d'evitar atacs. Això fa que, tot i que el servidor Tomcat escolta pel port 8080, aquest port no està obert a Internet.

El que s'ha fet ha estat col·locar un servidor web per davant del propi Tomcat. El servidor web escollit ha estat Nginx (<http://www.nginx.com>). S'ha escollit aquest servidor web per la seva robustesa i velocitat, després d'avaluar també l'alternativa del servidor web Apache (<http://httpd.apache.org>). A més, la configuració del servidor Nginx és més simple que la del servidor Apache.

La configuració del servidor Nginx pel que fa a l'aplicació Tomcat del servidor de paraules és la següent:

```
location /pfc2012 {
    access_log          /tmp/pfc2012.log;
    root                /opt/apache-tomcat-7.0.26/webapps/pfc2012;
    proxy_set_header    X-Real-IP $remote_addr;
    proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
        proxy_set_header    Host $http_host;
        proxy_set_header    X-NGINX-Proxy true;
        proxy_pass            http://localhost:8080;
        proxy_redirect        off;
    }
    location /Idiomes {
        access_log            /tmp/idiomes.log;
        root                  /var/www;
        index                  index.html;
    }
```

Podem fer les següents observacions:

- Totes les peticions a **/pfc2012** seran servides pel servidor Tomcat, en el seu port 8080.
- Totes les peticions a **/Idiomes** seran servides pel propi Nginx, a la ruta **/var/www**, on hi ha l'aplicació mòbil client en HTML5.

6.2.10 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.

6.2.10.1 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**).
- Respostes amb els codis d'error (**RESPONSE_WRONG_PARAMETERS**, etc.).
- Noms de tots els paràmetres que poden arribar en les peticions HTTP (**HTTP_REQUEST_PARAM_TEXT_CA**, etc.).
- Codi d'idioma (**LANG_CAT**, **LANG_JAP**).
- Mètodes de peticions HTTP (**HTTP_REQUEST_POST**, **HTTP_REQUEST_GET**).
- Màxim nombre de resultats per defecte (**MAX_RESULTS_DEFAULT**).
- Configuracions dels fitxers de *properties* (fitxers, nom de les claus).
- Configuracions dels noms dels camps de la base de dades.
- Rutes de les peticions (**PATH_CREATE_CONCEPTE_PARAULA**).
- Paràmetres especials que envia el *framework* Sencha. No s'utilitzen en l'aplicació però els tenim en compte per si es volen considerar com a paràmetres opcionals en alguna de les accions.

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*.

6.2.10.2 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** o **GET**.

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.

6.2.10.3 EL SERVLET CONTROLLER SERVLET

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.

6.2.10.4 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 searchKoncept(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 és aquest:

```
private static SearchKonceptWorker searchKoncept(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 aquest cas concret, en la cerca a la base de dades.

Un cop hagi fet la feina (desar a la base de dades, consultar la base de dades, etc., depenent de cada acció), es determina quina és la resposta que cal enviar al client (bean *ResponseBean*) i es crida el mètode genèric per escriure la resposta i enviar-la.

En el cas del *SearchKonceptWorker*, el codi del mètode **processRequest()** és aquest:

```
String text = getParams().get(PFCConstants.HTTP_REQUEST_PARAM_TEXT_SEARCH)[0];
String idioma = getParams().get(PFCConstants.HTTP_REQUEST_PARAM_IDIOMA)[0];
KoncepteParaula k = DBController.getKoncept(text, idioma);

ResponseBean rb = (k == null) ? new ErrorResponseBean("No s'ha trobat cap paraula"): new KonceptResponseBean(k);

writeResponse(rb);
```

Com podem veure, el *worker* li passa la tasca de fer la cerca a la classe que gestiona totes les peticions de relació amb la Base de Dades. Un cop aquesta classe ha finalitzat la cerca, es recull el resultat i s'escriu a la resposta HTTP que rebrà el client.

6.2.10.5 L'ACCÉS A LA BASE DE DADES

Per tal d'accedir a la base de dades, hem creat una estructura de controlador. Hi ha una classe, **edu.uoc.pfc2012.edusalva.db.DBController**, que s'encarrega de gestionar totes les comunicacions amb el servidor MongoDB.

Pel que fa al *driver* que és necessari per connectar-se amb el servidor, hem utilitzat el que proporciona MongoDB per connectar-se des de java, en la versió 2.9.3 (es pot obtenir en aquesta adreça: <https://github.com/mongodb/mongo-java-driver/downloads>).

Si en el futur es volgués canviar de motor de base de dades, seria tan senzill com fer alguns canvis molt simples en aquesta classe i la resta del sistema (bàsicament, els workers) no ho notaria.

La classe té dos mètodes bàsics:

- **getDBCollection()**: s'utilitza per accedir a la col·lecció de la base de dades que conté la informació rellevant. El que fa és consultar el fitxer *properties* amb la configuració de la base de dades (host, port, usuari, contrasenya, col·lecció) i intentar-se connectar amb aquests paràmetres. Si ho aconsegueix, retorna la col·lecció, per tal que sigui utilitzada per qualsevol mètode que el cridi; si no ho aconsegueix, es llença una excepció. Una mostra del codi que intenta adquirir la connexió és a continuació:

```
if (props == null) {
    try {
        props =
PFCConstants.getProperties(PFCConstants.KEY_PROPERTIES_DB_FILE);
    } catch (Exception e) {
        throw new Exception("Unable to load DB properties file.");
    }
}

try {
    m = new Mongo(
        props.getProperty(PFCConstants.PROPERTY_DB_MONGO_HOST),
```

```
Integer.parseInt(props.getProperty(PFCConstants.PROPERTY_DB_MONGO_PORT)))
;
} catch (NumberFormatException e) {
    // ...
} catch (UnknownHostException e) {
    // ...
}
m.setWriteConcern(WriteConcern.SAFE);
DB db = m.getDB(props.getProperty(PFCConstants.PROPERTY_DB_MONGO_DBNAME));
DBCollection coll =
db.getCollection(props.getProperty(PFCConstants.PROPERTY_DB_MONGO_COLLECTION_NAME));
return coll;
```

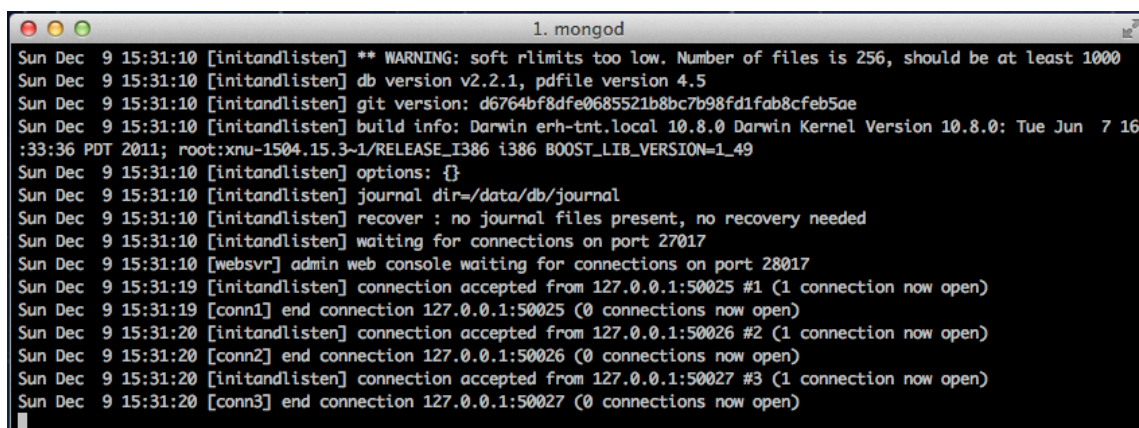
Podem veure que, un cop obtinguda la connexió amb la base de dades, s'estableix el valor del paràmetre **writeConcern** al valor segur (*safe*). Això ho fem per garantir que les operacions d'escriptura es comporten de manera que si hi ha errades o bloquejos, o puguem detectar i reaccionar-hi com sigui convenient. Si, per contra, o deixéssim en el mode normal (*normal*), el comportament seria el clàssic *fire and forget* asíncron, és a dir, es llençaria la petició d'escriptura, sense preocupar-se pel resultat; això és més ràpid, però més perillós en casos d'escriptura de registres preexistents, o d'escriptura simultània amb d'altres processos, però té com a contrapartida que és més ràpid. En el nostre cas, la velocitat no serà un problema, perquè les operacions d'escriptura no seran molt freqüents (ho seran més les de consulta), i per tant no hi ha problema per posar el valor a *safe*.

- **closeDB()**: s'utilitza per tancar la connexió amb la base de dades, un cop que s'ha finalitzat l'acció sol·licitada pel client. El codi d'aquest mètode és extremadament simple:

```
m.close();
```

On **m** és una variable de tipus **com.mongodb.Mongo**, que representa la connexió amb el servidor.

Si monitorem el log del servidor MongoDB, podem veure com, a cada petició, s'obre una connexió, i a continuació es torna a tancar. A la captura de pantalla següent podem veure com hi ha tres peticions successives. Cada una, en acabar, tanca la connexió amb el servidor, com es desprèn sempre del darrer missatge (*0 connections now open*):



```
1. mongod
Sun Dec 9 15:31:10 [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
Sun Dec 9 15:31:10 [initandlisten] db version v2.2.1, pdfile version 4.5
Sun Dec 9 15:31:10 [initandlisten] git version: d6764bf8dfe0685521b8bc7b98fd1fab8cfcb5ae
Sun Dec 9 15:31:10 [initandlisten] build info: Darwin erh-tnt.local 10.8.0 Darwin Kernel Version 10.8.0: Tue Jun 7 16
:33:36 PDT 2011; root:xnu-1504.15.3~1/RELEASE_I386 i386 BOOST_LIB_VERSION=1_49
Sun Dec 9 15:31:10 [initandlisten] options: {}
Sun Dec 9 15:31:10 [initandlisten] journal dir=/data/db/journal
Sun Dec 9 15:31:10 [initandlisten] recover : no journal files present, no recovery needed
Sun Dec 9 15:31:10 [initandlisten] waiting for connections on port 27017
Sun Dec 9 15:31:10 [websvr] admin web console waiting for connections on port 28017
Sun Dec 9 15:31:19 [initandlisten] connection accepted from 127.0.0.1:50025 #1 (1 connection now open)
Sun Dec 9 15:31:19 [conn1] end connection 127.0.0.1:50025 (0 connections now open)
Sun Dec 9 15:31:20 [initandlisten] connection accepted from 127.0.0.1:50026 #2 (1 connection now open)
Sun Dec 9 15:31:20 [conn2] end connection 127.0.0.1:50026 (0 connections now open)
Sun Dec 9 15:31:20 [initandlisten] connection accepted from 127.0.0.1:50027 #3 (1 connection now open)
Sun Dec 9 15:31:20 [conn3] end connection 127.0.0.1:50027 (0 connections now open)
```

Il·lustració 38. Pantalla de log del servidor MongoDB

A més d'aquests dos mètodes fonamentals, **DBController** també conté un mètode per cada acció (creació de concepte, consulta de concepte, etc.).

Apart d'aquests mètodes, n'hi ha algun d'accessori que es fa servir per tal de fer comprovacions. Per exemple, el mètode **konceptExists()**, que verifica si un concepte paraula ja existeix a la base de dades.

Les consultes realitzades sobre el servidor MongoDB des de la classe **DBController** són les següents:

- **Consulta per saber si un concepte existeix:** consisteix a verificar si una paraula existeix a la base de dades. Es verifiquen els dos camps d'idioma (català i japonès). No es fa cap validació dels fitxers de so, només dels textos en els dos idiomes. Seria equivalent a aquesta consulta: **db.words.find({"text_catala" : "blau", "text_japones": "日本"})**; A continuació es pot veure el codi utilitzat per la cerca:

```
DBCollection col;  
  
try {  
    BasicDBObject query = new BasicDBObject();  
    query.put(PFCConstants.DB_FIELD_TEXT_CA, k.getTextCatala());  
    query.put(PFCConstants.DB_FIELD_TEXT_JP, k.getTextJapones());  
  
    col = getDBCollection();  
  
    DBCursor cursor = col.find(query);  
    if (cursor != null && cursor.hasNext()) {  
        return true;  
    }  
} catch (Exception e) {  
    e.printStackTrace();  
    return true;  
} finally {  
    closeDB();  
}  
  
return false;
```

- **Consulta per crear un nou concepte paraula:** creació d'un nou registre a la col·lecció *words*. En aquest cas, és equivalent a la consulta **db.words.save({"text_catala": 'vermell', text_japones: '日本', so_catala: '/tmp/ca/vermell.mp3', so_japones: '/tmp/jp/vermell.mp3'})**; A continuació podem veure el codi per crear un nou concepte paraula:

```
DBCollection coll;  
  
try {
```



```
coll = getDBCollection();

BasicDBObject doc = new BasicDBObject();

doc.put(PFCConstants.DB_FIELD_TEXT_CA, k.getTextCatala());
doc.put(PFCConstants.DB_FIELD_TEXT_JP, k.getTextJapones());

coll.insert(doc);

ObjectId id = (ObjectId)doc.get(PFCConstants.DB_FIELD_ID);

return id.toString();

} catch (Exception e) {

    logger.error(e.getMessage());

} finally {

    closeDB();

}

return null;
```

- **Consulta per obtenir un concepte paraula:** a partir del text (en català o en japonès), s'obté el concepte paraula, amb tots els camps necessaris. És l'equivalent a la consulta `db.words.find({text_catala: 'blau'})`; No enganxem aquí el codi, per la seva extensió, però resumint podem dir que es limita a dues operacions:
 - Construcció de l'objecte **query** (`com.mongodb.BasicDBObject`), posant valors als camps que convingui en funció de la consulta que ens hagi arribat.
 - Execució de la consulta, i obtenció d'un cursor (`com.mongodb.DBCursor`) per tal de recuperar les dades que volem retornar.
- **Consulta per obtenir un concepte paraula a partir del seu id:** molt similar a la consulta anterior, però ara buscant només per **id**. La diferència és que ara ja sabem el camp pel qual busquem, i a més sabem que la cerca només pot retornar un resultat (per tant, cridarem el mètode `findOne()` enlloc del mètode `find()` com fèiem al punt anterior).
- **Consulta per actualitzar un concepte paraula:** aquest mètode substitueix, en l'objecte amb un **id** especificat, els valors no nuls que s'indiquin. Per exemple, si volem canviar el text català d'una paraula, el mètode rebrà un objecte de tipus **KoncepteParaula** amb un **id** i un **textCatala**. Un cop tinguem clar quins camps s'han d'actualitzar, farem una crida al mètode `update()` de l'objecte **Collection**.

6.2.10.6 ELS BEANS

Aquesta aplicació té tres beans fonamentals: el que representa el concepte paraula (`edu.uoc.pfc2012.edusalva.bean.KoncepteParaula`); el que representa una petició HTTP (`edu.uoc.pfc2012.edusalva.bean.PFC2012Request`); i el que representa una resposta que s'enviarà al client (`edu.uoc.pfc2012.edusalva.bean.ResponseBean`).

El bean **KoncepteParaula** és molt simple, té cinc propietats (**id**, **text català**, **text japonès**, **so català**, i **so japonès**). Les propietats rellevants per tal d'establir si dos objectes d'aquest tipus són iguals són el **text català** i el **text japonès**.

El bean **PFC2012Request** és un bean que representa una petició que rep el sistema, i té aquestes quatre propietats:

- *path*, un String que representa la ruta de la petició (per exemple, “/get_concepte”).
- *mandatory*, de tipus *Set<String>*, que conté els noms dels paràmetres obligatoris d'aquesta petició.
- *optional*, de tipus *Set<String>*, que conté els noms dels paràmetres opcionals que admet aquesta petició.
- *minimumOptional*, de tipus *int*, que indica quin és el nombre mínim de paràmetres opcionals que ha de contenir aquesta petició.

Quan s'inicialitza un objecte de tipus **PFC2012Request** es fa en base al *path*. A partir d'això es posa valors en les llistes de paràmetres, al mètode **setupParameters()**, cridat pel constructor.

Es considera que dos objectes de tipus **PFC2012Request** són iguals quan els seus *paths* són iguals.

El bean **ResponseBean** encapsula una resposta al client, i consisteix en un atribut booleà que indica si la petició ha estat exitosa o no (*success*). Aquesta és una classe abstracte, que llavors té unes subclasses que la implementen, en funció de quina resposta s'ha d'enviar:

- **ErrorResponseBean**: per enviar una resposta que indica que s'ha produït un error, i, opcionalment, afegir-hi un missatge d'error.
- **KonceptIdResponseBean**: per enviar una resposta que conté un ID de paraula.
- **KonceptResponseBean**: per enviar una resposta que conté un Koncepte sencer.
- **SuccessResponseBean**: per indicar que la petició ha anat correctament, sense retornar cap resultat específic (habitual en casos com el d'esborrar o actualitzar una paraula).
- **WordListResponseBean**: per retornar una llista d'objectes Koncepte.

6.2.10.7 LA CLASSE PFCUTILS

Aquesta classe conté mètodes d'utilitats, tots ells estàtics, que no es poden posar en cap altra classe concreta.

Té un bloc de codi estàtic que serveix per inicialitzar tots els tipus diferents de peticions (objectes de tipus **PFC2012Request**) que hi ha al sistema.

Els mètodes els llistarem a continuació, per veure la funcionalitat de cadascun.

- **getProperties()**: retorna un objecte *java.util.Properties*, referenciant el fitxer que convingui segons el paràmetre rebut (pot ser el de base de dades o el de configuració general del servidor).
- **checkPath()**: serveix per verificar que el *path* rebut pel servidor en una petició és correcte. Només es verifica que sigui un dels *paths* existents.
- **checkRequestMethod()**: verifica que la petició s'ha fet amb el mètode apropiat (POST).
- **checkRequestParameters()**: comprova que la petició conté els paràmetres correctes (obligatoris i/o opcionals).
- **saveBase64()**: agafa un *String* codificat en Base64, el descodifica i el desa al disc.
- **getBase64FromFile()**: obre un fitxer, el codifica en Base64, i el retorna en format *String*.

6.2.11 EL PROCÉS D'EMPAQUETATGE I DESPLEGAMENT

El procés d'empaquetatge (creació d'un fitxer **war**) és molt simple, gràcies a la creació inicial dels fitxers de construcció (*build files*), amb l'ajuda de l'eina Ant (<http://ant.apache.org/>).

Ant és una eina que ens permet d'automatitzar processos repetitius, configurant-los en un fitxer XML (els *build files*). En el nostre cas, les tasques que realitza Ant són les següents:

- Esborrat dels directoris de destinació del nou fitxer a desplegar.
- Creació dels directoris de destinació del nou fitxer a desplegar.
- Compilació del codi font java.
- Empaquetatge en un fitxer *war*.
- Desplegament al servidor.

Durant el procés de desenvolupament, s'han utilitzat tres fitxers *build.xml* diferents (un per l'entorn de desenvolupament, un per l'entorn de testing, un per l'entorn de producció).

L'IDE Eclipse està molt ben integrat amb Ant, de manera que quan es volia desplegar una nova versió, era tan senzill com triar el fitxer *build* apropiat (dev, test, prod) i dir que aquell era el que calia instal·lar. Un cop fet això, el procés automàticament creava els fitxers necessaris i els pujava a l'entorn corresponent.

Per tal que cada entorn pogués tenir configuracions diferents (per exemple usuaris o contrasenyes diferents de la base de dades diferents, o diferents nivells de logging, o diferents rutes d'emmagatzematge dels fitxers d'àudio), hi havia uns fitxers que tenien extensió **prod** o **test** segons si es tractava de fitxers de l'entorn de desenvolupament o no. Aquests fitxers són:

- db.properties.prod / db.properties.test
- log4j.properties.prod / log4j.properties.test
- pfc2012.properties.prod / pfc2012.properties.test

Llavors, al fitxer *build*, l'apartat de la tasca de compilació és de la següent manera:

```
<target name="compile" depends="init">
    <echo message="SRC = ${src}" />

    <javac includeantruntime="no" srcdir="${src}" destdir="${build}"
    classpathref="compilation_libs" debug="true"/>

    <echo message="Copying properties files ..." />

    <copy file="${log4j_file}.${environment}" tofile="${build}/${log4j}" />

    <copy file="${dbconfig_file}.${environment}" tofile="${build}/${dbconfig}" />

    <copy file="${server_config_file}.${environment}" tofile="${build}/${server_config}" />
</target>
```

Podem veure que es copien els fitxers de configuració seguits de l'extensió en funció de la variable *environment*, de manera que en cada fitxer *build* es copia el fitxer apropiat.

6.2.12 LLIBRIES UTILITZADES

Les llibreries principals que s'utilitzen en la implementació del servidor són les següents (posteriorment les analitzarem amb més detall):

- **Commons Codec 1.7:** s'utilitza per realitzar les funcions de conversió a i de format Base64, imprescindible per l'intercanvi d'objectes binaris a través de peticions HTTP.
- **Commons IO 2.4:** per funcions de tractament de fitxers.
- **Jackson 2.1.0:** per realitzar funcions de serialització de dades d'objectes Java a streams JSON per a ser enviats al client.
- **Log4j 1.2.16:** per les funcions de Logging.
- **Mongo 2.9.3:** per connectar-se amb bases de dades MongoDB.

Totes les llibreries utilitzades són Open Source

6.2.12.1 COMMONS CODEC 1.7

Disponible a <http://commons.apache.org/codec/>, aquesta llibreria proporciona un conjunt d'implementacions de codificadors i descodificadors (Base64, Hex, Phonetic, URL).

En aquesta implementació del servidor s'utilitza en dos moments:

- 1) Si arriba l'àudio del client, aquest arribarà codificat en Base64. En aquest cas, cal descodificar-ho per tal d'obtenir les dades binàries, i escriure-les al fitxer.
- 2) Si s'ha d'enviar l'àudio al client, cal convertir les dades binàries en cadena de text, per la qual cosa s'ha de codificar el fitxer en format Base64 abans d'enviar-lo en la resposta.

Commons Codec està disponible sota la llicència Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>).

6.2.12.2 COMMONS IO 2.4

Disponible a <http://commons.apache.org/io/>, aquesta llibreria consisteix en un conjunt d'utilitats que fan més fàcil l'accés i la manipulació de fitxers.

En el codi de la implementació que proposem es fa servir en el punt on cal convertir un fitxer en un *array* de *bytes*.

Commons IO està disponible sota la llicència Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>).

6.2.12.3 JACKSON 2.1.0

Disponible a <http://jackson.codehaus.org/Home>, dona suport a l'escriptura de dades Java en format JSON per tal de ser enviades com a resposta a les peticions del client.

Gràcies a l'ús d'aquesta llibreria es fa molt fàcil la serialització d'objectes Java en un format JSON, ja que només és qüestió de treballar amb un objecte de la classe **com.fasterxml.jackson.databind.ObjectMapper**. Cridant al mètode **writeValue()** d'aquesta classe, passant-li l'objecte que volem enviar, la llibreria Jackson s'encarrega, per mitjà de *reflection* (<http://docs.oracle.com/javase/tutorial/reflect/index.html>), de trobar els atributs de l'objecte a escriure, i els serialitza a la resposta que s'envia al client.

Jackson està disponible en dues llicències, permetent que els qui el facin servir utilitzin el més apropiat pel seu projecte:

- Apache License 2.0: <http://www.apache.org/licenses/LICENSE-2.0>
- LGPL 2.1: <http://www.gnu.org/licenses/lgpl.html>

6.2.12.4 LOG4J 1.2.16

Aquesta és la llibreria de *logging* més utilitzada en les aplicacions escrites en Java actualment. Està disponible a <http://logging.apache.org/log4j/1.2/>.

Proporciona un *framework* molt flexible i potent per tal que qualsevol aplicació Java (de servidor o no) pugui escriure missatges de l'estat de l'aplicació. És molt senzilla de configurar, a través de fitxers *properties*, i permet d'escriure a tot tipus de destinacions (fitxers de text, bases de dades, correus electrònics, etc.).

Log4j està disponible sota la llicència Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>).

6.2.12.5 MONGO 2.9.3

Disponible a <https://github.com/mongodb/mongo-java-driver>, consisteix en un conjunt de classes que permeten que una aplicació Java es connecti a una base de dades MongoDB.

El Mongo Java Driver està disponible sota la llicència Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>).

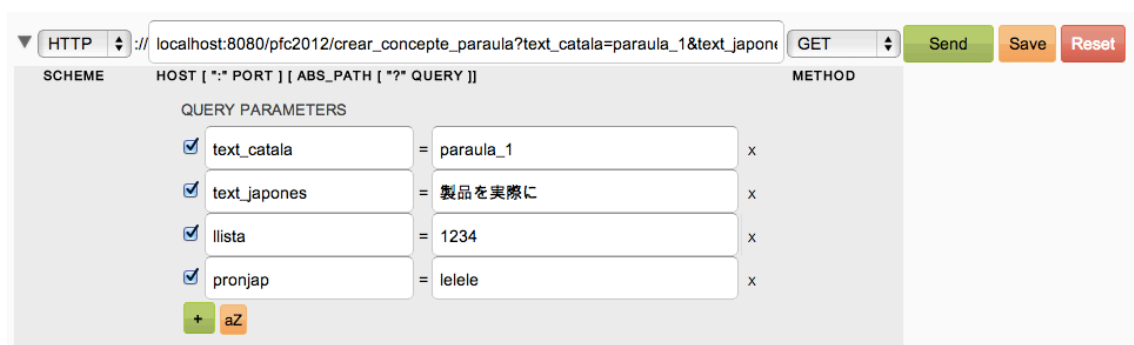
6.2.13 SIMULACIÓ DE PETICIONS AL SERVIDOR

Per tal de fer les proves i veure com es comporta el servidor quan rep peticions, hem fet servir una eina que permetia de reproduir peticions i analitzar-ne la resposta del servidor.

L'eina utilitzada ha estat **Dev HTTP Client** (<http://goo.gl/ZXivx>). És una eina pel navegador Google Chrome (versió 22 o superior). És molt flexible, i permet d'especificar opcions diverses (mètode de la petició, paràmetres, adjunció de fitxers, etc.). A més, quan la resposta és en JSON, mostra els detalls gràficament, la qual cosa facilita l'anàlisi dels resultats. A continuació veurem algunes de les proves fetes.

6.2.13.1 TEST DE CREACIÓ DE PARAULA NOVA

En aquesta prova, mirarem de crear una paraula. Aquí veiem com construïm la petició:



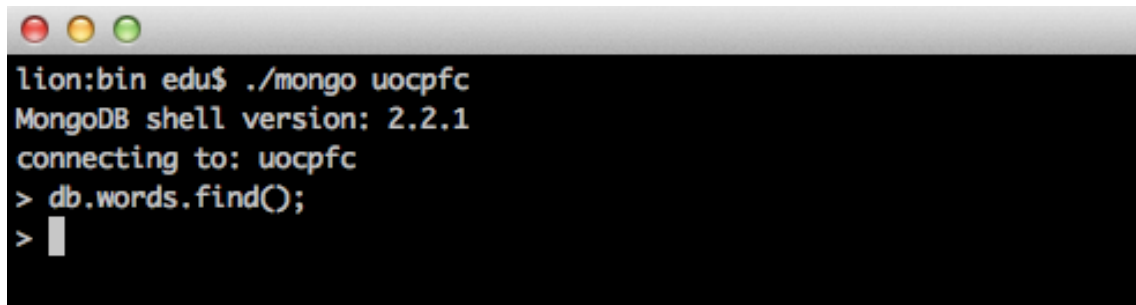
Il·lustració 39. Pantalla de l'aplicació Dev HTTP Client, per fer proves amb peticions al servidor

Característiques de la petició:

- Host: localhost
- Ruta a l'aplicació web: pfc2012
- Path: crear_concepte_paraula
- Paràmetres:

- text_catala
- text_japones
- llista
- pronjap
- Mètode: GET

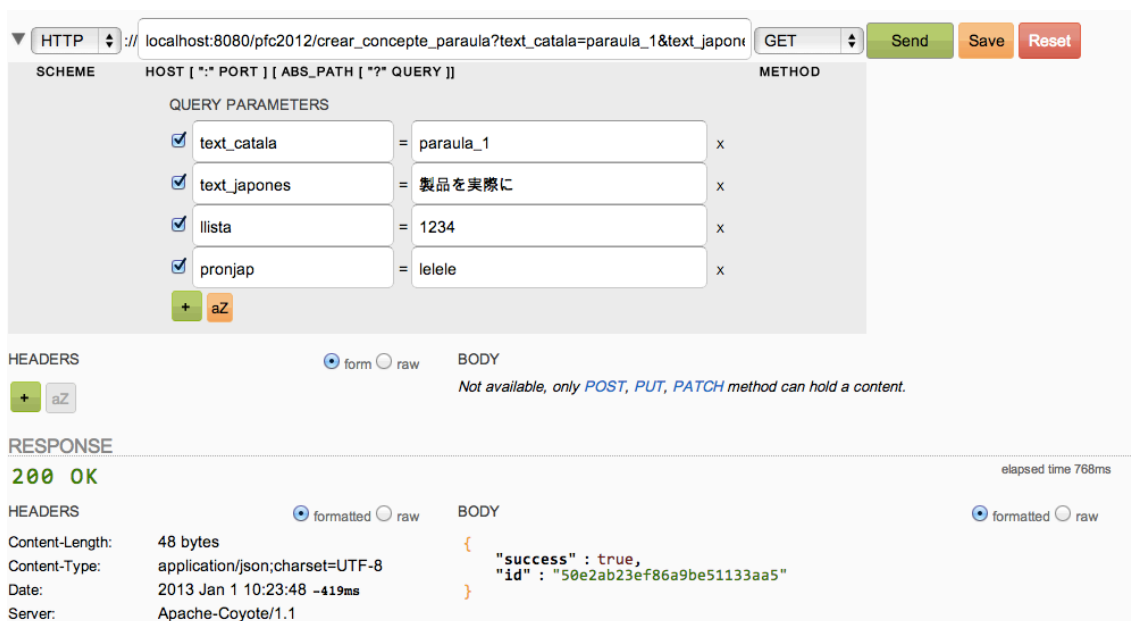
Quan fem clic al botó 'SEND', s'enviarà la petició al servidor, que la processarà. Fem la prova sobre un servidor on encara no hi ha cap registre a la base de dades:



```
lion:bin edu$ ./mongo uocpfc
MongoDB shell version: 2.2.1
connecting to: uocpfc
> db.words.find();
>
```

II·lustració 40. Servidor de base de dades MongoDB, sense registres

Un cop fem la petició, la pantalla del Dev HTTP Client ens queda així:



HTTP GET localhost:8080/pfc2012/crear_concepte_paraula?text_catala=paraula_1&text_japones=製品を実際に

QUERY PARAMETERS

Parameter	Value
text_catala	paraula_1
text_japones	製品を実際に
llista	1234
pronjap	lelele

RESPONSE: 200 OK

Content-Type: application/json; charset=UTF-8

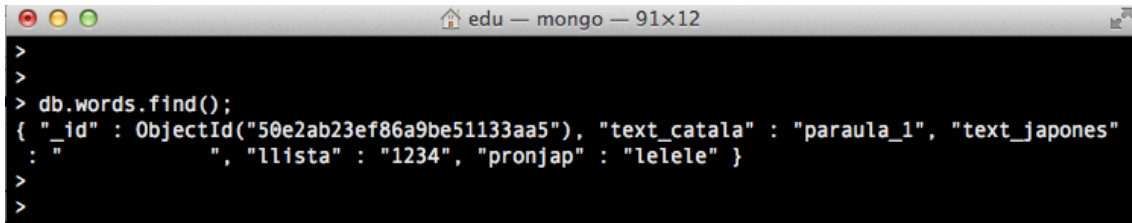
Body: {"success": true, "id": "50e2ab23ef86a9be51133aa5"}

II·lustració 41. Resposta del servidor a una petició de creació de nova paraula

Observem que a la part inferior (secció 'RESPONSE') veiem el cos de la resposta ('BODY'), on hi trobem una tira de caràcters. Aquests caràcters són l'ID de la paraula que s'acaba de crear al servidor, això vol dir que la paraula s'ha creat correctament, de manera que tot ha anat correctament. Comprovem, a més, que la resposta està en format JSON:

- Atribut *success*, que indica si la petició ha tingut èxit o no.
- Atribut *id*, que conté l'ID de la paraula acabada de crear.

Si mirem ara el contingut de la base de dades:



```
>
>
> db.words.find();
{ "_id" : ObjectId("50e2ab23ef86a9be51133aa5"), "text_catala" : "paraula_1", "text_japones" : "製品を実際に", "llista" : "1234", "pronjap" : "lelele" }
>
>
```

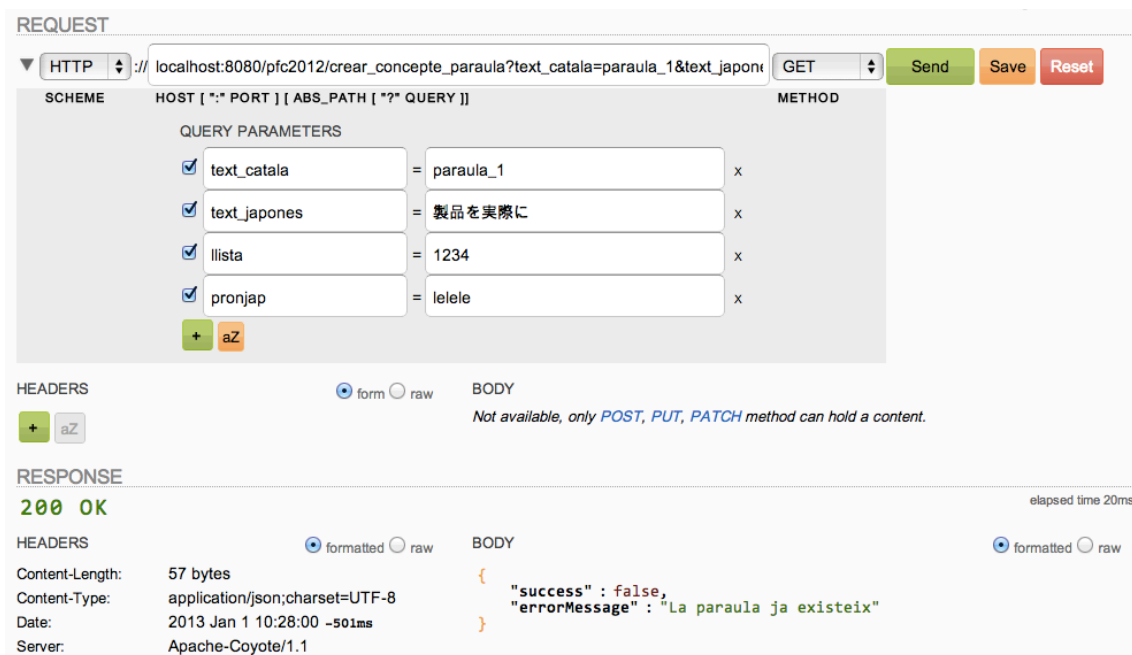
Il·lustració 42. Consulta de registres a la base de dades, amb la paraula recent creada

6.2.13.2 TEST DE CREACIÓ DE PARAULA EXISTENT

Ara anem a veure què passa quan intentem crear una paraula que ja existeix.

Recordem que una paraula es considera com ja existent en el cas que tant el text en català com el text en japonès siguin els mateixos que una que ja sigui a la base de dades.

La petició que farem serà exactament igual que en el cas anterior. Per tant, ara ens hauria de retornar un codi d'error que indiqui que no es pot crear la paraula, perquè ja està creada:



REQUEST

HTTP localhost:8080/pfc2012/crear_concepte_paraula?text_catala=paraula_1&text_japone GET Send Save Reset

SCHEME **HOST** [":" PORT] [ABS_PATH ["?" QUERY]] **METHOD**

QUERY PARAMETERS

<input checked="" type="checkbox"/> text_catala	=	paraula_1	x
<input checked="" type="checkbox"/> text_japones	=	製品を実際に	x
<input checked="" type="checkbox"/> llista	=	1234	x
<input checked="" type="checkbox"/> pronjap	=	lelele	x

HEADERS **form** **raw** **BODY**

Not available, only POST, PUT, PATCH method can hold a content.

RESPONSE

200 OK elapsed time 20ms

HEADERS **formatted** **raw** **BODY**

Content-Length: 57 bytes
Content-Type: application/json; charset=UTF-8
Date: 2013 Jan 1 10:28:00 -501ms
Server: Apache-Coyote/1.1

```
{
  "success": false,
  "errorMessage": "La paraula ja existeix"
}
```

Il·lustració 43. Resultat de l'intent d'inserció de paraula ja existent a la base de dades

Com podem veure, ara la resposta conté l'atribut *success*, però amb el valor *false*, que indica que no s'ha pogut crear la paraula. A més, hi ha un missatge d'error, *errorMessage*, que indica què ha passat.

6.2.13.3 TEST DE CONSULTA DE PARAULA PER ID

Ara veurem com veiem les dades d'una paraula que consultem a partir de l'ID de la mateixa.

Utilitzarem l'ID de la paraula creada dos passos enrere (50e2ab23ef86a9be51133aa5).

The screenshot shows a web client interface with a 'REQUEST' section at the top. The URL is 'localhost:8080/pfc2012/get_concepte_paraula?id=50e2ab23ef86a9be51133aa5' and the method is 'GET'. The 'QUERY PARAMETERS' section shows 'id' set to '50e2ab23ef86a9be51133aa5'. The 'RESPONSE' section shows a '200 OK' status and a JSON body:

```
{  "success": true,  "concept": {    "id": "50e2ab23ef86a9be51133aa5",    "idllista": "1234",    "textcat": "paraula 1",    "textjap": "製品を実際に",    "pronjap": "lelele"  }}
```

Il·lustració 44. *Petició de paraula a partir de l'ID*

Observem que en la resposta, en format JSON, hi veiem, per una banda, que la petició ha tingut una resposta correcta (*success: true*), i per altra banda hi veiem els camps que hi hem entrat anteriorment quan l'hem creada.

Si, en canvi, haguéssim intentat consultar una paraula que no existia, haguéssim obtingut aquesta resposta:

The screenshot shows the same web client interface but with the ID changed to '50e2ab23ef86a9be51133aa'. The 'RESPONSE' section shows a '200 OK' status and a JSON body:

```
{  "success": false,  "errorMessage": "No s'ha trobat la paraula."}
```

Il·lustració 45. *Resposta a petició de paraula per ID sense resultat*

Podem veure que, en no haver-hi cap paraula amb l'ID indicat, es retorna el valor de *success* a *false*, i s'indica quin és el problema que hi ha hagut.

6.2.13.4 TEST DE CONSULTA DEL LLISTAT DE PARAULES

Quan volem fer una consulta de les paraules que hi ha al servidor, podem veure que fent la petició **get_words** obtenim la llista de paraules (inclou la que hem creat anteriorment, i una altra que hem creat per comprovar que veiem totes les que hi ha al servidor):

HTTP GET localhost:8080/pfc2012/get_words

SCHEME HOST [":" PORT] [ABS_PATH ["?" QUERY]] METHOD

QUERY PARAMETERS

HEADERS

BODY

Not available, only POST, PUT, PATCH method can hold a content.

RESPONSE

200 OK

elapsed time 33ms

HEADERS

BODY

Content-Length: 260 bytes
Content-Type: application/json; charset=UTF-8
Date: 2013 Jan 1 10:32:06 -865ms
Server: Apache-Coyote/1.1

```
{
  "success": true,
  "list": [
    {
      "id": "50e2ab23ef86a9be51133aa5",
      "idLlista": "1234",
      "textcat": "paraula_1",
      "textjap": "製品を実際に",
      "pronjap": "lelele"
    },
    {
      "id": "50e2ad11ef86a9be51133aa6",
      "idLlista": "12345",
      "textcat": "paraula_2",
      "textjap": "製品",
      "pronjap": "pr_2"
    }
  ]
}
```

length 260 bytes

download

Il·lustració 46. Resposta a petició de llistat de paraules

Podem veure que la petició ha tingut èxit (*success* : *true*), i la llista de paraules, tot en format JSON.

Si no hi hagués paraules al servidor, hauríem vist aquesta resposta:

HTTP GET localhost:8080/pfc2012/get_words

SCHEME HOST [":" PORT] [ABS_PATH ["?" QUERY]] METHOD

QUERY PARAMETERS

HEADERS

BODY

Not available, only POST, PUT, PATCH method can hold a content.

RESPONSE

200 OK

elapsed time 14ms

HEADERS

BODY

Content-Length: 62 bytes
Content-Type: application/json; charset=UTF-8
Date: 2013 Jan 1 10:34:37 -664ms
Server: Apache-Coyote/1.1

```
{
  "success": false,
  "errorMessage": "No s'ha trobat cap paraula."
}
```

length 62 bytes

download

Il·lustració 47. Petició de llista de paraules sense resultats

6.2.13.5 TEST DE PETICIÓ D'ACCIÓ INCORRECTA

Si es fa una petició que no està prevista o que és incorrecta (paràmetres obligatoris no introduïts), els filtres no deixen passar la petició i el Servlet Controlador mai arriba a realitzar cap operació.

Ho veiem en la següent pantalla, on intentem crear una paraula sense els paràmetres correctes:

The screenshot shows a web browser's developer tools interface. The top section displays the HTTP request details: Scheme (HTTP), Host (localhost:8080), Path (pfc2012/crear_concepte_paraula), and Query Parameters (param1=1234). The Method is GET. Below this, the Headers section shows the response status as 200 OK. The Headers list includes Content-Type: application/json; charset=UTF-8, Date: 2013 Jan 1 10:37:05 -20ms, Server: Apache-Coyote/1.1, and Transfer-Encoding: chunked. The Body section shows the response body as -1. The elapsed time for the request is 18ms.

Il·lustració 48. Petició d'acció incorrecta

Veiem que la resposta és **-1**, perquè s'intenta fer una petició que no està correcta, perquè hi falten paràmetres obligatoris.

6.2.14 LLICÈNCIA DEL MÒDUL DEL SERVIDOR

El codi font del mòdul del servidor es distribueix sota la llicència Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0.html>).

Un fitxer amb la còpia d'aquesta llicència (LICENSE.txt) es distribueix amb el codi font del mòdul del servidor.

7 ANNEX – RECURSOS ADJUNTS

Juntament amb aquest document de memòria, s'hi troben els següents recursos adjunts:

- Codi font: inclou les llibreries utilitzades (fitxers .JAR) i fitxers de configuració. Nom del fitxer: **uocpfc2012_eduard_capell_salvador_lorca_server_source.zip**.
- Fitxers binaris: aplicació web, preparada per ser desplegada en un servidor *J2EE compliant* (recomanem Apache Tomcat versió 7.0.x). Nom del fitxer: **pfc2012.war**. Si es canvia el nom del fitxer, l'aplicació que es desplegui adoptarà el nom que es triï.
- Repositori GitHub: repositori disponible a l'adreça web següent: https://github.com/salvinha/UOCPFC_Eduard_Salva. Conté tot el codi i documents de treball que s'han anat generant durant la realització del projecte.
- Vídeos explicatius:
 - Vídeo del contingut del client. Disponible a les següents adreces de Youtube:
 - Part 1: <http://youtu.be/MTiOWLXKZF8>
 - Part 2: <http://youtu.be/DLv2iqucer8>
 - Part 3: <http://youtu.be/n2P3mK1ZBmg>
 - Vídeo demo de les peticions i respostes del servidor. Disponible a la següent adreça de Youtube: http://youtu.be/O2AB_XvVMsl

- Vídeo amb la presentació oral del mòdul del servidor. Disponible a la següent adreça de Youtube: <http://youtu.be/VUEoT-v7f2I>
- Vídeo amb la demostració de l'aplicació en un navegador Google Chrome a un PC. Disponible a les següents adreces de Youtube:
 - Part 1: <http://youtu.be/tZU4ZShouSI>
 - Part 2: <http://youtu.be/OGvLm6BUyNw>
- Presentació, disponible a Google Drive, permet de veure les diapositives del segon vídeo del punt anterior. Enllaç a la presentació: <http://goo.gl/vrj8X>