Universitat Autònoma de Barcelona

escola d'enginyeria

5966: Cerca i reconeixement de paraules en imatges

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Xavier Rodríguez Sabater
i dirigit per
Ernest Valveny Llobet
Bellaterra, 18 de Juny de 2015

El signant, Ernest Valveny Llobet, professor del Departament de Ciències de la Computació de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha estat realitzada sota la seva direcció per Xavier Rodríguez Sabater

Bellaterra, 18 de Juny de 2015

Signat: Ernest Valveny Llobet

Índex

1 Introducció

1.1 Objectius del projecte

L'objectiu principal d'aquest projecte és el disseny i l'elaboració d'un cercador de text en imatges mitjançant una aplicació web. L'aplicació rebrà el text a buscar per part de l'usuari i mostrarà imatges que continguin aquest text per ordre de rellevància.

Amb el desenvolupament d'aquesta aplicació es pretén posar en pràctica els mètodes de detecció i reconeixement de text en imatges descrits a [4] i [5]. Aquests mètodes, tot i ser ràpids, comparats amb altres mètodes existents, comporten un temps de càlcul massa gran per a la immediatesa que requereix una aplicació web. És per això, que l'aplicació web haurà de fer tots els càlculs dels mètodes que siguin possibles amb anterioritat. Utilitzant els càlculs desats de forma prèvia, l'aplicació web podrà oferir resultats amb la velocitat de resposta esperada a les pàgines web.

Per a resoldre els problemes que existeixen a l'hora de localitzar i reconèixer text, els mètodes de reconeixement que s'utilitzaran distingeixen imatges segons siguin

- imatges amb una part de text
- imatges de documents

Cal dir que, si les imatges contenen text manuscrit caldrà fer servir un mètode d'aprenentatge per distingir els caràcters i si és text imprès un altre diferent. Queda fora dels objectius del projecte distingir de forma automàtica el tipus d'imatge. Per tant, es tractaran dos tipus d'imatges: fotos on hi apareixen cartells amb text imprès i imatges de documents manuscrits.

També ens fixem com a objectiu que l'aplicació web inclogui una API pel cercador de text en imatges. Aquesta API ha de permetre desenvolupar aplicacions mòbils que utilitzin la funcionalitat del cercador.

1.2 Estat de l'art

En matèria de buscadors, tant els més coneguts (Google, Bing o Yahoo) com els que no ho són tant (DuckDuckGo, Baidu) ofereixen algun mecanisme per buscar imatges. En tots aquests casos, la cerca d'imatges que ofereixen, dona resultats basat en etiquetes i conceptes que assignen a les imatges i no pas al text que contenen. I no és sorprenent que sigui així. Si busquem "Torre Eiffel" és més probable que vulguem trobar una imatge del monument que una imatge d'un cartell a la carretera amb direccions de com arribar-hi.

Aquest projecte s'enfocarà d'una altra manera: buscar el text que pugui aparèixer a les imatges.

OCR (Optical Character Recognition) és el tipus de software que més s'assembla a aquest projecte. Un OCR s'utilitza per digitalitzar documents, transformant-los en text, descartant la part d'imatge. A més a més, es necessita una eina extra per buscar dins el text. Pel que fa a aquest projecte, conserva les imatges perquè sigui possible analitzar-les juntament amb el text i és capaç d'admetre imatges que no funcionarien amb un sistema d'OCR.

No sempre és més interessant guardar el text que les imatges. Una mostra és l'aplicació per mòbils OneShot [2]. Aquesta aplicació s'especialitza en capturar imatges de text al mòbil per després compartir-les. Els mòbils són la principal raó per la qual treballem amb imatges en comptes de text. Avui en dia, tothom porta una càmera a la butxaca i ens es més fàcil i còmode fer i enviar fotos que seleccionar i copiar text o escriure.

L'anàlisi de documents antics és un altre cas. Per poder conservar els documents de la millor forma possible s'ha de minimitzar el número de vegades que els manipulem. L'aplicació web permetria crear un arxiu d'imatges dels documents i buscar de forma eficient en l'arxiu, sense riscos pels documents originals.

1.3 Contingut de la memòria

La memòria d'aquest projecte es separa en 6 capítols.

El primer d'ells ha servit per introduir el projecte i les motivacions pel seu desenvolupament.

Al segon capítol es fa un estudi de viabilitat del projecte. En aquest estudi es fa una planificació temporal del desenvolupament. A més s'analitzen els diferents riscos a l'hora de desenvolupar el projecte.

Al tercer capítol es fa un anàlisi de requeriments del projecte. Es divideix en requeriments funcionals i no funcionals.

Al quart capítol s'expliquen les diferents parts del disseny de l'aplicació. S'inclou el procés d'integració dels mètodes de detecció i reconeixement de text. Es fa un disseny per la base de dades. I es veu l'estructura i disseny general de l'aplicació.

El cinquè capítol conté el resultat de la implementació de l'aplicació web. Es mostra l'aplicació i s'explica el funcionament d'aquesta.

Finalment, a l'últim capítol tenim les conclusions. S'analitzen els objectius complerts i es posen en contra posició amb els objectius i la planificació prèvia. També s'inclouen possibles formes de continuar amb el projecte.

2 Estudi de la viabilitat del projecte

El projecte es farà amb el llenguatge de programació Haskell. Aquest llenguatge de programació permet: treballar amb un nivell d'abstracció molt alt, un rendiment proper al de C, llibreries i eines prou madures i un sistema anomenat Foreign Function Interface (FFI) que permet integrar codi d'altres llenguatges.

Cal dir que el codi de cerca i reconeixement de text en imatges ja ve proporcionat. Concretament, hi ha dos algorismes: un que funciona amb imatges manuscrites i un que funciona amb imatges amb text imprès. Tots dos algorismes depenen de la llibreria OpenCV (Open Source Computer Vision) que té llicència BSD i per tant es pot utilitzar de forma gratuïta per a usos acadèmics i comercials. El codi proporcionat està implementat amb `C++` i s'utilitzarà la FFI per fer de pont amb Haskell.

També vénen proporcionades un conjunt d'imatges on buscar text. N'hi ha de dos tipus: imatges amb text manuscrit i imatges amb text imprès.

Cal accés a un ordinador per ubicar-hi el servidor i una base de dades on guardar les imatges i altres dades. Aquest ordinador el proporcionarà el tutor del projecte i estarà ubicat al Centre de Visió per Computador.

2.1 Riscos

El projecte involucra moltes eines de software: compiladors, llenguatges de programació i llibreries. En la majoria de casos s'ha treballat poc o gens amb aquestes eines. Per tant caldrà un aprenentatge sobre la marxa de cadascuna de les eines utilitzades per fer el projecte. En particular, l'experiència a l'hora de desenvolupar aplicacions web és escassa.

Al realitzar l'aplicació ens podem trobar problemes amb el rendiment dels mètodes de reconeixement i detecció de text. Els algorismes de visió són en general costosos i en canvi la web és una plataforma gairebé instantània. S'haurà de fer un bon treball en el disseny i l'optimització de l'aplicació per poder adaptar els dos mons. Tot i així, al final podríem trobar que factors inherents als algorismes impedissin una resposta ràpida per part de la web.

2.2 Planificació temporal del treball

La planificació temporal està pensada per a realitzar el gran gruix del treball durant el segon semestre del curs 2014-2015.

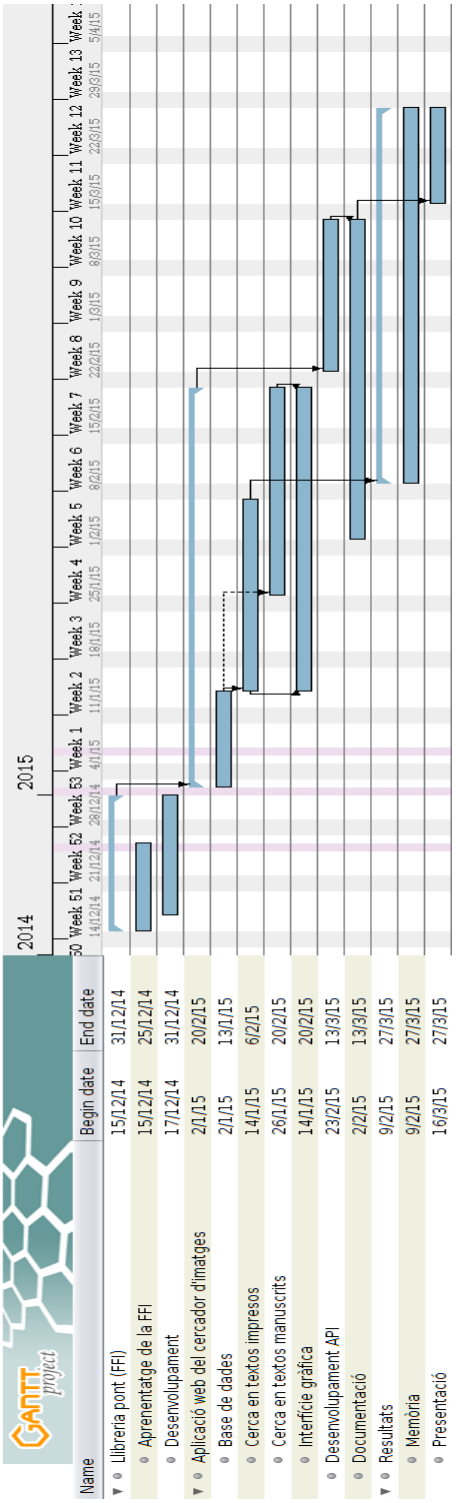
Podem dividir les tasques a realitzar de la forma següent:

- Desenvolupament de la llibreria pont entre el codi `C++` i Haskell mitjançant la FFI.
  - Cerca de documentació i aprenentatge de la FFI (8 hores)
  - Elaboració de la llibreria (16 hores)
- Aplicació web del cercador d'imatges
  - Disseny i elaboració de la base de dades (22 hores)
  - Implementació de la cerca en textos amb text imprès (82 hores)
  - Implementació de la cerca en textos amb text manuscrit (82 hores)
  - Disseny i refinament de la interfície gràfica (60 hores)
- API d'accés al servidor (37 hores)
- Documentació de les diferents parts del projecte (24 hores)
- Per últim s'ha de fer la memòria del projecte i preparar la presentació per exposar-ho:
  - Memòria (32)
  - Presentació (8 hores)

Total: 375 hores

A continuació podem veure la planificació amb el diagrama de Gantt

2 Estudi de la viabilitat del projecte



3 Anàlisi de requeriments

El propòsit d'aquesta secció és recollir, analitzar i definir les necessitats d'alt nivell i les característiques del cercador web.

3.1 Requeriments funcionals

3.1.1 Cercador

L'usuari introduirà un text al buscador i aquest mostrarà les imatges que continguin el text per ordre de rellevància.

El resultat d'una cerca de text ha de permetre mostrar a les imatges el requadre on es troba el text que s'ha buscat.

A més, haurem de poder seleccionar una imatge per poder veure-la de forma individual.

En un moment donat, les imatges amb les que treballarà el buscador només podran ser de un dels dos tipus:

- Imatges de documents manuscrits
- Fotografies on hi apareixen textos impresos.

L'aplicació web ha de poder buscar en les imatges de tipus manuscrit.

L'aplicació web ha de poder buscar en les imatges amb text imprès.

3.1.2 Administració

L'administració de l'aplicació web s'ha de poder realitzar directament a la pròpia web.

Per poder realitzar la part d'administració caldrà identificar-se amb contrasenya.

Des de l'administració s'han de poder crear conjunts d'imatges amb els que treballarà el cercador.

L'administrador ha de poder seleccionar amb quins d'aquests conjunts d'imatges treballarà el cercador.

Un conjunt d'imatges s'ha de poder editar, modificant el nom que el representa o afegint-hi imatges.

A l'hora d'afegir imatges, s'ha de poder afegir un grup d'imatges mitjançant un zip.

Els conjunts d'imatges s'han de poder esborrar.

L'administrador ha de poder forçar que es facin els càlculs previs de qualsevol conjunt d'imatges.

Els càlculs també s'han de poder programar per realitzar-los en una hora concreta.

3.1.3 API

L'API ha de permetre fer la cerca de text en imatges al servidor com si fos l'usuari i obtenir les dades per reproduir els mateixos resultats.

3.2 Requeriments no funcionals

Temps de resposta

L'aplicació web ha de reaccionar amb els temps de resposta esperats a la web. No cal establir un valor màxim concret ja que tots utilitzem la web i sabem de forma intuïtiva quan triga massa a respondre. Intuïtivament podem dir que les respostes haurien de ser com a molt de l'ordre de segons. A partir de la desena de segons ja seria massa.

Comput de fons

Quan l'usuari fa una consulta a l'aplicació web s'han d'haver realitzat el màxim de càlculs possibles amb anterioritat. Els càlculs de detecció i reconeixement d'imatges s'hauran de realitzar en moments en que no dificultin el correcte funcionament de l'aplicació web.

Adaptat a mòbil

La web també s'ha de poder utilitzar de forma correcta en navegadors moderns per a mòbil.

4 Disseny

4.1 Introducció

Haskell és un llenguatge de programació:

- de propòsit general
- estandarditzat [6]
- amb avaluació mandrosa
- funcional
- amb tipat fort

Per què utilitzar Haskell? En el món de la indústria del software sembla que gairebé tothom utilitza un dels dos estils de llenguatge de programació següents:

- Llenguatges de tipatge fort com Java, `C#`, `C++`. Aquests llenguatges són ràpids i donen garanties en temps de compilació, però es fan farragosos d'utilitzar.
- Llenguatges de tipatge dinàmic com Ruby, Python, PHP. Aquests llenguatges incrementen la productivitat (com a mínim a mig termini) i en canvi són lents i ofereixen poques garanties de funcionament correcte.

Haskell ens permet agafar la millor part dels dos mons. Té un tipatge fort que és conegut per oferir moltes garanties de correctesa. De fet, Haskell captura molts més errors en temps de compilació que els llenguatges de tipatge fort mencionats. Haskell és suficientment eficient i, tot i no arribar a la mateixa velocitat d'execució que els llenguatges de tipatge fort, està al mateix nivell de magnitud que aquests. Finalment, Haskell és un llenguatge molt expressiu que permet un alt nivell d'abstracció.

A part dels mèrits propis del llenguatge de programació, calen més elements a l'hora d'escollir-lo per desenvolupar un projecte. En particular, per

desenvolupar una aplicació web, cal que el llenguatge incorpori llibreries i eines per a desenvolupament web i que aquestes siguin prou madures. També és interessant tenir coneixements i experiència prèvia amb el llenguatge.

Cal fer menció especial a `C++`, ja que el codi amb mètodes de detecció d'imatges que s'utilitza al projecte està implementat amb aquest llenguatge. S'ha triat Haskell i no `C++` pels pocs coneixement de què es disposen i perquè, tot i oferir algunes llibreries pel desenvolupament web, no és un llenguatge molt adequat per aquesta funció.

4.2 Mètodes de detecció i reconeixement de text

Per detectar i reconèixer el text a les imatges calen els següents passos:

- El primer pas consisteix en detectar les seccions significatives on hi puguem trobar text. Amb el mètode utilitzat obtenim rectangles dins de la imatge on es pensa que hi apareix text escrit. A vegades aquests rectangles poden donar seccions molt solapades o paraules trencades.

- Un cop tenim retallada d'una imatge la part on sabem que hi ha text s'han de fer una sèrie de transformacions. Basat en la probabilitat de que apareguin les diferents lletres o uns certs bigrames es construeix un vector de dimensió 604. Aquest vector no és directament comparable amb el que obtindrem al processar el text que vulguem cercar. Per poder comparar-los i per reduir la dimensió de l'espai es fa una projecció a un altre espai i s'obté un vector de dimensió 80. En els passos a realitzar hi ha involucrats uns models d'aprenentatge prèviament calculats. El fitxers amb els resultats de l'aprenentatge realitzat que utilitzem serveixen per l'anglès.

- Per poder comparar un text amb la imatge en comptes de treballar amb probabilitat es conten directament les lletres i bigrames que apareixen. Amb aquests càlculs obtenim un vector similar al de la imatge i també de dimensió 604. Aprofitant el mateix model d'aprenentatge que a l'apartat anterior es fa una projecció equivalent del vector al mateix espai de dimensió 80.

Per poder comparar el text amb una imatge es calcula una distància entre els dos vectors. Al final per l'aplicació el que volem és comparar un conjunt de vectors que corresponen a imatges amb un sol vector que correspon al text. Així el que es fa és construir una matriu amb tots els vectors de les imatges i fer un producte de matrius amb el vector del text. El resultat és un vector on cada posició conté un valor entre -1 i 1. Aquest valor ens dona una escala per veure si s'assemblen molt o poc la imatge i el text, on -1 seria el pitjor cas i 1 el millor.

Per a integrar a l'aplicació web el codi en `C++` de detecció i reconeixement d'imatges que tenim generem els 3 executables següents:

- `textDetection image foutput`

  on `image` és el fitxer de la imatge amb la que volem treballar i `foutput` és el fitxer on obtindrem el resultat.

  La primera línia del fitxer resultant conté el número de rectangles on s'hi podria trobar text. A continuació tenim tots els rectangles, cadascun en 4 línies:

  1. Coordenada `x` de la cantonada superior esquerra del rectangle.
  2. Coordenada `y` de la cantonada superior esquerra del rectangle.
  3. Amplada del rectangle.
  4. Alçada del rectangle.

- `textVectorization` image frects foutput!

  on `image` és el fitxer de la imatge amb la que volem treballar, `frects` és el fitxer on podem trobar els rectangles de la imatge (amb el mateix format que amb `textDetection`) i `foutput` és el fitxer on obtindrem el resultat.

  La primera línia del fitxer resultant conté el número de rectangles. A continuació tenim a cada línia el vector resultant de transformar a representació vectorial la subimatge corresponent a cada rectangle. Els vectors estan en el mateix ordre que el fitxer amb els rectangles.

- `stringVectorization` words foutput!

  on `words` és la cadena de caràcters que volem transformar i `foutput` és el fitxer on trobem a la primera linia el vector resultant.

4.3 Disseny de la base de dades

A la base de dades cal guardar tota la informació necessària perquè en cas de reiniciar el servidor es puguin recuperar les dades afegides a l'aplicació web. També és important que tot el comput que representen els mètodes de la secció 4.2 no s'hagi de realitzar més d'una vegada.

Anomenarem els conjunts d'imatges amb que treballi l'aplicació *image collections* o *collection* per abreujar. Una *collection* ha de tenir un nom per identificar-la, un tipus d'imatges associat (o bé documents manuscrits o fotos amb text imprès) i hem de saber si s'està utilitzant de forma activa pel buscador o està desactivada.

Les imatges han d'estar associades a una *collection* concreta. De les imatges cal saber si ja han estat processades pels mètodes de 4.2 per poder-les incloure en resultats de cerca. A l'hora de desar a la base de dades les imatges tenim dos opcions. Es pot desar una imatge dins de la pròpia base de dades com el que s'anomena *blob* (Binary Large OBject) o emmagatzemar el fitxer i guardar el *path* a la base de dades. L'avantatge d'escollir l'opció del *blob* és que a l'hora de fer *backup* de la base de dades no cal buscar els fitxers a part i tampoc cal gestionar on guardar els fitxers de forma manual. El problema dels *blobs* és que hi ha *backends* que no els suporten o si ho fan és de forma no molt eficient. En el nostre cas guardarem el *path* de la imatge.

Finalment, un cop han estat processades les imatges obtenim unes seccions rectangulars de les imatges on hi podem trobar text. A més, aquests rectangles tenen associat un vector numèric que permet després comparar-los per fer les cerques.

4.3.1 Model entitat-relació

Amb la descripció de la base de dades anterior podem construir un model entitat-relació.

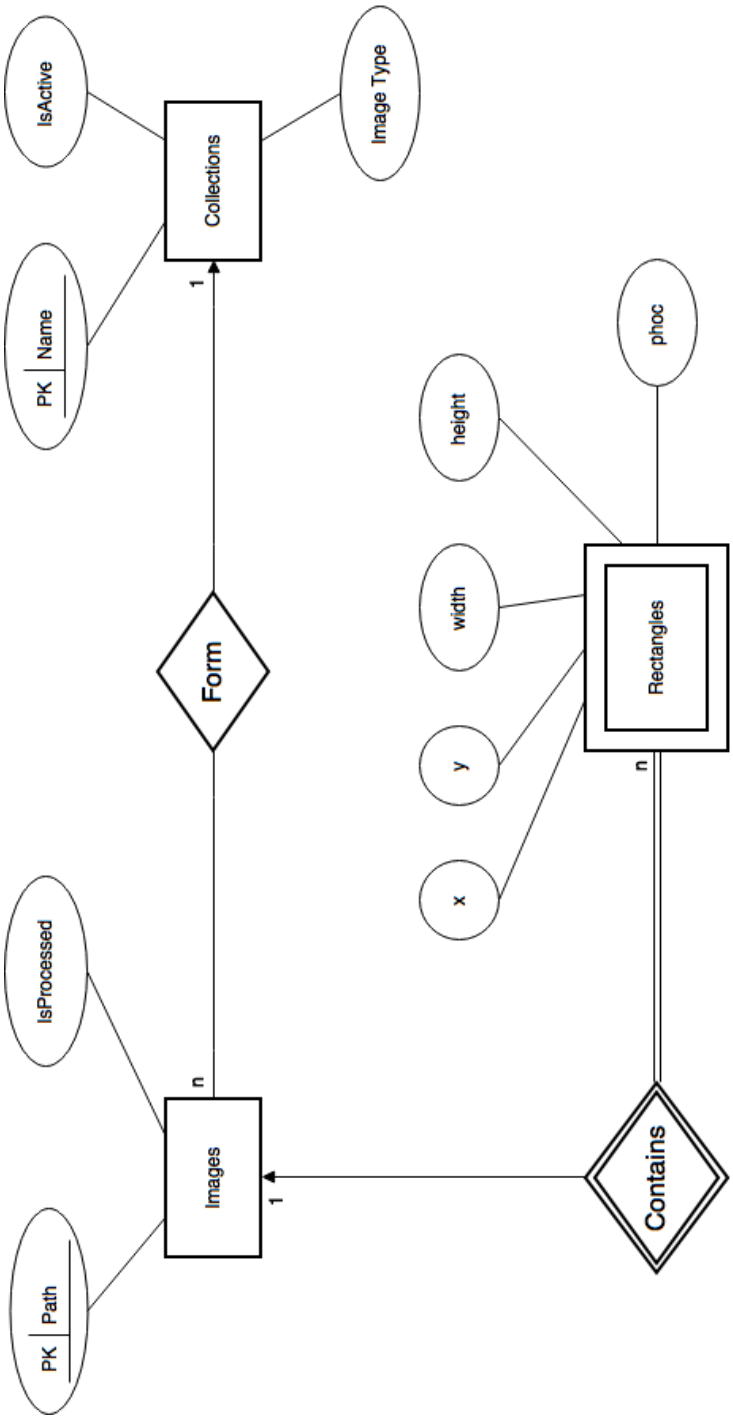
Entitats

- Collections(Name, isActive, ImageType)
- Images(Path, isProcessed)
- Rectangles(x, y, width, height, phoc)

Interrelacions

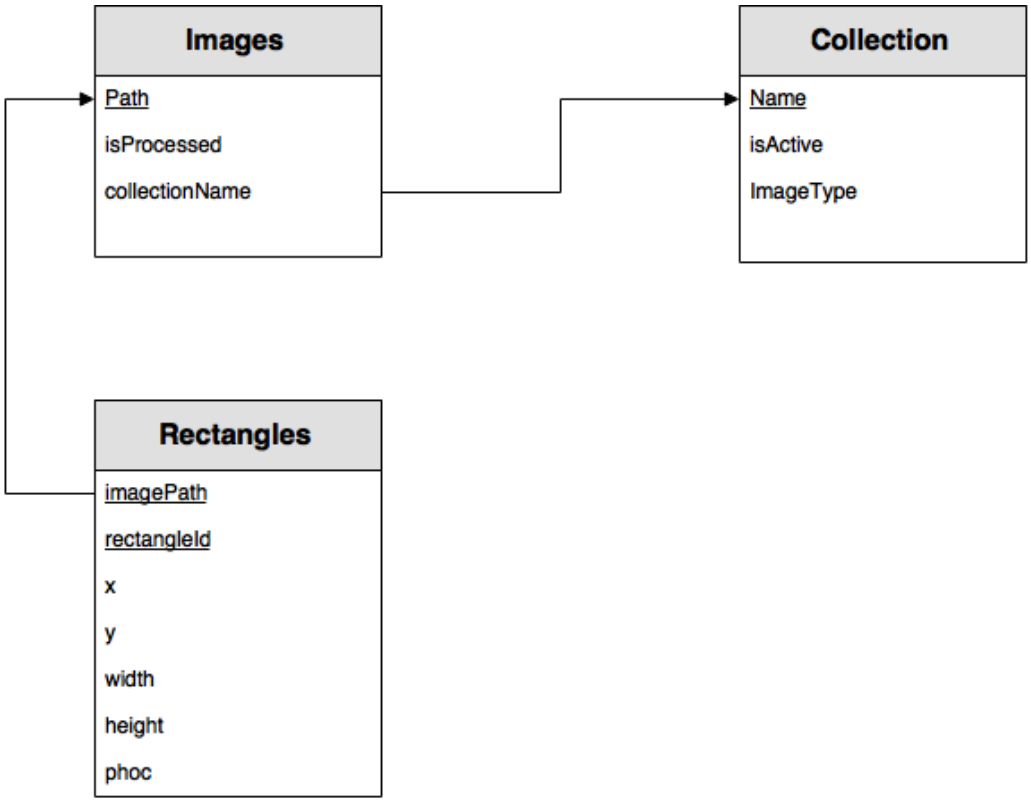
- Forms: Interrelació $n - 1$ entre Images i Collections.
  Expressa que una imatge és dins d'una única col·lecció d'imatges i que una col·lecció conté n imatges.
- Contains: Interrelació $1 - n$ entre Images i Rectangles.
  Expressa que un rectangle és dins d'una única imatge i que una imatge pot tenir n rectangles.

Diagrama E-R


IsActive
Collections
Image Type
PK
Name
1
phoc
height
width
y
x
Rectangles
Form
n
IsProcessed
Images
Contains
1
PK
Path

4.3.2 Taules

A partir del diagrama relacional, obtenim les taules per la nostra base de dades.



La única clau candidata de l'entitat Rectangles consisteix en agafar tots els seus atributs. Com veure'm a la secció 4.4.3, s'introdueix com a requisit tenir un identificador únic a cada taula. El camp *rectangleId* fa referència a aquest identificador únic i és el que utilitzarem en lloc d'agafar tots els atributs de l'entitat Rectangles.

4.3.3 Backend

Pel *backend* s'utilitza la base de dades relacional SQLite. Aquesta base de dades és una de les més utilitzades gràcies a les seves característiques:

- Requereix molt poques dependències per utilitzar-la i es compila com a llibreria estàtica.

- La majoria de bases de dades estan implementades com a servidor extern a l'aplicació, mentre que SQLite va integrat dins el procés de l'aplicació. Això té l'avantatge de no requerir instal·lacions ni configuracions.

- El contingut de la base de dades consisteix en un sol fitxer. Fer un *backup* de la base de dades és tan fàcil com fer *backup* del fitxer.

Tot això fa que SQLite sigui un excel·lent punt de partida com a *backend* per la base de dades.

No obstant, SQLite també té desavantatges. Que la base de dades sigui tan simple d'integrar i d'utilitzar implica que tampoc pots configurar-la i afinar-la per treure'n més rendiment. SQLite tampoc dona suport a molts tipus de dades per emmagatzemar ni instruccions SQL.

A la secció 4.4.3 veure'm com aquests desavantatges no suposen gaires problemes.

4.4 Arquitectura de l'aplicació

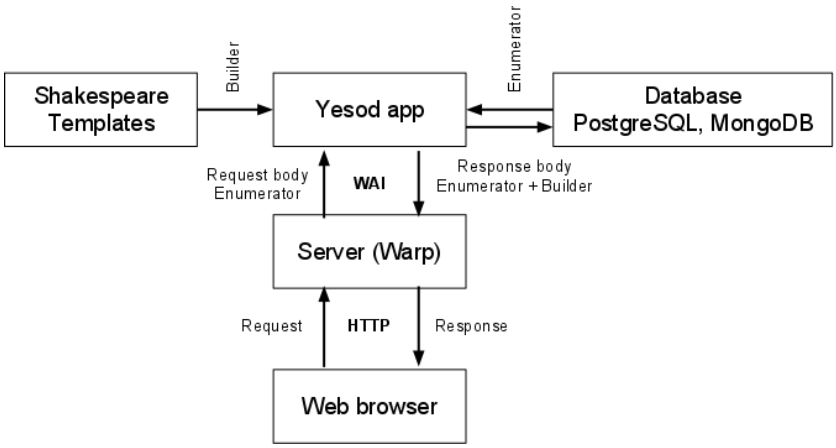
L'arquitectura de l'aplicació segueix en general un model MVC (Model-View-Controller). La part de la vista de l'aplicació correspon al HTML, CSS i javascript típic d'una web i que forma la interfície gràfica. La part del model consisteix en quines dades s'utilitzen per representar la informació programa. El controlador s'encarrega de la lògica per transformar les dades i de com passar-les a la vista per actualitzar la seva representació. Cada capa del model ve recolzada per una llibreria de Haskell que ens permet implementar-la.

La llibreria Shakespeare ofereix un sistema de *templates* per generar codi HTML, CSS i javascript i correspon a la vista. La llibreria Persistent defineix el model de les dades i ajuda a definir les regles per transferir-les al *backend* de la base de dades. El controlador correspon a la llibreria Yesod que s'encarrega de proporcionar la majoria d'eines per la lògica de l'aplicació, dirigir cada petició web al codi que li correspon i integrar tots els altres sistemes.

Entre tots aquests components no n'hi ha cap que entengui el protocol HTTP per poder rebre i contestar peticions al servidor web. Aquí entra en acció una altra llibreria, WARP, que només s'utilitza per interpretar el protocol HTTP i queda al marge del funcionament concret de l'aplicació web. Aquesta llibreria es comunica amb el controlador utilitzant una interfície estàndard anomenada Web Applicaction Interface (WAI).

D'aquesta manera, s'aconsegueix una separació de responsabilitats entre les diferents parts de l'aplicació permetent construir una aplicació més modular i desacoblada.

Podem veure l'estructura de general de l'aplicació web en el següent esquema:



A continuació veure'm amb una mica més de detall com s'utilitzen cadascun d'aquests components.

4.4.1 Shakespeare

La vista de tota aplicació web consisteix en una mescla de HTML (estructura), CSS (estil) i javascript (lògica). Shakespeare treballa amb tres llenguatges: Hamlet, Cassius i Julius que generen codi HTML, CSS i javascript, respectivament. Aquests nous llenguatges difereixen una mica sintàcticament i semànticament respecte dels originals (HTML, CSS i javascript) en què afegeixen alguna funcionalitat extra.

La funcionalitat més important que ofereixen és la d'interpolar valors de Haskell, semblant a com es pot fer amb PHP, per exemple. La principal diferència amb altres sistemes de *template* que permeten interpolació de valors és que amb Shakespeare hi ha més garanties estàtiques sobre què es pot interpolar i què no. Per començar, no es pot executar codi que causi efectes secundaris dins del *template*. Només es poden interpolar valors purs que es puguin transformar en HTML o valors de *type-safe URLs* que veurem a la secció 4.4.2. És important mencionar que no és el mateix un valor HTML que un *string*. En el cas que permetés interpolar valors de *string* un usuari podria posar-se un nom com "`myName<script>src=maliciousScript.js</script>`"

i seria un problema de seguretat. Per això, per interpolar *strings* es passen primer a HTML *escapant* tots els caràcters que calgui.

Dels tres llenguatges, Hamlet és el més complex i el que difereix més del seu homòleg. A més d'interpolar valors i les diferències sintàctiques, Hamlet incorpora algunes estructures de control molt pràctiques per poder tractar amb els valors interpolats i el HTML generat. Aquesta complexitat té un cost i és que canvis en un fitxer de Hamlet comporten haver de recompilar el codi. En canvi, els altres llenguatges (Cassius i Julius) permeten fer canvis i recarregar-los sense haver de compilar de nou. Això només té importància a l'hora de desenvolupar el projecte. En l'executable final, es compilen tots els fitxers amb els següents avantatges:

- Tot el codi HTML, CSS i javascript forma part de l'executable final. Això comporta un increment del rendiment ja que l'aplicació no ha fer operacions de *input/output* amb fitxers.
- Tot el codi javascript es compila amb un *minifier*[1] automàticament.

4.4.2 Yesod

Routing La funcionalitat central que proporciona Yesod és el *routing*. Ho fa amb una sintaxi declarativa i amb un *routing* de tipatge segur. Ho podem veure amb un exemple tret de l'aplicació.

```
/img/#ImageId ImageR GET
```

Amb aquesta declaració diem que a la ruta `/img/#ImageId` hi tenim un *resource* `ImageR` que només accepta peticions `GET`. A la URL, la secció `#ImageId` indica que aquesta part de la ruta ha de ser una representació del tipus de dades que tenen els identificadors únics de les imatges `ImageId`. Qualsevol ruta pot definir tots els mètodes HTTP que vulgui, en l'exemple només `GET`. Igualment les rutes poden ser simples, contenir tipus de dades o combinar les dos coses de forma mixta.

Es podria dir que la característica més significativa de Yesod és *type-safe URLs*. En comptes d'haver de enganxar diferents*strings* per generar URLs, Yesod assigna a cada ruta un únic valor de Haskell. En el cas de l'exemple es tracta de `ImageR`. Utilitzant aquests valors no és possible construir una URL incorrecta. Tot i així, encara és possible generar URLs que donin un error 404 *Not Found*, com a l'exemple, on podríem demanar una imatge que no existís.

Handlers La major part del codi de l'aplicació web resideix als *Handlers*. Seguint l'exemple d'abans, caldria definir una funció (el *handler*) anomenada `getImageR` que rebés per paràmetre el valor de tipus `ImageId` i generés la resposta a la petició HTTP `GET`. El funcionament és simple, per cada mètode HTTP que accepti un *resource* `ResourceR` cal definir un *handler* `methodResourceR`.

Dins de les funcions dels *handlers* podem accedir a tota la informació necessària per construir una resposta (accedir a la base de dades, informació de sessió, cookies, etc). En concret, podem accedir al valor de l'anomenat *foundation type*, on s'emmagatzema la informació de l'aplicació. Tota aplicació Yesod ha de definir el seu propi *foundation type*. Entre d'altres, hi tenim les dades per connectar amb la base de dades, la funció de *logging* i els

diferents valors de configuració de l'aplicació. Una característica important del *foundation type* és què es manté en memòria per treballar amb totes les peticions al servidor web. Per això, és aquí on guardem la matriu amb els vectors que corresponen als rectangles on hi ha text de les imatges.

A l'hora de rebre una cerca de text, el *handler* corresponent no ha de fer cap petició a la base de dades. Dins el *handler*, es transforma la cadena a cercar a la representació vectorial i es multiplica amb la matriu que tenim al *foundation type* per obtenir els resultats.

Widgets Un dels problemes a l'hora de desenvolupar una aplicació web és fer components reutilitzables. Per exemple, per fer una barra de navegació cal un component de HTML i després la part de CSS i javascript. Una opció és posar el CSS i el javascript en un fitxer global i inserir-lo a cada pàgina web. En el millor dels casos, afegeixes declaracions en llocs on no els necessites i en el pitjor, un component de javascript pot fer funcionar malament una pàgina. Yesod ens ofereix una solució molt pràctica: els *widgets*. Un widget veu un component com un conjunt de HTML, CSS i javascript i ens permet treballar-hi de forma composicional. Podem declarar *widgets* i combinar-los entre ells per produir pàgines web més grans. Al final, el codi genera un HTML i s'encarrega d'agrupar el codi CSS i javascript i inserir-lo on toqui.

En la majoria dels casos, els *handlers* de l'aplicació retornen una pàgina web produïda amb la combinació d'un o més *widgets*.

4.4.3 Persistent

La llibreria Persistent s'encarrega de la frontera entre l'aplicació i la capa física. Una de les característiques més importants que la defineixen és que està dissenyada per ser independent de la base de dades utilitzada. També s'encarrega de mantenir la seguretat que ens proporciona Haskell amb el seu fort tipatge a l'hora de transformar valors de tipus de dades de Haskell a valors de la base de dades. Així, permet utilitzar una interfície per fer consultes de forma productiva i amb garanties estàtiques.

Com a resultat de ser independent de la base de dades utilitzada, ha de donar suport a bases de dades relacionals i no relacionals. Això introdueix

unes certes restriccions. Al utilitzar Persistent no es poden fer *joins* i s'introdueix a cada taula un camp d'identificador únic a més a més de la clau primària que hi hagi definida.

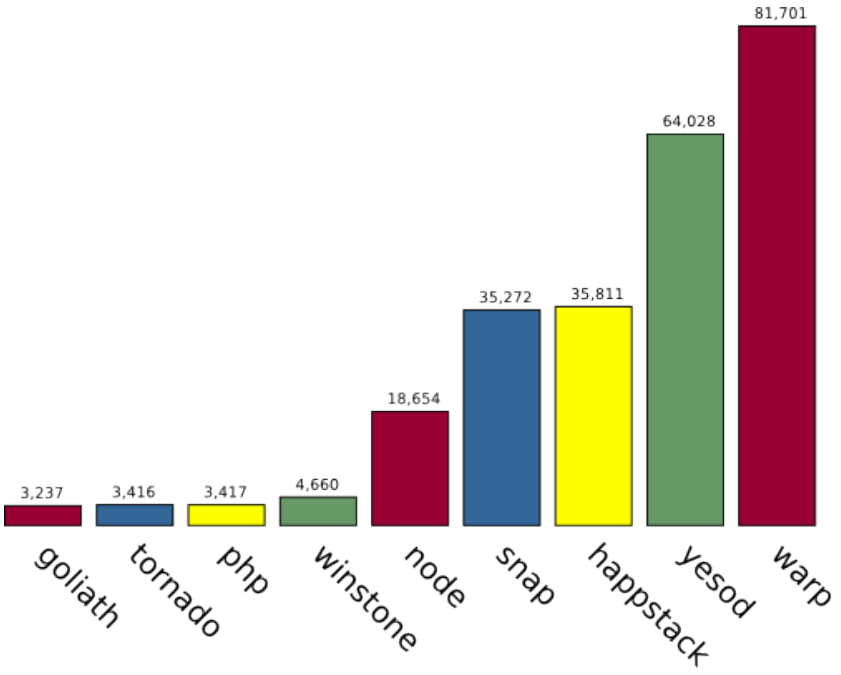
Per resoldre el problema dels *joins* existeixen 3 solucions. La primera solució és fer els *joins* a nivell d'aplicació de forma manual. La segona és utilitzar una altra llibreria per sobre de Persistent, anomenada *Esqueleto*, que permet realitzar *joins* en casos concrets i que afegeix una altra capa de complexitat i dependències. Per últim, Persistent també proporciona eines per accedir a les bases de dades amb *raw* SQL. En aquest cas, sí que depèn del *backend* concret que s'utilitzi per la base de dades i obliga a canviar codi si canviem de *backend*. Per realitzar l'aplicació web s'ha utilitzat la primera solució, ja que no cal realitzar *joins* molt complexos.

El model que hem definit amb Persistent correspon a les taules vistes a la secció 4.3.2. Amb aquest model, Persistent genera: els tipus de dades per Haskell que s'utilitzen en tota l'aplicació, les taules que corresponen a aquests tipus i el codi que s'encarrega de transformar valors de Haskell a valors de SQL.

4.4.4 Warp

Per a la nostra aplicació web no cal interactuar amb el servidor més enllà de per escollir-ne'n un i inicialitzar-lo. En el nostre cas, s'utilitza WARP principalment perquè és la opció estàndard.

En la secció 4.1 es comentava que Haskell tenia millor rendiment que els llenguatges dinàmics. Tot i que no s'ha de prendre com una mostra molt significativa, tenim un *PONG benchmark* que data del 2011[7]. El *PONG benchmark* compta les sol·licituds per segon de diferents servidors per la resposta de 4 bytes "PONG" (més és millor).



Notes al respecte:

- Goliath és un servidor de Ruby, Tornado de Python i Winstone de Java.
- El test de Yesod utilitza Yesod a sobre de WARP.
- El cas de `Node.js` és particularment interessant perquè ha despuntat en els últims anys i el principal argument per utilitzar-lo és el rendiment que té.

5 Resultats

5.1 Detecció i reconeixement de text

A la secció 2 es va planificar integrar el codi `C++` a Haskell mitjançant la FFI. Com hem vist a la secció 4.2, s'han elaborat 3 executables per poder integrar el codi. Al resultat final no hi ha intervingut la FFI; els executables són tot codi `C++`. Per entendre aquest canvi veurem una mica els requisits que calien per seguir la planificació original. A l'hora d'integrar el codi extern mitjançant la FFI cal saber com tractar tres elements:

- Quan compilem un programa a codi màquina les funcions i procediments esdevenen etiquetes que estan associades a una posició en el codi màquina. Per cridar una funció cal posar els paràmetres a la memòria i als registres i saltar a l'etiqueta corresponent. Per poder utilitzar la funció cal saber on desar els paràmetres i d'on treure'n el resultat. Això és el que s'anomena una convenció de crida.

- Per poder transformar els tipus de dades del llenguatge extern a Haskell cal saber com estan guardats en memòria i quants bytes ocupen. Donat un tipus de dades, cal saber l'*offset*de cada camp del tipus respecte de la direcció base on es troba. Per fer-ho s'han de saber quins *paddings* s'afegeixen entre cada camp. A més, en alguns casos cal utilitzar les eines que proporciona la FFI per reservar i alliberar memòria, fet que no cal en general amb Haskell ja que té gestió de memòria automàtica.

- Per poder compilar el codi final cal tenir dos compiladors, el del codi extern i el de Haskell. I per tant, cal integrar els diferents sistemes de *build* que utilitzin les dos parts.

Tot i que l'estàndard de Haskell defineix a la FFI[6, capítols 8, 24-37] algunes convencions de crida, deixa detalls d'implementació sense especificar. Per això els compiladors de Haskell avui dia només donen suport a dos convencions: `ccall` (convenció de C) i `stdcall` (convenció de pascal). En general el codi `C++` no es pot utilitzar directament utilitzant `ccall`. A

diferència de C, `C++` té sobrecarrega de funcions. Això implica que les etiquetes que genera el codi `C++` depenen també de l'objecte i dels paràmetres, no només del mètode. Per solucionar-ho es pot treballar amb les etiquetes que genera el codi directament (treball costós i propens a errors) o exposar individualment cada funció o mètode com a `extern "C"`. Les declaracions de `extern "C"` s'utilitzen en general per poder integrar codi `C++` i C dins d'un mateix projecte.

Integrar codi C amb la FFI és relativament senzill gràcies a que hi ha moltes eines que automatitzen les parts més farragoses i propenses a errors. Existeixen eines per intentar automatitzar el procés amb `C++`, però no acaben de ser satisfactòries, entre d'altres raons, perquè no hi ha una manera clara de traduir un model orientat a objectes a un de funcional.

A l'inici del desenvolupament amb codi `C++` van aparèixer una sèrie de problemes amb dependències de llibreries i el sistema de compilació. Per això, i per la complexitat que hem vist que implica treballar amb la FFI per integrar codi `C++`, es va decidir optar pels executables de la secció 4.2. Aquest enfoc obté pitjor rendiment ja que treballa amb fitxers temporals per passar les dades, en comptes de transformar valors en memòria directament. Tot i així, aquest empitjorament del rendiment només afecta als càlculs en segon pla de les imatges i és relativament poc, en proporció al temps total de càlcul. Per un altre costat, si s'han de fer canvis al codi de detecció de text no cal recompilar tota la web, n'hi ha prou amb substituir els executables pels nous.

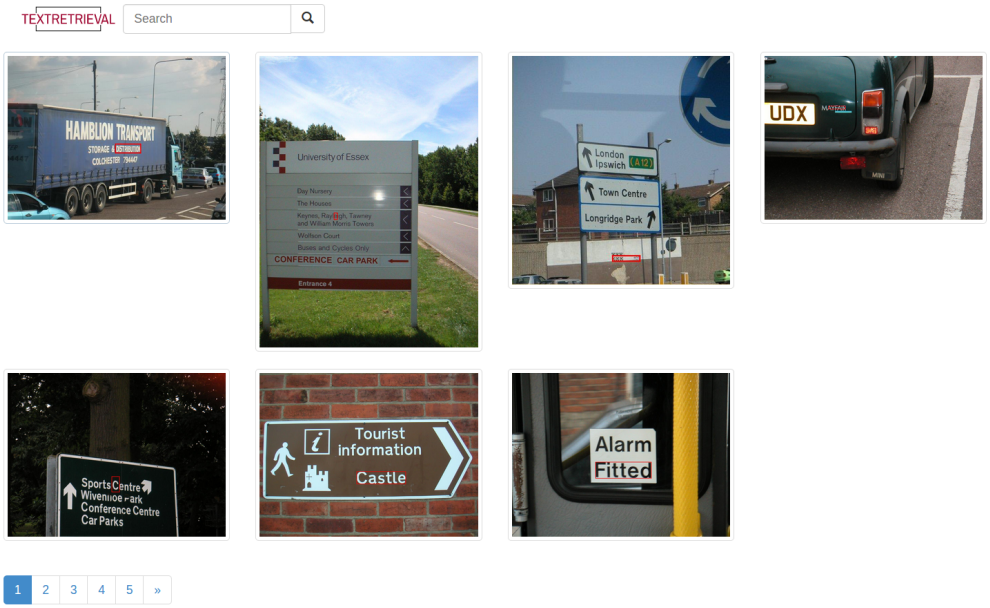
5.2 Web

5.2.1 Cercador

Home

Al entrar a la web ens trobem amb la pàgina principal. Des d'aquí podem fer les cerques a l'aplicació o identificar-nos com a administrador per poder realitzar canvis en la configuració.

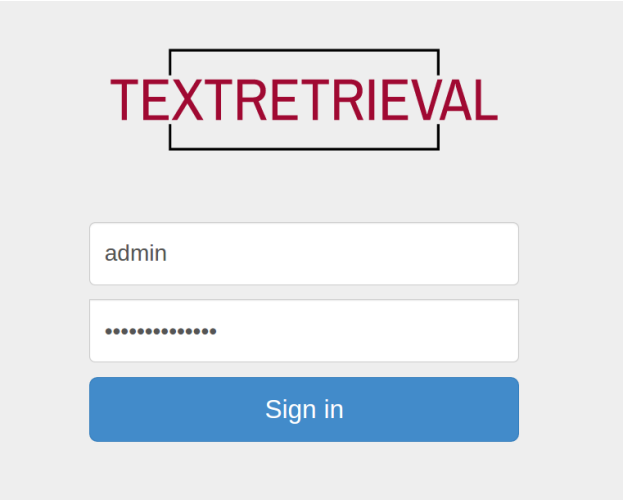
Cerca



Si fem una cerca, es mostren les imatges resultants amb els requadres significatius marcats en vermell. En aquest cas, les imatges no es mostren amb la mida original: s'ha fet un escalat de les imatges per poder veure tots els resultats a la vegada. Si fem *click* en una imatge, la podrem veure en la seva mida original. Depenent de la configuració de l'aplicació canvia una mica el triatge per saber quins resultats mostrar. Ho veurem en més detall a la secció dels *settings*.

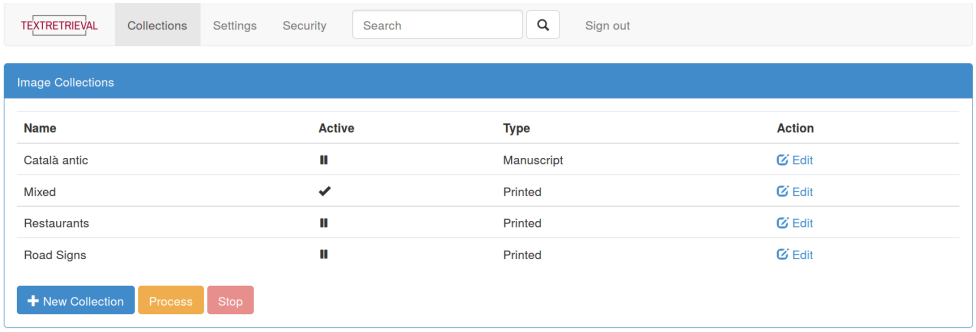
5.2.2 Administració

Sign In



El primera pas per realitzar modificacions a l'aplicació és fer el *Sign In*, introduint l'usuari i la contrasenya.

Col·leccions



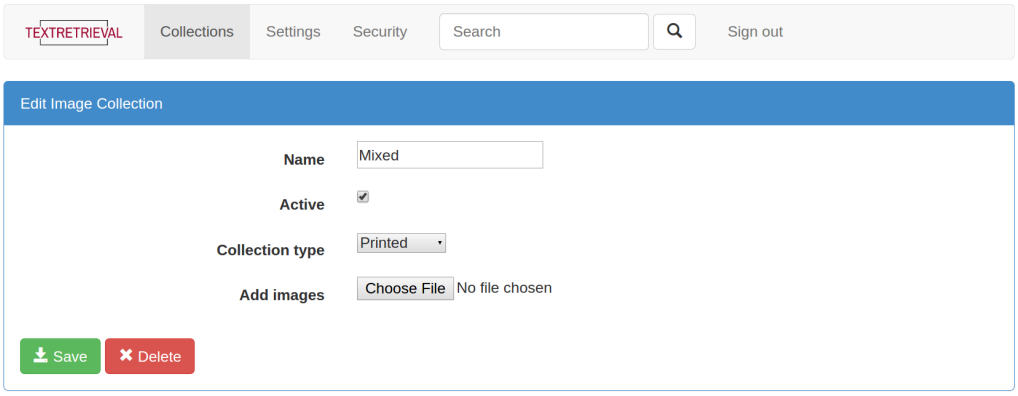
Un cop estem identificats, podem modificar les col·leccions d'imatges amb les que treballa l'aplicació.

Les col·leccions marcades com a actives són les que utilitza el cercador com a font per buscar els resultats. Només es poden utilitzar les imatges que hagin estat prèviament processades de les col·leccions actives. En cas que s'acabin d'afegir imatges aquestes encara no poden aparèixer als resultats.

A més, no es pot tenir una mescla de col·leccions actives de diferent tipus, activar una col·lecció d'un tipus diferent desactiva totes les anteriors.

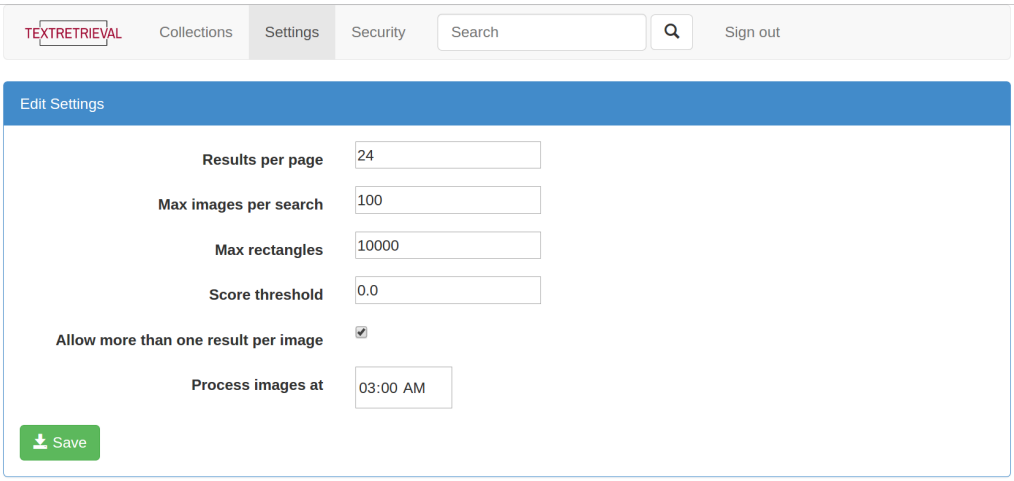
Des de la mateixa pàgina de gestió de les col·leccions podem activar el processament de les imatges o aturar-lo.

Podem crear noves col·leccions o editar les que ja tenim.



En qualsevol moment de la creació de les col·leccions, o amb una posterior edició, es poden afegir imatges de forma individual o a la vegada empaquetades amb un zip. També podem esborrar la col·lecció i per tant totes les imatges que contingui.

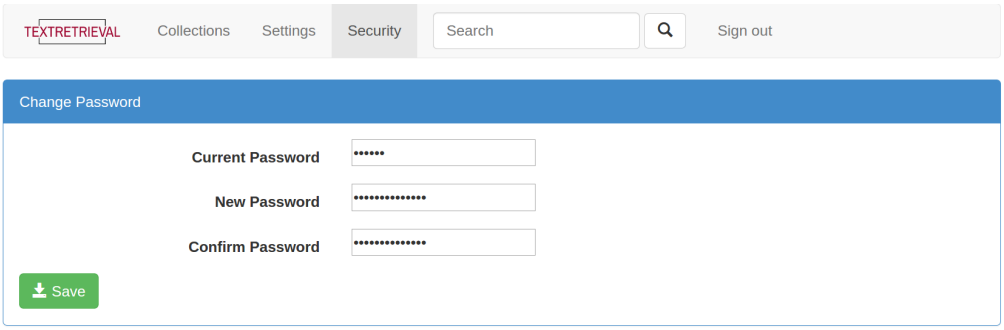
Settings



A la configuració podem editar paràmetres dels resultats de la cerca:

- *Results per page*: El número d'imatges que caben a cada pàgina.
- *Max images per search*: El màxim número d'imatges que pot retornar una cerca.
- *Max rectangles*: El màxim de rectangles amb els que pot treballar a la vegada l'aplicació. Permet limitar la memòria que consumeix l'aplicació web.
- *Score threshold*: Els resultats de les cerques estan valorats amb una puntuació de -1 a 1. En general, no hi ha una manera de saber a partir de quin valor aquesta puntuació dona bons resultats, depèn de factors com el de quines imatges s'utilitzen. El valor del *Score Threshold* determina a partir de quina puntuació l'aplicació dona els resultats com a bons.
- *Allow more than one result per image*: Com que les imatges tenen, en general, més d'una zona on s'hi detecta text, poden aparèixer de forma repetida amb diferents rectangles en una mateixa cerca. A la configuració podem escollir que només mostri el millor resultat per cada imatge o que mostri tots els resultats que pugui donar una imatge.
- *Process images at*: Aquí podem programar l'hora a la que l'aplicació s'activarà per processar les imatges.

Security



Finalment, podem canviar la contrasenya que utilitza l'administrador.

5.2.3 API

Totes les API disponibles s'accedeix utilitzant simples URLs HTTP d'estil REST[3]. En general, les API implementen la mateixa funcionalitat que el buscador web. Utilitzant la capçalera Accept de HTTP podem obtenir del client una llista, ordenada per preferència, amb els tipus que el client vol. Així podríem retornar JSON per una crida API i HTML per navegar la web. En comptes de fer-ho d'aquesta manera, les API estan ubicades en URLs a part i retornen directament el valor adequat. Hi ha dos raons per fer-ho d'aquesta manera. La més important és per facilitar canvis en el futur. Si es volgués restringir accés a la API al tenir-ho separat del buscador web no hi hauria problemes. A més, al tenir-ho separat facilitem la feina pels clients de l'API que no s'han de preocupar amb capçaleres HTTP.

Buscador

`/api/search/?q=query`

La API retorna un objecte JSON amb un array amb objectes "imatge". Cada element conté el `image_id` i un objecte `rectangle`. Cada objecte `rectangle` conté els camps `x`, `y`, `width`, `height`. A l'array les imatges resultants de buscar la cadena `query` estan ordenades de més significativa a menys.

Imatges

`/api/img/#ImageId/?x=20&y=30&width=100&height=120`

La API retorna l'arxiu de la imatge amb identificador `#ImageId`.

Paràmetres opcionals: `x`, `y`, `width`, `height`.

Si s'utilitzen els paràmetres opcionals la imatge que retorni la API tindrà pintat un rectangle vermell a les coordenades on indiquen els paràmetres.

5.3 Dependències

El resultat final de l'aplicació conté unes quantes dependències. Resumim les dependències més importants que s'inclouen al projecte:

- La llibreria dinàmica de visió per computador OpenCV, implementada en `C++` i que s'utilitza per tot el codi de detecció i reconeixement de text.
- La llibreria dinàmica de visió per computador vlFeat[8], implementada en C i que s'utilitza pel codi de reconeixement de text en imatge.
- El *framework* Yesod de Haskell. Form part del nucli principal de l'aplicació.
- La llibreria Persistent de Haskell. Habilita totes les connexions amb la base de dades.
- Com a *backend* per la base de dades ens cal SQLite. Recordem que només és necessari pel desenvolupament, la màquina on executem l'aplicació web no necessita servidor de base de dades.
- El *framework* Bootstrap (mescla de HTML, CSS i javascript). S'utilitza per tota la interfície web i permet que la web sigui *responsive*.
- La llibreria Shakespeare de Haskell, proporciona eines avançades per la generació del codi HTML, CSS i javascript.
- El compilador `g++` juntament amb les eines *cmake* i *make* per compilar el codi de `C++`.
- El compilador GHC (Glasgow Haskell Compiler) juntament amb l'eina *cabal* per compilar el codi de Haskell.
- Les imatges utilitzades a l'aplicació provenen del *street view dataset*[9]

6 Conclusions

6.1 Objectius complerts

Abans d'analitzar els objectius assolits, tractarem de l'únic objectiu incomplert del projecte què és la implementació de la cerca pels textos manuscrits. A causa de variacions en la planificació inicial, s'ha disposat de menys temps per poder realitzar aquest objectiu. Integrar els sistemes de *build* del codi `C++` al projecte van comportar molt més temps de l'esperat. Això es deu a la inexperiència amb els sistemes involucrats i les dependències que s'utilitzaven. Es tracta del clàssic problema d'esbrinar perquè si funciona en un entorn concret deixa de fer-ho quan canvia lleugerament l'entorn. En paral·lel, es van invertir recursos en aprendre el funcionament de la FFI i tractar d'integrar el codi `C++` amb aquesta eina. Finalment es va abandonar aquest camí.

Des d'un punt de vista general, la majoria de les parts planificades han portat més temps que l'esperat. La implementació de l'API ha sigut l'únic pas que ha comportat molt menys temps del que originalment estava planificat.

La resta d'objectius plantejats han estat assolits. L'aplicació fa una cerca de text dins de les imatges i dona resultats per odre de rellevància. Es poden seleccionar individualment les imatges i es marquen els requadres corresponents a la secció significativa de la imatge. L'usuari pot identificar-se com a administrador dins de l'aplicació web. Això li permet fer tots els canvis necessaris pel funcionament. Es poden crear col·leccions d'imatges, modificar-les i esborrar-les. En qualsevol moment pots afegir més imatges, ja sigui de forma individual o en grup mitjançant un zip. El sistema d'imatges per col·leccions permet a l'administrador escollir les imatges actives, amb les que treballarà el buscador, de forma modular. Tot i que no es pot cercar amb imatges manuscrites l'aplicació evita de forma automàtica equívocs i desactiva les col·leccions que tenies si n'actives una de tipus diferent. Per realitzar el comput previ de les imatges, l'aplicació proporciona dos opcions: la primera és activar de forma manual el comput en el moment que vulguem, la segona és programar l'aplicació perquè realitzi el comput necessari a una

certa hora del dia. En qualsevol cas es pot parar de forma manual abans de que finalitzin si així es desitja.

Un altre objectiu assolit ha estat la web *responsive*, és a dir, la web s'adapta a la mida de la pantalla amb la que estem veient l'aplicació. En particular, s'adapta a pantalles de mòbil i tauletes.

El resultat del disseny i la implementació de l'aplicació web ha permès obtenir uns bons temps de resposta. S'ha minimitzat el comput que cal realitzar en una cerca de forma *online* i això ha permès poder adaptar els mètodes de detecció i reconeixement de text a un entorn web.

6.2 Possibles ampliacions

L'ampliació al projecte més obvia és integrar la part per tractar les imatges de textos manuscrits. Tot i que no ha donat temps, com ja s'ha dit, a implementar aquesta part, moltes de les eines que calen per poder fer-ho ja són al projecte. Tota la lògica de l'aplicació està pensada per poder treballar amb els dos tipus d'imatges. La part més interessant a l'hora d'integrar-ho seria veure com afecta al temps de resposta d'un cerca ja que hauria de comportar càlculs una mica més lents.

També es poden fer altres tipus de millores:

- En comptes de recolzar-se en l'administrador per indicar el tipus de cada imatge, es podria fer un sistema de detecció automàtica. Aquest sistema automàtic classificaria les imatges segons si fossin les fotos amb text imprès o els documents manuscrits. Per fer-ho caldria implementar un sistema nou de reconeixement de text en imatges.

- Caldria fer un sistema d'aprenentatge per detectar els moments d'inactivitat i un sistema amb *threads* més avançat que l'actual per poder parar i recuperar càlculs en qualsevol moment. El sistema actual per fer el càlcul d'imatges dona suficients eines per poder fer els càlculs de les imatges en moments d'inactivitat. O bé es poden posar en funcionament mètriques per veure els moments de baixa activitat de l'aplicació i programar-los a aquella hora, o bé es pot forçar a fer els càlculs manualment a la vegada que es deixa inaccessible el servidor. El millor

dels casos seria que el propi servidor web pogués detectar moments d'inactivitat i aprofitar-los per fer càlculs automàticament.

- Ampliant més encara el cas anterior, es pot afegir un sistema de *log* que mostri a l'administrador detalls sobre els càlculs de les imatges. Detalls com a quina hora s'han realitzat i quant de temps han trigat executant-se.

- En termes d'interfície d'usuari sempre es poden fer millores.

  - Es pot afegir suport per *drag and drop* a l'hora d'afegir les imatges.

  - Es podria pensar alguna forma d'afegir més control individual sobre les imatges sense haver de complicar massa d'interfície. Quantes més opcions dones a l'usuari més complexes es fan les interfícies. Per això a la web no hi ha opció per eliminar individualment imatges.

  - El suport de diferents navegadors i diferents versions és un dels problemes més comuns del disseny de webs. En aquest projecte s'han fet relativament poques proves en aquest aspecte, utilitzant només uns pocs navegadors i unes versions concretes. Especialment en el cas d'*Internet Explorer*, que és un dels navegadors més utilitzats i que més problemes comporta, no s'ha fet cap esforç per donar-hi suport. La millora consistiria en fer testos més exhaustius i donar amb navegadors que no s'han mirat.

Referències

[1] Minification. https://en.wikipedia.org/wiki/Minification_%28programming%29. Accedit: 18-06-2015.

[2] Oneshot-for sharing ios screenshots. http://oneshot.link/. Accedit: 18-06-2015.

[3] Representational state transfer. https://en.wikipedia.org/wiki/Representational_state_transfer. Accedit: 18-06-2015.

[4] Jon Almazan, Albert Gordo, Alicia Fornes, and Ernest Valveny. Word spotting and recognition with embedded attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[5] Lluis Gomez and Dimosthenis Karatzas. Multi-script text extraction from natural scenes. In *ICDAR*, 2013.

[6] Simon Marlow et al. The haskell 2010 language report. Technical report, 2010. https://www.haskell.org/onlinereport/haskell2010/#QQ2-15-159.

[7] Michael Snoyman. Preliminary warp cross-language benchmarks. http://www.yesodweb.com/blog/2011/03/preliminary-warp-cross-language-benchmarks. Accedit: 18-06-2015.

[8] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[9] Kai Wang and Serge Belongie. Word spotting in the wild. In *Proc. ECCV*, 2010.