Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño de una aplicación de búsqueda de imágenes vectoriales mediante su estilo visual

Autor

Daniel Osanz Tello

Directora

Elena Garcés

Ponente

Diego Gutierrez

Escuela de Ingeniería y Arquitectura

2013

Tomo 2/2

Índice de Anexos

Apéndice A

Resultados pruebas con aplicaciones

Los resultados completos obtenidos en las pruebas realizadas para evaluar la búsqueda por similaridad en Google Imágenes, Pixolution, la galería de Microsoft Office, y Xerox se muestran en Las Figuras 1.2, 1.3, 1.4 y 1.5 respectivamente.

Se recuerda que para la prueba con Google Imágenes y Pixulution se utilizaron las imágenes patrón que se muestran en la Figura 1.1, mientras que en las demás aplicaciones se utilizaron las imágenes que se explica en el comentario de la figura debido a que trabajaban con imágenes de la base de datos de la propia aplicación.

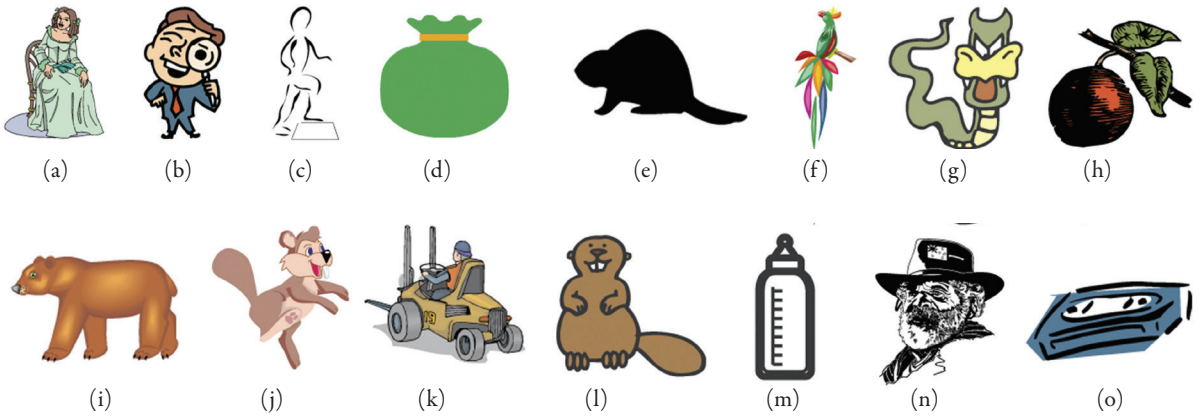


Figura 1.1: ***Imágenes patrón*** *utilizadas en las pruebas con Google Images y Pixolutión, teniendo como característica marcada: (a) Contorno fino, (b) Contorno grueso, (c) Contorno irregular, (d) Sin contorno(e) Blanco y negro, (f) Muchos colores, (g) Pocos colores, (h) Sombreado duro, (i) Transición suave, (j) Transición dura, (k) Forma 3D, (l) Forma plana, (m) Forma simple, (n) Forma compleja, (o) Relleno por fuera.*

Figura 1.2: ***Resultados de búsqueda de Google Images*** *tomando como referencia las correspondientes imágenes de la* Figura *1.1.*

Figura 1.3: ***Resultados de búsqueda de Pixolution*** *tomando como referencia las correspondientes imágenes de la* Figura *1.1.*

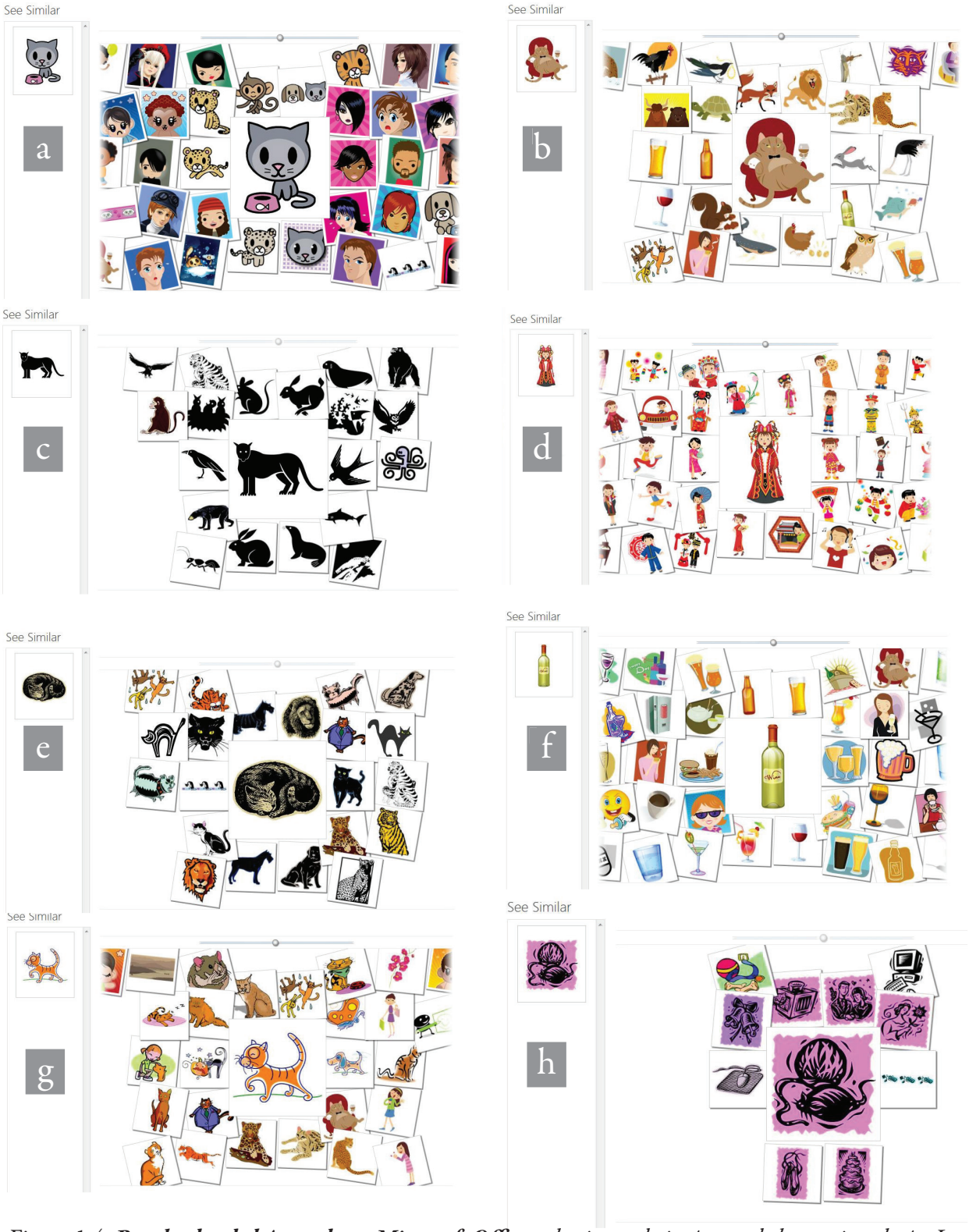


Figura 1.4: ***Resultados de búsqueda en Microsoft Office****, seleccionando imágenes de la propia galería. Las imágenes patrón se muestran en la columna aparte a la izquierda.*

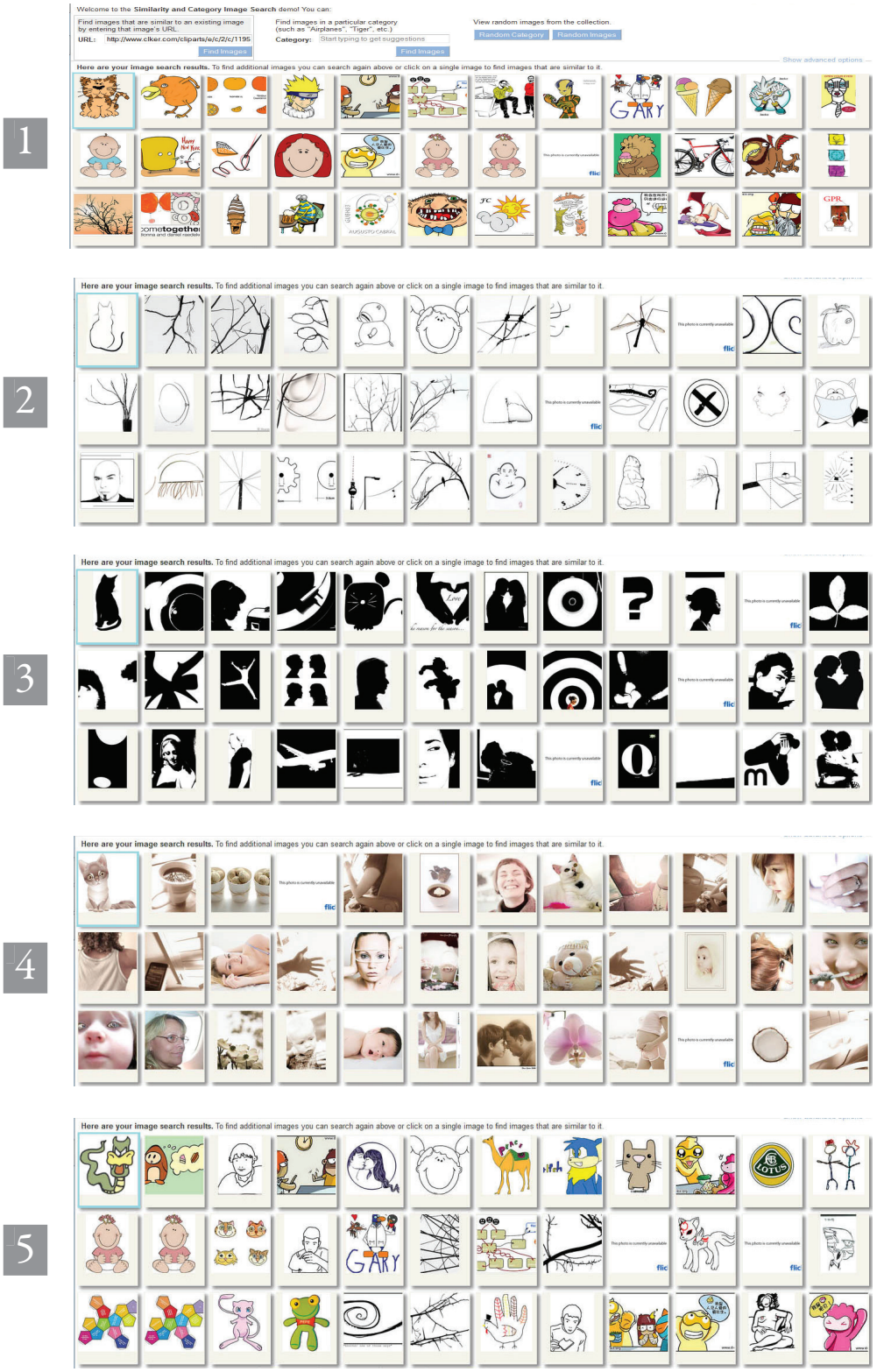
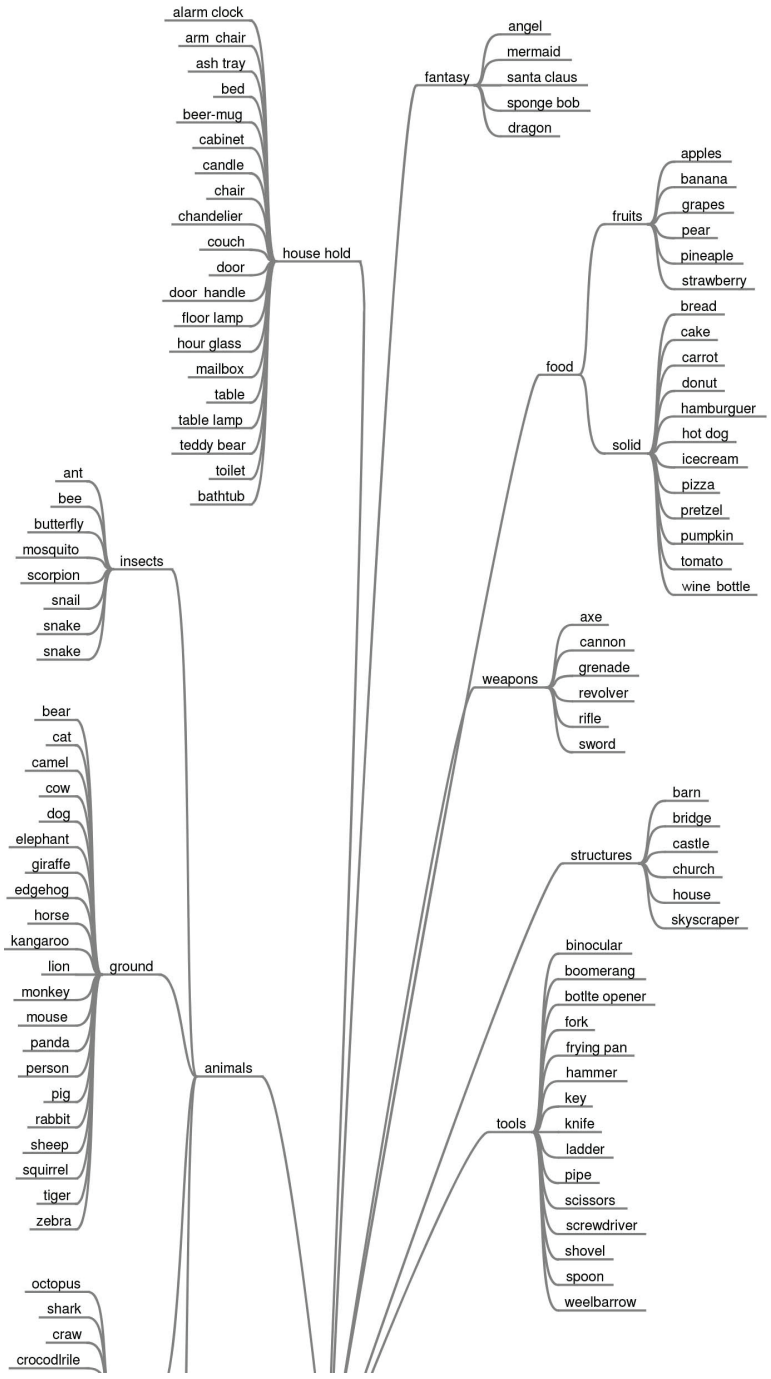


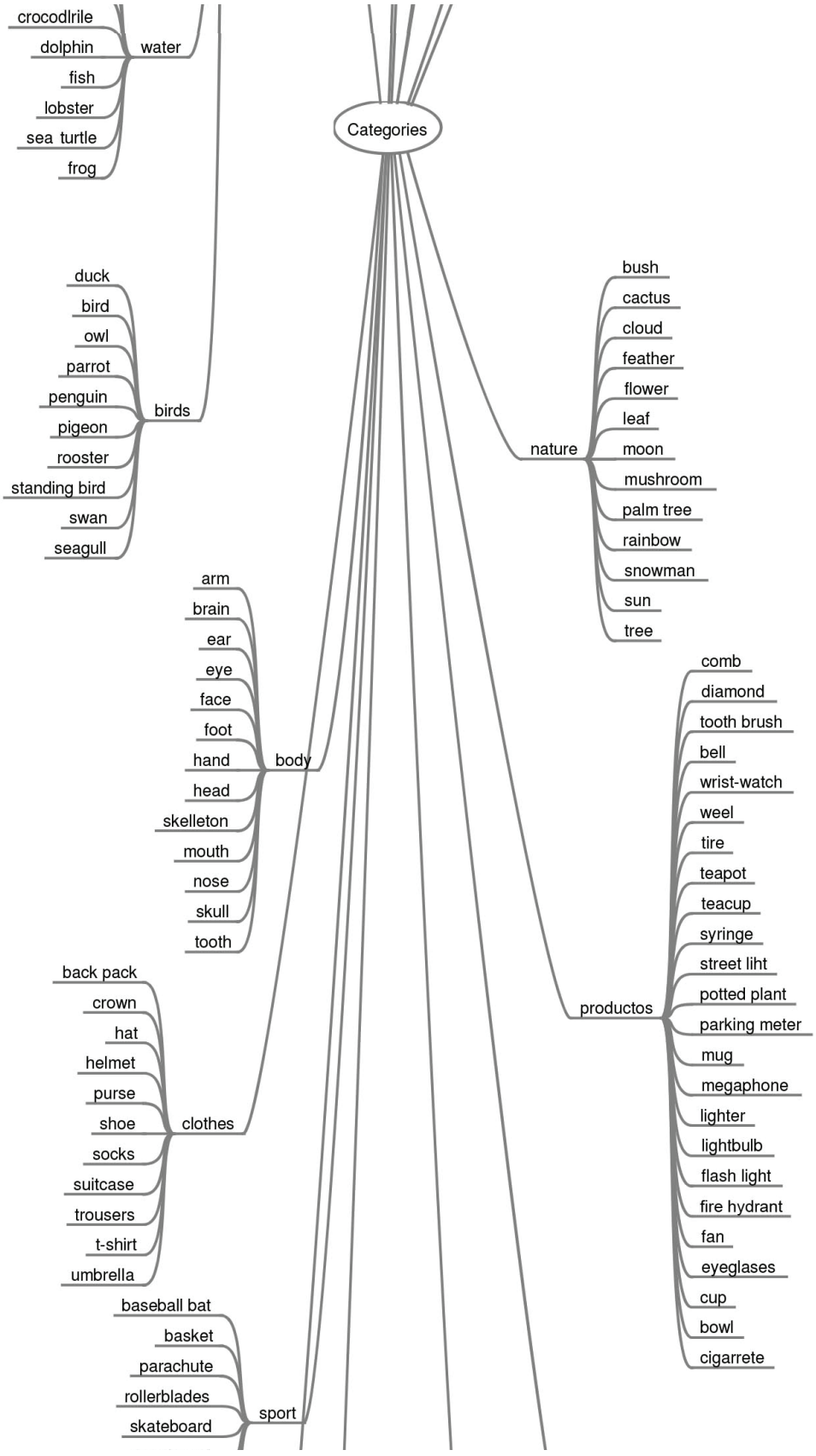
Figura 1.5: ***Resultados de búsqueda en Xerox,*** *seleccionando imágenes de la propia galería. Las imágenes patrón se muestran como primer resultado de búsqueda (arriba a la izquierda).*

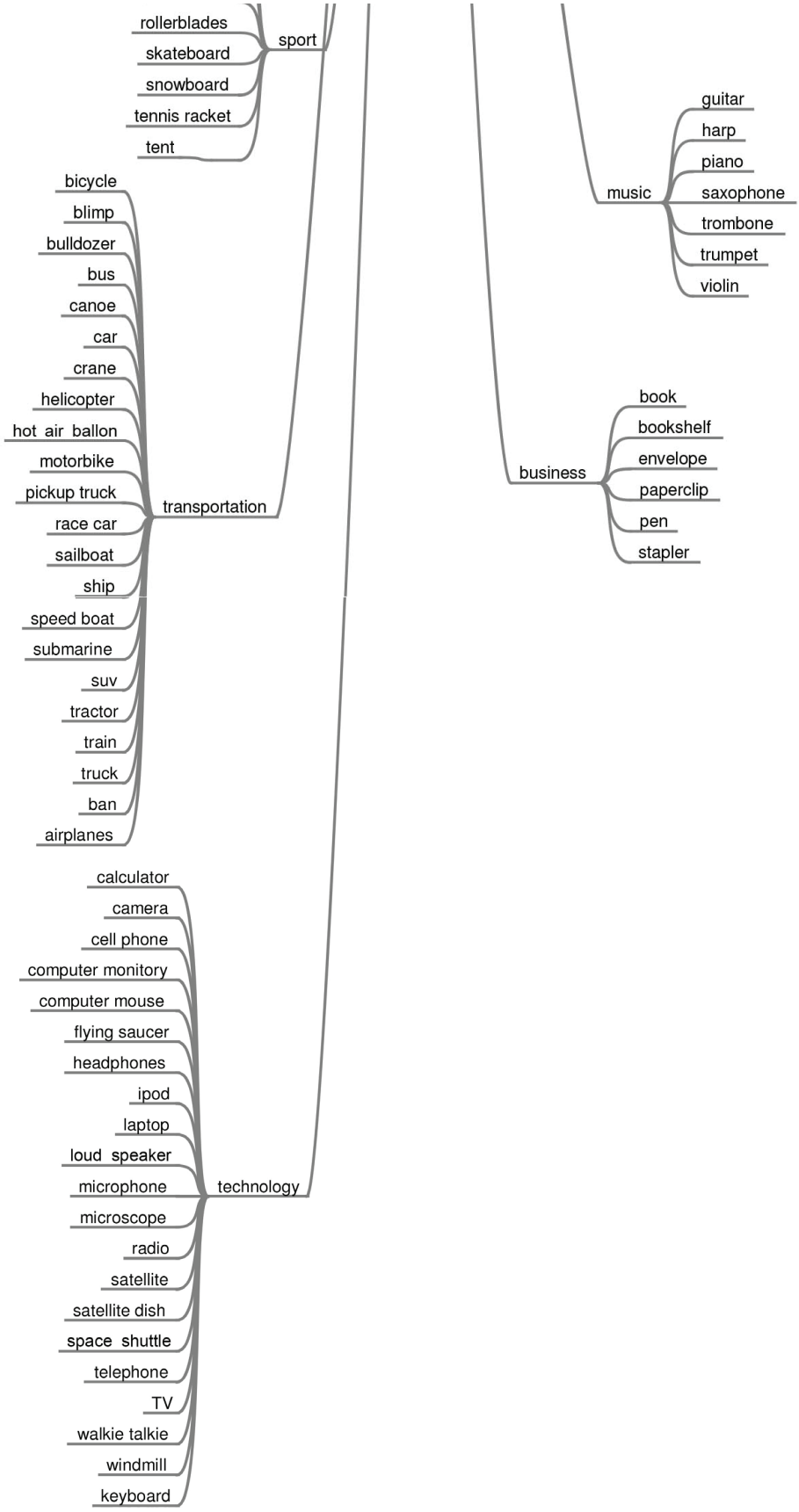
Apéndice B

Categorías seleccionadas

A continuación se muestran las categorías del paper *Caltech-256 Object Category Dataset* organizadas en los grupos que se tomaron como principales.




Categories
water
crocodlrile
dolphin
fish
lobster
sea turtle
frog
birds
duck
bird
owl
parrot
penguin
pigeon
rooster
standing bird
swan
seagull
nature
bush
cactus
cloud
feather
flower
leaf
moon
mushroom
palm tree
rainbow
snowman
sun
tree
body
arm
brain
ear
eye
face
foot
hand
head
skelleton
mouth
nose
skull
tooth
productos
comb
diamond
tooth brush
bell
wrist-watch
weel
tire
teapot
teacup
syringe
street liht
potted plant
parking meter
mug
megaphone
lighter
lightbulb
flash light
fire hydrant
fan
eyeglases
cup
bowl
cigarrete
clothes
back pack
crown
hat
helmet
purse
shoe
socks
suitcase
trousers
t-shirt
umbrella
sport
baseball bat
basket
parachute
rollerblades
skateboard


rollerblades
skateboard
snowboard
tennis racket
tent
sport
bicycle
blimp
bulldozer
bus
canoe
car
crane
helicopter
hot air ballon
motorbike
pickup truck
race car
sailboat
ship
speed boat
submarine
suv
tractor
train
truck
ban
airplanes
transportation
calculator
camera
cell phone
computer monitory
computer mouse
flying saucer
headphones
ipod
laptop
loud speaker
microphone
microscope
radio
satellite
satellite dish
space shuttle
telephone
TV
walkie talkie
windmill
keyboard
technology
guitar
harp
piano
saxophone
trombone
trumpet
violin
music
book
bookshelf
envelope
paperclip
pen
stapler
business

Apéndice C

Funcionalidades brainstorming

A continuación se muestra la lista de aplicaciones obtenida mediante el brainstorming organizada por funcionalidades básicas, avanzadas y generales como se explicó en la memoria.

Funcionalidades básicas:

- Unificar todas las imágenes de una composición adaptándolas a un mismo estilo para lograr coherencia.
- Añadir elementos del mismo estilo para rellenar espacio simplemente con buscar la categoría correspondiente.
- Detectar que elemento no mantiene una relación de estilo con el resto de los elementos, remarcándolo en un color y dando la posibilidad de eliminarlo, sustituirlo o mantenerlo.
- Dibujar un esquema básico del objeto a dibujar y con la ayuda del programa de la publicación *How Do Humans Sketch Objects* sustituya esta primera aproximación con un objeto del mismo estilo que el resto de la composición
- Copiar el estilo de una imagen y usarlo en otras, de manera similar a la herramienta cuentagotas de los programas de edición de imágenes.
- Generar imágenes de manera aleatoria parecidas a una dada con la ayuda de elementos básicos prefijados, sobre todo para formas simples o abstractas como botones, cuadros web, logos...
- Posibilidad de guardar los estilos que te gustan en el panel Estilos, en el cual también se encuentran estilos generales definidos por el programa y estilos para crear composiciones gráficas (cartel ciencia ficción, publicidad, disco pop...)
- Evaluar/puntuar cohesión, de forma que se puedan clasificar las imágenes por parecidos ya sea buscando la "perfección" o lo "innovador/diferente"
- Buscar una imágenes "vectorial/clip art" parecida a una "bitmap/real" y viceversa para poder transformar una composición clip art en una realista y viceversa.
- Posibilidad de transformar una composición con imágenes no relacionadas de diferentes estilos en una coherente con solo presionar un botón.
- Buscar imágenes de un estilo concreto en internet o entre tus imágenes.
- Marcar elementos diferentes o parecidos entre dos imágenes.
- Generar una lista relacionando características de una imagen con objetivos conseguidos como por ejemplo el borde grueso expresa dureza.
- Lista relacionando características con sus "funciones" (trazo grueso expresa confianza)

Funcionalidades avanzadas:

- Si detecta cual es el contorno interior y exterior de una imagen se podría analizar la forma de un elemento de la composición para asignarle otra imagen de la misma categoría semántica gracias a la aplicación de la publicación *How Do Humans Sketch Objects*.

- Si detecta la iluminación podrían buscar imágenes solo con esa iluminación o incluso intentar dibujarla encima de la imagen.
- Si analiza reglas de composición, en función del tamaño y los objetos situados en la composición podría recomendar donde ponerlos, que añadir y que eliminar.
- Si es capaz de editar características podría editar la imagen automáticamente para adaptarla al estilo de otro elemento.
- Si es capaz de valorar características podría decir cuanto de buena es una fotografía.
- Si es capaz de analizar la forma y compararla con otras imágenes podría comparar un texto con caracteres "universales" de forma que sepa lo que hay escrito.
- Si es capaz de reconocer formas en una imagen en jpg podría "extraer"partes con un estilo diferente y modificarlas por otras del mismo estilo.
- Si es capaz de generar imágenes de manera aleatoria podría generar imágenes nuevas (o combinar imágenes) que se parezcan a un estilo.

Funcionalidades generales:

- Composiciones gráficas: seleccionando qué se quiere realizar (cartel de cine, portada disco...) y estilo (ciencia ficción, romántica...) el programa crea un ejemplo de composición o adapta los elementos existentes.
- Creación de escenarios y personajes en videojuegos: analizando las texturas de todos los elementos para comprobar que tienen una relación de estilo.
- Diseño web: analizando el parecido entre los diferentes botones, cuadros, imágenes...
- Google Glass: búsqueda de objetos, seleccionar un objeto con un estilo y busca parecidos (en una tienda, arquitectura, cuadro...)
- Google Glass, cámara fotos: búsqueda de buena fotografía, mostrando una foto que te guste de referencia te aconseja lugar y encuadre.
- Diseño de logos: busca el logo más parecido a otro que te gusta (o incluso los genera automáticamente dibujándolos o fusionando elementos básicos prefijados)
- Diseño de Packaging: rellenando la caja con elementos parecidos a los de uno que te gusta y adaptándolos para que no se diferencien mucho de las fotos que se quieran emplear sobre el producto a vender.
- Decoradores: buscando muebles parecidos a los del estilo de la casa. También analizaría el estilo de la casa.
- Organizar fotos en carpetas: en función de su estilo (hdr, blanco y negro, callejeras, clip art...)
- Facebook: clasificar fotos (ó recomendar) ciertas fotos para diversas carpetas (foto de perfil, banner, deporte, montaña...)
- Google imágenes: buscador inteligente que conforme se descargan fotos las analiza y centra las posteriores búsquedas en el estilo de las descargadas.
- Spotify: recomendación de discos (función radio) en función de los estilos de la portada.
- Películas,videojuegos...: recomendación de películas en función de las carátulas, posters.
- Comparación de personas o objetos: a partir de un retrato/foto busca imágenes parecidas (retrato robot, búsqueda de un objeto o cuadro que te gusta...)
- Ejercicios memoria: te muestra imágenes y tienes que decir cuales se parecen, al final te devuelve cuantas has acertado.
- Diseño de producto: ayuda a diseñar adecuadamente la estética de un producto para que sea coherente en su totalidad.

- Anuncios, publicidad: te recomiendan los anuncios de estilo parecidos a lo que te gusta.
- Recomendación de fotografías: en redes de fotografía como Picasa en función de un estilo.
- Creación de Collages: seleccionando las imágenes parecidas.
- Tunear coche o bicicleta: con pegatinas parecidas a un estilo.
- Tatuajes: recomendación de tatuajes parecidos a un estilo.
- Diseño de interfaz: buscando botones, cuadros, texto... parecidos para web,programas,menús de videojuegos...
- Analizar tipografías: viendo cuales se parecen a las usadas y aconsejando algunas en función del estilo de la composición sobre la que van.
- Video, telediario: hacer que una cabecera con texto o imagen pase desapercibida del fondo.
- Diseño rápido de camisetas: para alguien que quiere hacerse un dibujo para una camiseta rápido y simplemente seleccionando imágenes que le gustan
- Diseño de bolsas de plástico: creando elementos auxiliares parecidos al logo de la empresa.
- Recomendaciones "diferentes": en vez de recomendarte una película o grupo por genero recomienda por portada parecida.
- Publicidad: poner en carteles o anuncios elementos que les guste a toda la gente o al público objetivo (el programa puede almacenar características que les gusta a la gente).
- Retoque fotográfico: detectar imperfecciones en composición o piel de la persona y corregirlas automáticamente.
- Base de datos: categorizar rápidamente muchas imágenes.
- Interfaz Windows, navegador, videojuego: cambiar la apariencia en función del estilo de una foto.
- Creación de un personaje en videojuego: selecciono un estilo se busca o adpta una textura con ese estilo.
- Recomendación de películas, discos videojuegos: en función de imágenes que le vas marcando que te gusta como la radio de Spotify.
- Corrección de dibujos: por comparación detecta los parecidos y los diferentes.
- Arte: ayudar a buscar las características de un pintor o cuadro.
- Textos formales: adaptar imágenes, gráficos y maquetación al texto.
- Medicina: analizar diferencias entre una radiografía perfecta y la hecha.
- Pintura: ver cuanto se parece el dibujo que acabas de hacer al real.
- Texturas: analizar imperfecciones en una textura.
- Biología: clasificar animales parecidos mediante características comunes a una raza.

Apéndice D

Interfaz aplicación

A continuación se muestran imágenes de la interfaz definitiva de la aplicación.

Figura 1: *Aplicación definitiva con pestañas abiertas y ocultas.*

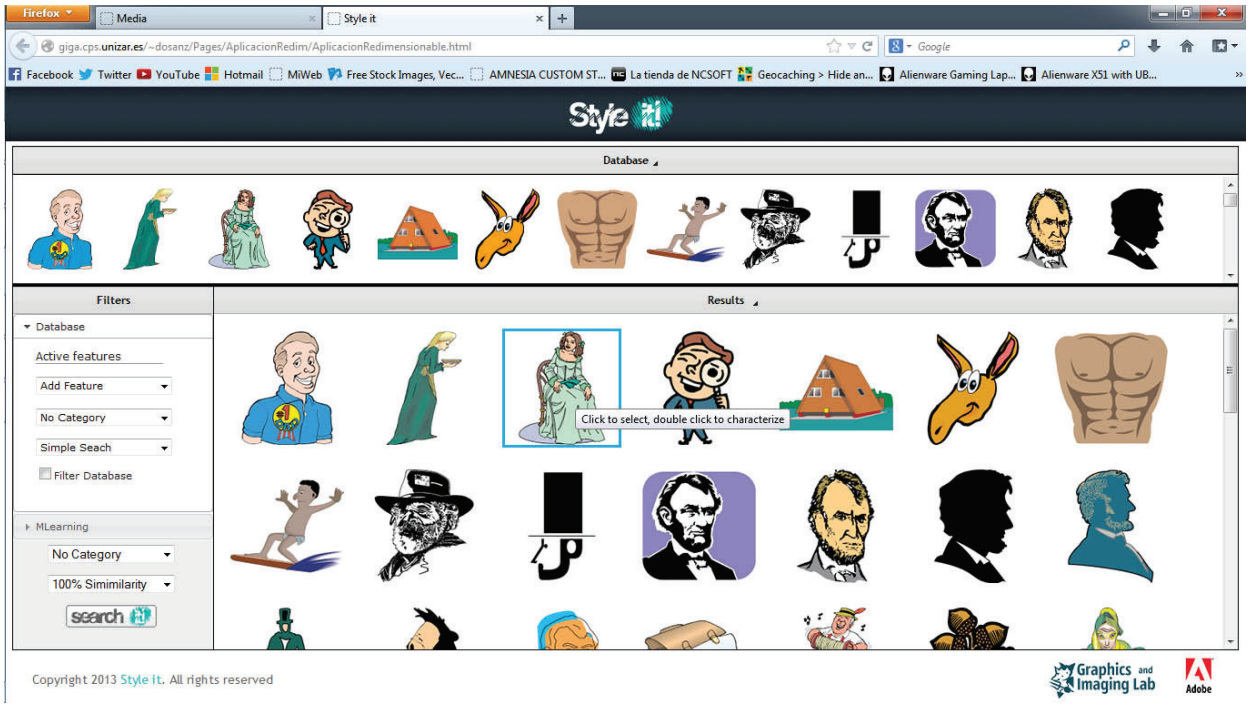


Figura 2: *Muestra de los paneles redimensionables*

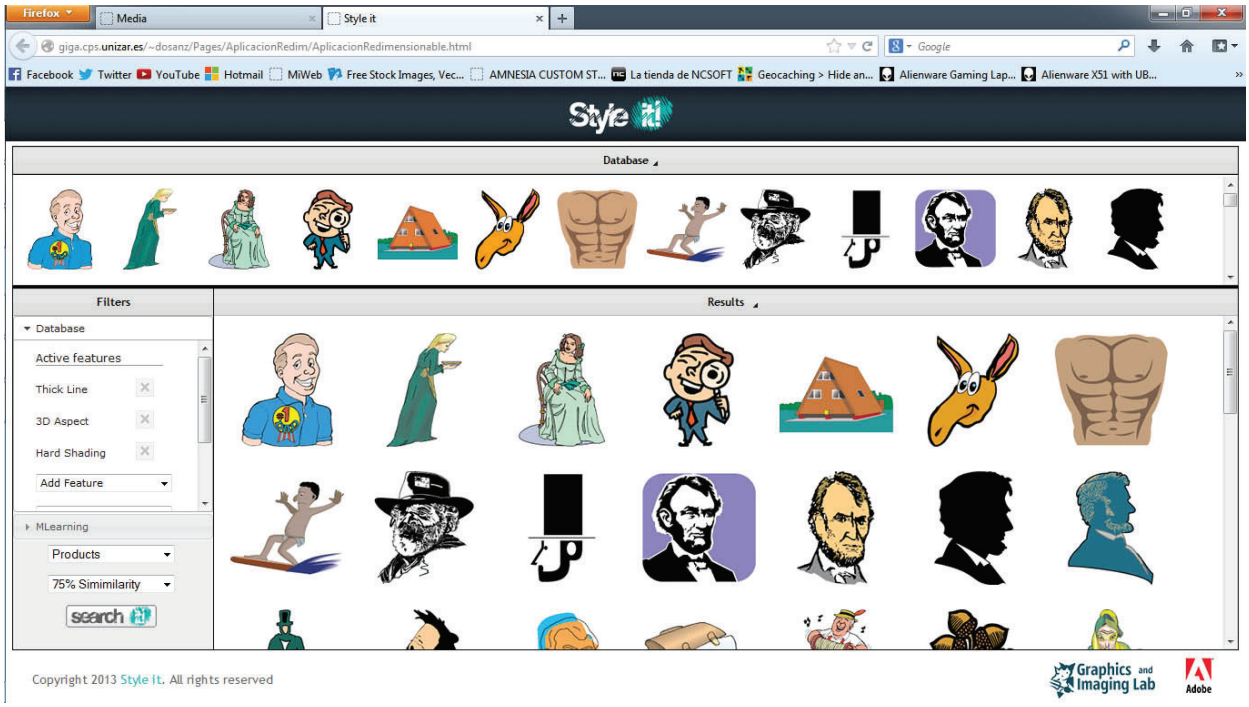


Figura 3: *Uso de filtros para la búsqueda*

Apéndice E

Manual de marca

El manual de marca de la aplicación style it se muestra en las páginas siguientes.

Style it!

MANUAL DE MARCA

Daniel Osanz Tello

Proyecto Final Carrera

Universidad Zaragoza 1542

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Graphics and Imaging Lab

Adobe

ÍNDICE

ÍNTRODUCCIÓN

Style it es una aplicación desarrollada por Elena Garcés y Daniel Osanz Tello en colaboración con Adobe destinada al análisis de similitud entre imágenes vectoriales.

Este manual de marca unifica criterios y normaliza el manejo de la imagen gráfica y corporativa de la marca, con el fin de orientar a los miembros de la empresa, colaboradores y proveedores en la forma de hacer un buen uso de los lemas y signos institucionales.

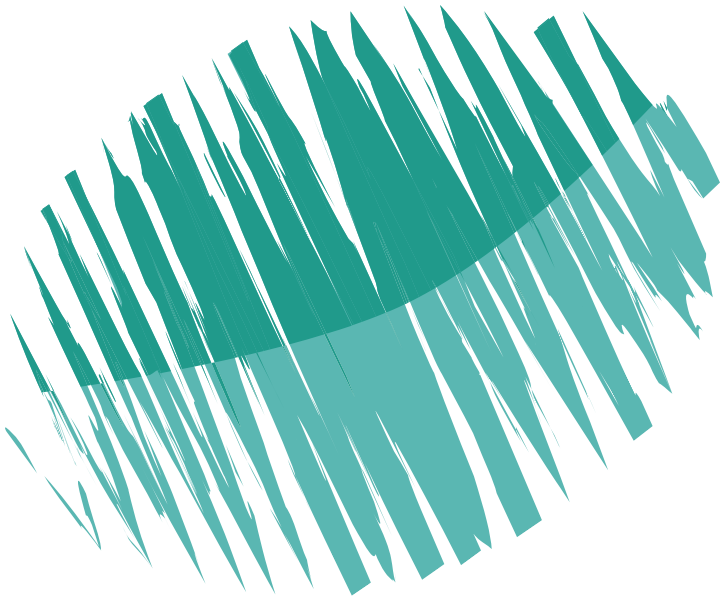
1.1 SÍMBOLO

El símbolo de STYLE IT se basa en una elipse girada formada por trazos, una de las características fundamentales de las imágenes vectoriales. La elipse, al ser una forma simple y sin aristas representa la sencillez de uso de la aplicación.

Los trazos irregulares que forman el símbolo imitan a los que se podrían obtener con un pincel, representando un estilo de los que la aplicación es capaz de interpretar, y tienen un color verde azulado para mantener una relación con el color del logo del GILAB. También se han usado diferentes colores para cada una de las mitades del símbolo con el fin de hacerlo más visual.

La rotación del símbolo es debida a la tipografía del logotipo ya que si este estuviese en otra orientación sus dimensiones serían demasiado grandes y destacaría sobre el texto.

El sentido del trazo del símbolo es simétrico al de las rayas del texto para mejorar la legibilidad. de este



1.2 LOGOTIPO

El logotipo, compuesto por las palabras Style it! irá siempre acompañada del símbolo y en las proporciones que se muestran más adelante. La tipografía escogida es la Disco Night modificada, a la que se le ha añadido la ranura de la e y rediseñado el símbolo de exclamación que no estaba incluido.

El idioma empleado es inglés ya que es el universal y la aplicación va destinada a todo el mundo. Style viene de que la aplicación analiza estilos y el it! es para hacerlo más ameno y fácil de recordar, desafiando al usuario a que pruebe la aplicación.

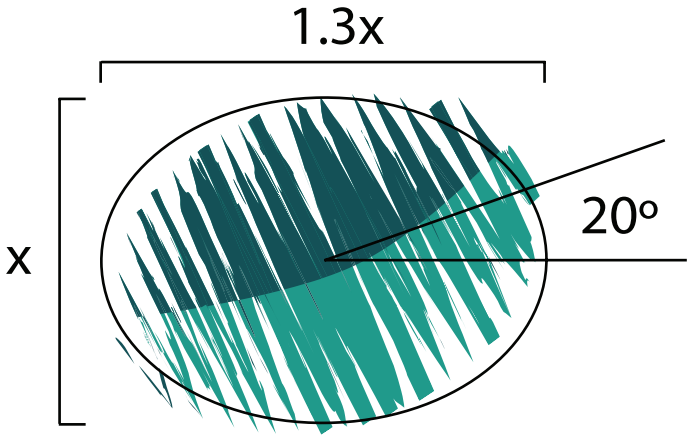
1.3 IMAGOTIPO

Este imagotipo será el identificador principal de la aplicación Style it! y la imagen gráfica con la que se identificará la misma.
A la izquierda se sitúa el logotipo y a la derecha por detrás del it! se encuentra el símbolo, siempre manteniendo las proporciones que serán descritas en las próximas hojas.

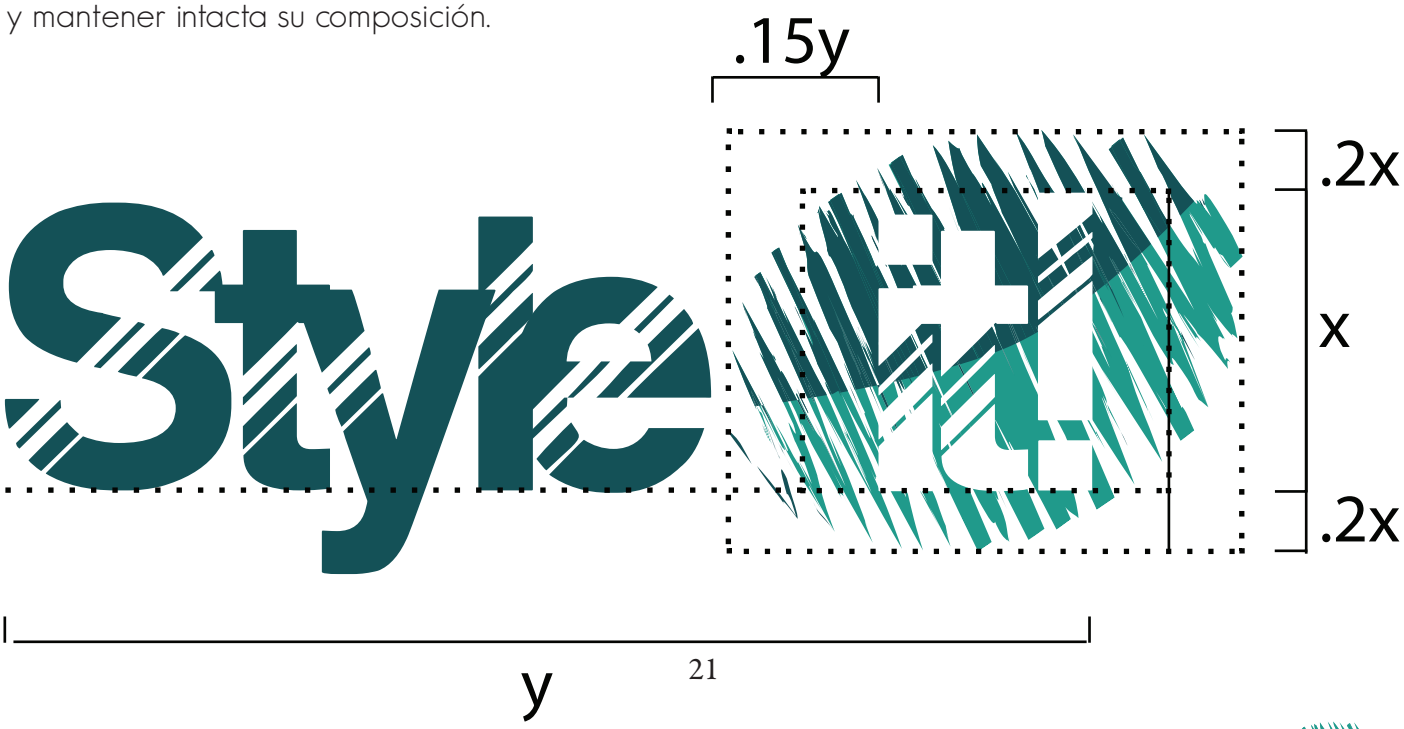
El imagotipo representa un estilo muy característico que con la aplicación se podría obtener y hace referencia a la sencillez de uso de la misma.

1.3.1 CONSTRUCCIÓN

La imagen de la derecha muestra la relación existente entre los ejes de la elipse así como la rotación a la que está sometida.

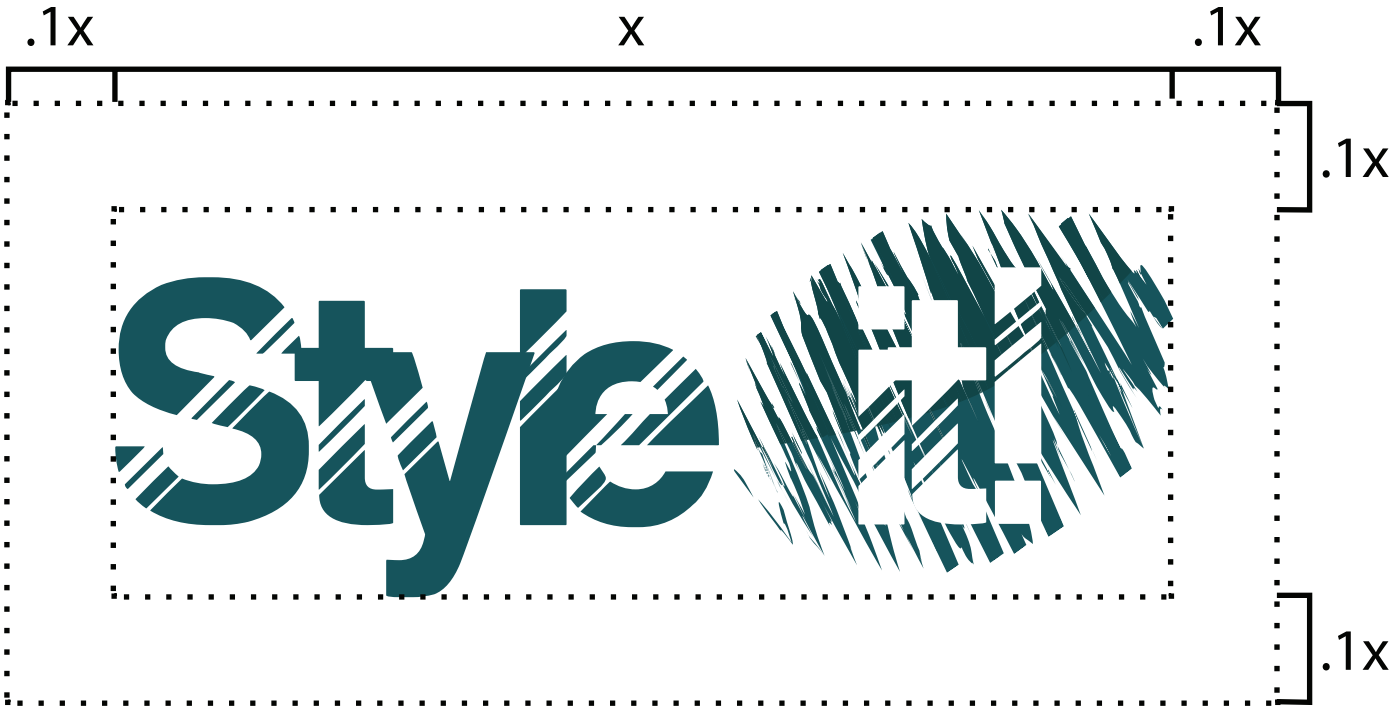


En cuanto a la posición del símbolo respecto al logotipo se realizará teniendo en cuenta la palabra it!, de manera que esta quede centrada dejando un espacio de 0.2 veces la altura de los caracteres. A su vez, el espacio entre las palabras Style e it será igual a 0.15 veces el tamaño de la cadena de palabras entera.
Siempre que se utilice este imagotipo deberá respetar las dimensiones de su símbolo y logotipo y mantener intacta su composición.

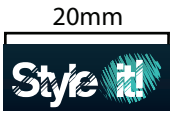


1.3.2 ÁREA DE RESPETO

Para asegurar la correcta legibilidad y dignidad del imagotipo y que no se pueda confundir con otros elementos de la composición se ha establecido un área en blanco alrededor de este de una décima parte de su longitud horizontal.



Para la correcta legibilidad y visualización del imagotipo la anchura mínima no deberá ser inferior a 20 mm.



1.3.4 VARIANTES CROMÁTICAS

En el caso de que no se disponga de los Pantones adecuados para la impresión se podrá emplear la siguiente versión a una tinta usando el Pantone 7541 C como sustituto de los anteriores.

En el caso de que Solo se disponga de tinta negra se deberá usar la versión a una tinta anterior siempre y cuando el fondo sea lo suficientemente claro para no dificultar la legibilidad.

En caso de aplicar el logotipo sobre un fondo oscuro se utilizará como color para el imagotipo el blanco y se cambiarán los colores de este como se especifica en la siguiente página.

Versión a 2 tintas sobre fondo oscuro

En caso de no disponer de los Pantones adecuados al imprimir sobre un fondo oscuro se deberá usar la versión en blanco del logotipo que se muestra a continuación.

Versión a 1 tinta sobre fondo oscuro

Como fondo oscuro se recomienda el uso del Pantone 303 C, que permite una correcta legibilidad del imagotipo y mantiene la gama de colores de este.

Aplicación sobre fondo oscuro PANTONE 303 C

En el caso de cartelería, se podrá usar un degradado vertical sobre las letras del logotipo, siempre y cuando sea de blanco a gris claro y no dificulte la legibilidad..

Versión con degradado en letras

1.4 COLORES CORPORATIVOS

La identidad corporativa de Style it! quedará reforzada por el uso del color utilizado en el imagotipo. El color escogido para el símbolo tiene su origen en el del color del logotipo del GLab, por lo que se asociará directamente al mismo.

Es muy importante que se utilicen correctamente los colores en sus diferentes versiones, mostradas a continuación. Siempre que sea posible se utilizará tinta Pantone.

PANTONE 7473 C
CMYK 79c 17m 51y 2k
RGB 30 R / 152 G / 138 B

PANTONE 7472 C
CMYK 64c 5m 35y 0k
RGB 90 R / 182 G / 178 B

PANTONE 7476 C
CMYK 89c 43m 50y 42k
RGB 7 R / 80G / 86 B

PANTONE 7473 C
CMYK 79c 17m 51y 2k
RGB 30 R / 152 G / 138 B

1.5 USOS INDEBIDOS

Para asegurar la coherencia e identidad de Style it! es recomendable seguir las normas anteriormente comentadas.
Aquí exponemos una colección de usos indebidos del imagotipo

Imagotipo original

Usos indebidos

Uso de tipografía incorrecto

Uso de proporciones incorrectas

Incorrecta disposición símbolo-logotipo

Incorrecta disposición símbolo-logotipo

Uso indebido de colores corporativos

Uso incorrecto de proporciones y disposición

1.6 TIPOGRAFÍA CORPORATIVA

Emplear una misma tipografía para la comunicación dentro y fuera de la empresa reforzará la imagen y aportará cohesión. Así pues se instauran dos familias de tipografías como mostramos a continuación.

La GEOSANS LIGHT se usará siempre que sea posible, puesto que transmite correctamente los valores e ideas de Style it! (sencillez, amistad, facilidad...).

En caso de desear una tipografía más formal y con gran variedad de familias, como podría interesar en el caso de cartelería o presentaciones, también se considera adecuado el empleo de la HELVETICA NEUE LT PRO. Especialmente, se recomienda el uso de las familias 65 medium, 25 UltraLight y 45 Light.

GeosansLight
ABCDEFGHIJKLMNÑOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz
123456789 .:;()¿!¿?' "+-*/%=_

Helvetica Neue LT Pro 65 Medium
ABCDEFGHIJKLMNÑOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz
123456789 .:;()¿!¿?' "+-*/%=_

Helvetica Neue LT Pro 25 UltraLight
ABCDEFGHIJKLMNÑOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz
123456789 .:;()¿!¿?' "+-*/%=_

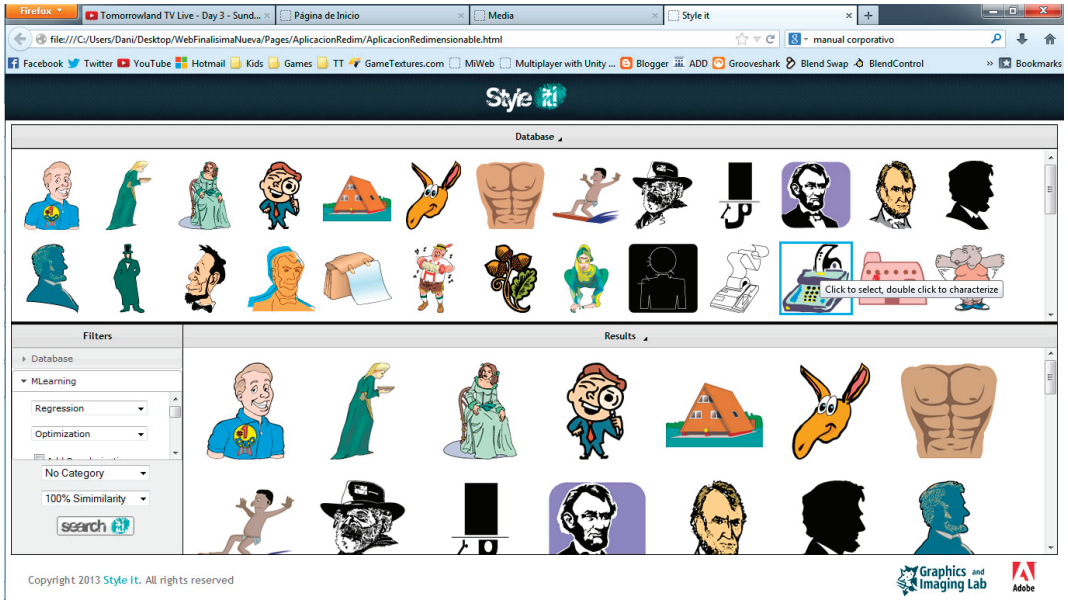
Helvetica Neue LT Pro 45 Light
ABCDEFGHIJKLMNÑOPQRSTUVWXYZ
abcdefghijklmnñopqrstuvwxyz
123456789 .:;()¿!¿?' "+-*/%=_

1.7 APLICACIONES

En esta sección se recomiendan directrices sobre el uso del imagotipo así como el de las fuentes tipográficas para crear composiciones reforzando la imagen de marca que se pretende mostrar.

1.7.1 WEB

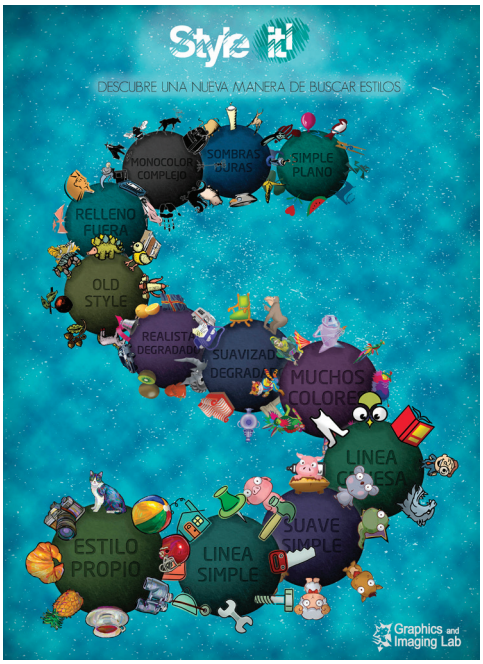
A la hora de añadir el imagotipo de Style it! a páginas web se recomienda su uso en una cabecera o pie de página con un fondo oscuro de los recomendados anteriormente. Además se separará el logo claramente de cualquier otro texto o icono que haya cerca de este.



1.7.2 CARTELERÍA

Respecto a la cartelería, el imagotipo se posicionará en un lugar de gran peso compositivo como podría ser arriba del todo o en el centro del soporte, el tamaño será suficiente para su correcta identificación, mantendrá separación adecuada con los demás elementos de la composición para que destaque más, y se preferirá la versión sobre fondo oscuro siempre y cuando la composición lo permita.

Excepto usos autorizados, no se podrá situar ningún elemento de la composición por encima del logotipo o modificar los colores de este.

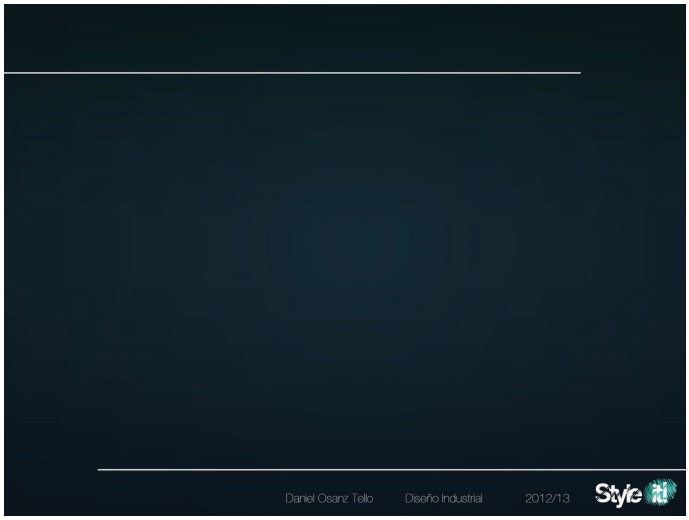


1.7.3 PRESENTACIONES

Para terminar se muestra la aplicación del imagotipo en una plantilla para realizar presentaciones relacionadas con Style it!.

En la portada se mostrará el logotipo sobre un fondo oscuro con degradado verde-azulado ocupando la mayor parte del espacio y debajo de este, y utilizando la tipografía Helvética mencionada en la sección correspondiente de este manual, irá el nombre de la presentación, autores y logotipos relacionados.

Para las páginas de la presentación se utilizará el mismo fondo que la portada con un viñeteado, el título de la página irá en la parte de arriba separado por un trazo blanco horizontal y abajo se mostrará el logo de Style it! con información sobre autores, fechas, y otros datos relacionados con la presentación. Se jugará con el uso de las familias tipográficas para destacar lo más importante, recomendando el uso de la negrita para otorgar importancia al texto.



Apéndice F

Pósters

Los póster realizados para promocionar la aplicación se muestran en las siguientes páginas.

StyleIt

Descubre la aplicación de búsqueda de estilos similares

Graphics and Imaging Lab

Style it

DESCUBRE UNA NUEVA MANERA DE BUSCAR ESTILOS

Graphics and Imaging Lab


MONOCOLOR COMPLEJO
LINEA GRUESA
MUCHOS COLORES
RELLENO FUERA
SUAVE SIMPLE
SOMBRAS DURAS
REALISTA DEGRADADO
SIMPLE PLANO
SUAVIZADO DEGRADADO
ESTILO PROPIO
LINEA SIMPLE
OLD STYLE
Style it!
Graphics and Imaging Lab


MONOCOLOR COMPLEJO
LINEA GRUESA
MUCHOS COLORES
RELLENO FUERA
SUAVE SIMPLE
REALISTA DEGRADADO
OLD STYLE
LINEA SIMPLE
ESTILO PROPIO
SUAVIZADO DEGRADADO
SIMPLE PLANO
SOMBRAS DURAS
Style it!
Graphics and Imaging Lab

Apéndice G

Sitio web

Imágenes del sitio web

A continuación se muestran imágenes del sitio web de la aplicación.

Figura 4: *Página de inicio de la web*

Figura 5: *Información de la web*

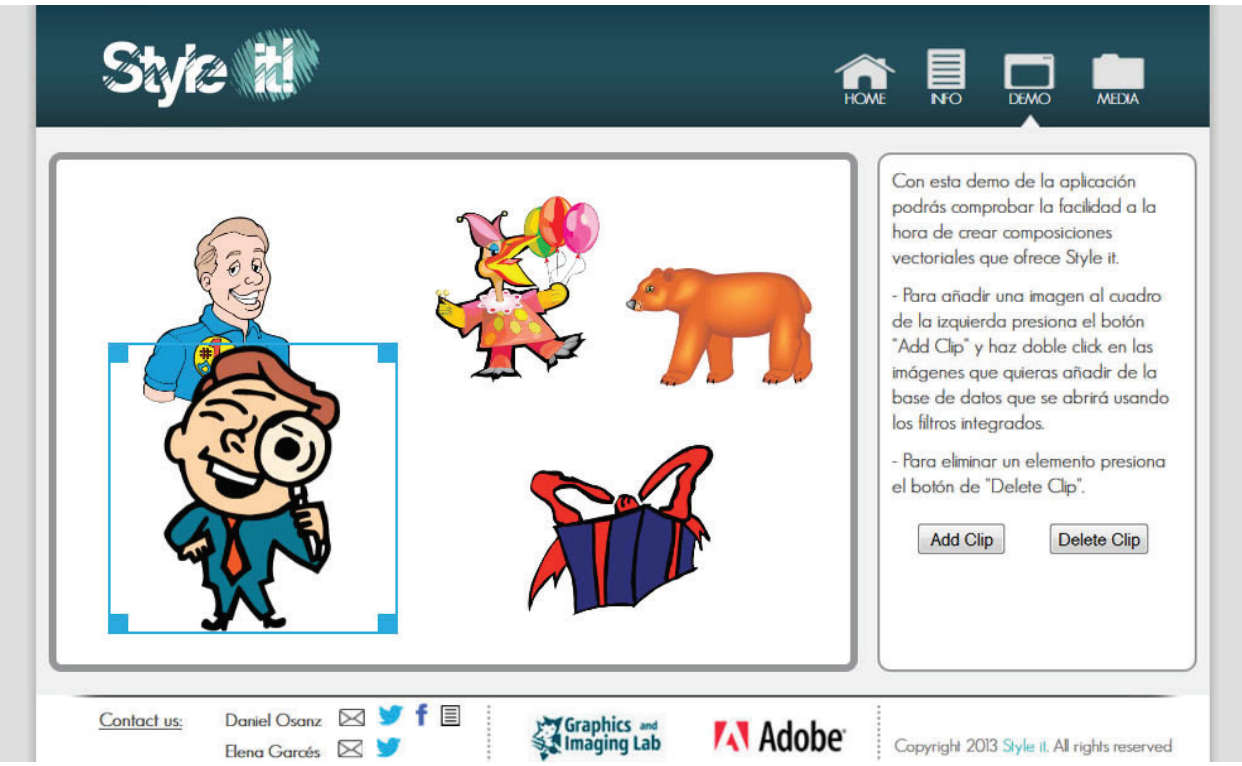


Figura 6: *Demo de la web*

Figura 7: *Contenido multimedia de la web*

Apéndice G

Sitio web

Explicación código

En las siguientes páginas se explicará el código utilizado en javascripps, css y html para desarrollar por completo del sitio web, teniendo en cuenta tanto el definitivo como las pruebas que se realizaron.

Para realizar el movimiento de una imagen por un cuadro se basó en el código libre de Simon Sarris (http://simonsarris.com/blog/510-making-html5-canvas-useful), en el cual utiliza un elemento canvas y un script que actualiza la página al instante (30 milisegundos en este caso) cuando se arrastra el elemento borrando la estela que dejaría al moverse.

El elemento canvas se define de la siguiente manera en el html:

```
<canvas id="canvas" width="400" height="300">
  This text is displayed if your browser does not support HTML5 Canvas.
</canvas>
```

Es importante definir la altura y anchura en el html y no en el css puesto que sino usa una transformación reduciendo drásticamente la resolución del contenido del canvas.

La siguiente función lee la ruta de la imagen y la asigna a una variable que usará el constructor de objetos:

```
function GetSrc(f){
  rutmos =f.src;
}
```

La siguiente función crea una imagen en el canvas en función del valor de la variable anterior:

```
function Shape(x, y, w, h, impPo) {
 imgPo = new Image();
  imgPo.src = rutmos;
  this.x = x || 0;
  this.y = y || 0;
  this.w = w || 1;
  this.h = h || 1;
  this.imgPo = imgPo || 'Images/Basdat/1.png';
}
Shape.prototype.draw = function(ctx) {
   ctx.drawImage(this.imgPo, this.x, this.y, this.w, this.h);
}
```

Para añadir una foto cuando el usuario clica en su previsualización (img con id específico) se usa el siguiente código:

```
PonImg4.addEventListener('click', function(e) {
 var mouse = myState.getMouse(e);
   myState.addShape(new Shape(275, 175, 150, 150, 'Images/Basdat/2.png'));
 }, true);
```

Para mostrar muchas previsualizaciones de imágenes una seguida de otra y que se pueda navegar por todas se necesita un div contenedor que permita tener un scroll.

Definición en el css de clases para mostrar o no mostrar el scroll:

```
.hiddenscroll{overflow:hidden;
}
.showscroll{overflow:scroll;
            overflow-x:hidden;
}
```

Función que cambia el scroll de visible a oculto:

```
function BarraCambia(elemento) {
 if(elemento.className == "hiddenscroll")
  elemento.className ="showscroll"
 else elemento.className ="hiddenscroll"
}
```

Integración en el html:

```
<div id="ContenedorImagenes" class="hiddenscroll" onmouseover="BarraCambia(this)"
onmouseout="BarraCambia(this)">
  </div>
```

Para crear las previsualizaciones de las imágenes hay que crear un div dentro de otro div:

```
function anadFoto() {
  var div = document.createElement("div");
   div.style.position = "absolute";
   div.style.left = "20px";
   div.style.top = "20px";
   div.style.width = "89px";
   div.style.height = "90px";
   div.style.background = "url(Images/Tumb/1.png)";
document.getElementById("ContenedorImagenes").appendChild(div);
}
```

Para hacer lo mismo con un canvas se necesita el siguiente código, aunque no se realizará de esta forma por los inconvenientes al no tener barra de scroll este elemento.

```
function draw() {
   var ctx = document.getElementById('canvas').getContext('2d');
   var img = new Image();
   img.onload = function(){ctx.drawImage(img,0,0); };
   img.src = 'Images/Tumb/1.png';
 }
```

Para mostrar un recuadro de color al pasar por encima de un elemento y que este borde no mueva la imagen de sitio se necesita el siguiente arreglo:

En el css:

```
div.imgS img {
   border:3px solid #ffffff;
}
div.imgS a:hover img{
  border:3px solid #29ABE2;
}
```

En el html:

```
<div class="imgS" id="BasDat1">
   <a target="_blank" >
     <img id="PonImg1" src="Images/Tumb/1.png" onmouseup="GetSrc(this)"/>
   </a>
 </div>
```

Para cambiar la src de un elemento por otra concreta se empleó la siguiente función, aunque más adelante se mostrará una función más compleja que cambia la imagen por la siguiente:

```
function changeImage() {
      document.getElementById("img").src="image/3.png";
   }
```

Y el botón que llama a la función anterior:

```
<button type="button" onClick="changeImage();">Try it</button>
```

Para crear la función que cambia las previsualizaciones de las imágenes imitando al botón de pasar página se usa el siguiente código, que extrae los caracteres numéricos de la src y le suma 4. El contenido de la función se repite 5 veces para los ids PonImg1,2,3,4y5(el Auxiliar).

```
function changeImage() {
 var aux1 = document.getElementById("PonImg1").src;
 var numb1 = aux1.match(/\d/g);         //Esto lee todos los numeros, no olvidar quitar los espacios no deseados de ruta
 numb1 = numb1.join("");                //Une la cadena de numeros encontrados
 var i1 = 4 + parseInt(numb1);
   document.getElementById("PonImg1").src='Images/Tumb/'+i1+'.png';
}
```

Modificación de la función GetSrc de moverImaagen.js para que ponga la misma imagen de otra carpeta:

```
function GetSrc(f){
var aux = f.src;
var numb = aux.match(/\d/g);
numb = numb.join("");
rutmos = 'Images/Basdat/'+numb+'.png';
}
```

Para que cargue las imágenes correspondientes a la categoría seleccionada se necesita una función que lea el id de la imagen correspondiente a la categoría seleccionada y la asigne como parte de la ruta de las imágenes previsualizadas.

```
var id = "global variable";  //La variable debe ser global porque se requerirá en moverImagen.js también
function getId(ele) {
   id = ele.id;
}
```

En el html (se ha simplificado suprimiendo líneas que no afectan a lo explicado):

```
<img id="Animals" src="Images/bot1.png"
      onMouseDown="getId(this);iniBasdat()"
      onclick="CambiaClase(window.document.getElementById('CuadroBotones2'),'visible','oculta')"
      />
```

En el moverImagen habrá que modificar la función GetSrc así como las funciones de paginaprueba.js:

```
function GetSrc(f){
var aux = f.src;
var numb = aux.match(/\d/g);
numb = numb.join("");  //Une los numeros encontrados
rutmos = 'Images/Basdat/'+id+'/'+numb+'.png';
}
```

Para mostrar las previsualizaciones al seleccionar la categoría usando la función anterior habría que modificar el src concatenándolo con el id obtenido pero como esto no es sencillo se recurrió a crear una función que "inicia" (remplaza) todos los src al usarla..

```
function iniBasdat() {
   document.getElementById("PonImg1").src='Images/Tumb/'+id+'/'+'1.png';
 document.getElementById("PonImg2").src='Images/Tumb/'+id+'/'+'2.png';
 document.getElementById("PonImg3").src='Images/Tumb/'+id+'/'+'3.png';
 document.getElementById("PonImg4").src='Images/Tumb/'+id+'/'+'4.png';
document.getElementById("PonImg5").src='Images/Tumb/'+id+'/'+'5png'; //Div auxiliar para ver si hay mas fotos
}
```

Y para llamarla se ha modificado el html de la siguiente manera:

```
<div id="BotAnim">
  <img id="Animals" src="Images/bot1.png"
     onmouseover="MuestraImagen(this,2)"
     onmouseout="MuestraImagen(this,1)"
     onMouseDown="getId(this);iniBasdat()"
     onclick="CambiaClase(window.document.getElementById('CuadroBotones2'),'visible','oculta')"
     />
  </div>
```

Para mostrar y ocultar los botones de siguiente y anterior de la previsualización de imágenes se han usado las siguientes funciones:

La primera detecta cuando un div invisible que contiene la siguiente imagen a la última mostrada no contiene una imagen y oculta el botón siguiente gracias al evente onError del div invisible. Sirve para no dejar cargar divs sin contenido cuando ya no quedan más imágenes que mostrar en la carpeta:

```
function prueba() {
   //alert("hola");
 document.getElementById('NextBasDat').style.visibility = 'hidden';
}
```

Y en el html del div auxiliar (imagen siguiente a la mostrada):

```
<div id="BasDatAux">
   <img id="PonImg5" src="Images/Tumb/Animals/2.png" onError="prueba()"/>
  </div>
```

La segunda detetecta si el primer div tiene una imagen llamada 1 y oculta el botón anterior y muestra el de siguiente (el onerror del onclick se encargará de ocultarlo si no hace falta):

```
function PrevLoad(){
 var aux = document.getElementById("PonImg1").src;
 var numb = aux.match(/\d/g);            //Esta funcion lee todos los numeros
  if(numb==1)
  {document.getElementById('PrevBasDat').style.visibility = 'hidden';
   document.getElementById('NextBasDat').style.visibility = 'visible';}   else
   document.getElementById('PrevBasDat').style.visibility = 'visible';
}
```

La tercera muestra el botón siguiente:

```
function PrevM() {
 document.getElementById('NextBasDat').style.visibility = 'visible';
}
```

La última oculta todos los botones, necesario para cuando se quiera cambiar de categoría:

```
function Home() {
 document.getElementById('NextBasDat').style.visibility = 'hidden';
 document.getElementById('PrevBasDat').style.visibility = 'hidden';
}
```

Y para llamar a las funciones se llaman de la siguiente manera en el html (Aparte de llamar a la función PrevLoad() con el evento onMouseDown en cada uno de los botones de cada categoría):

```
<div id="PrevBasDat">
  <img src="Images/Prev.png"
      onmouseup="changeImage2();PrevLoad();PrevM()"
      />
  </div>
  <div id="NextBasDat">
  <img src="Images/Next.png"
      onmouseup="changeImage();PrevLoad()"
      />
  </div>
```

Para que la imagen que se sitúa en el canvas se pueda redimensionar, a parte del código anterior, hace falta una función que cree cuadros en las esquinas de la imagen (actuando como manejadores) y que en el evento onMouseMove detecte si se está seleccionando una zona de la imagen (y en su caso arrastre la imagen) o si se está seleccionando un manejador (y en su caso redimensione la imagen en función del manejador que se trate).

Para ello se siguió el tutorial de Simon Saarris (http://simonsarris.com/blog/225-canvas-selecting-resizing-shape), modificando su código libre para añadirle las imágenes a los cuadros.

Lo primero que se realizó fue renombrar la función init2() y la variable canvas2 para hacerlas coincidir con el html (init() y canvas1).

Después se añadió la variable rutmos, encargada de buscar la ruta de la imagen que se desea colocar en el canvas.

```
var rutmos = "Images/Tumb/Animals/1.png";
function GetSrc(f){
var aux = f.src;
var numb = aux.match(/\d/g);
numb = numb.join("");
rutmos = 'Images/Basdat/'+id+'/'+numb+'.png';
}
```

Acto seguido se añadió la src proporcionada por la variable rutmos a la función encargada de crear las cajas:

```
function Box2(x, y, w, h, imgPo) {
 imgPo = new Image();
  imgPo.src = rutmos;
  this.x = 0;
  this.y = 0;
  this.w = 1; // default width and height?
  this.h = 1;
  this.imgPo = imgPo || '1.png';
}
```

Por último se creo la función que crea la imagen seleccionada en el centro de la pantalla y se eliminaron funciones innecesarias como la creación inicial de cajas en el canvas del init() y el onDoubleClick(aunque este se dejó solo el nombre ya que sino no funcionaba):

```
function AdImageCanv() {
  addRect(275, 175, 150, 150);
}
```

Para llamar a la función anterior basta con añadir en el div de la imagen correspondiente del html:

```
onClick="AdImageCanv()
```

Por otra parte, la función que "elimina"(envía muy a la izquierda) la foto seleccionada:

```
function clearImg() {
 mySel.x = -1000; //Envia el elemento muy lejos
 invalidate();
}
```

Para que las imágenes de la base de datos se muestren en un contenedor y se pueda ”hacer scroll” sin tener que cargar todas las fotos al abrir la página web hace falta un código que cargue las imágenes poco a poco cuando se ha llegado a la última imagen mostrada.

Para ello se ha utilizado la biblioteca de javascript jQuery y más concretamente los ficheros *jquery.min.js* y *jquery.endless-scroll.js* modificado.

Lo primero que se definió fueron las imágenes que se cargan automáticamente al inicio de la página en el body:

```
<div class=”imgS”>
 <h3>Base de datos</h3>
 <ul id=”list”
  <p><a><img src=”img/1.png” onclick=”GetSrc(this);”></a> <a><img src=”img/2.png”onclick=”GetSrc(this);”></
    a></p>
  <p><a><img src=”img/3.png” onclick=”GetSrc(this);”></a> <a><img src=”img/4.png”onclick=”GetSrc(this);”></
    a></p>
  <p><a><img src=”img/5.png”onclick=”GetSrc(this);”></a> <a><img src=”img/6.png”onclick=”GetSrc(this);”></
    a></p>
 </ul>
</div>
```

Apoyado en el siguiente css:

```
/* Aquí se define el tamaño y color de fondo, junto con la visivilidad de la barra de scroll, del cuadro que contiene las
 imágenes */

ul#list { width: 250px; height: 200px; overflow-y: scroll; background-color:#FFF }


/* Para que se visualice el reborde azul al seleccionar */

div.imgS img {border:3px solid #ffffff;}

div.imgS a:hover img{border:3px solid #29ABE2;}
```

Una vez definida la lista inicial hay que añadir la función que carga más imágenes cuando el scroll llega al máximo:

```
<script type=”text/javascript” charset=”utf-8”>
  $(function() {
   $(‘ul#list’).endlessScroll({
    fireOnce: false,
          ceaseFire: function(i) { if (i==19) return true; },  //detiene el scroll infinito al llegar al elemento 19
    insertAfter: “ul#list div:last”,
    data: function(i) {
     return ‘<p><a><img src=”img/’ + i + ‘.png”onclick=”GetSrc(this);”></a> <a><img src=”img/’ + (i+1) +
     ‘.png”onclick=”GetSrc(this);”></a></p>’
    }
   });
  });
 </script>
```

Para terminar, hace falta modificar el código del fichero *jquery.endless-scroll.js* para adaptarlo a la cantidad de imágenes que se quieren mostrar.

```
.....
var options = $.extend({}, defaults, options),
      firing = true,
      fired = false,
      fireSequence = 5, //decir foto a partir de la que empieza a contar. En nuestro caso se cargan 6 imágenes al inicio
      didScroll = false,
      scrollTarget = this,
      inner_wrap = $(".endless_scroll_inner_wrap", this),
      is_scrollable;
.....
if (is_scrollable && (options.fireOnce == false || (options.fireOnce == true && fired != true))) {
        if (options.resetCounter.apply(scrollTarget) === true) {
          fireSequence = 0; //no se si hace algo, definir arriba
         }

        fired = true;
            fireSequence=fireSequence+2; //especificar aqui cuanto cambia la imagen cada linea. Nuestro caso 2 columnas

.....
```

Para mostrar dos cuadros diferentes en la misma páginas y que su scroll se comporte como en los ejemplos de scroll infinito (carga imágenes conforme se va bajando) se copió el script *jquery.endless-scroll2.js* y se modifico para evitar interferencias con el original.

La parte de código que se modificó fue al inicio:

```
(function($){
 $.fn.endlessScroll2 = function(options) {
 ...
}
```

Y en el correspondiente html se añadio la función:

```
//Funcion de los resultados
 $(function() {
    $('ul#list2').endlessScroll2({
     fireOnce: false,
            ceaseFire: function(i) { if (i==18) return true; },
     insertAfter: "ul#list2 div:last",
     data: function(i) {
      return '<p><a><img src="img/' + i + '.png"onclick="GetSrc(this);"></a> <a><img src="img/' + (i+1) +
      '.png"onclick="GetSrc(this);"></a></p>'
     }
    });
  });
```

Aparte, es muy importante recordar que hay que definir el numero de imágenes de la lista en el html en:

```
ceaseFire: function(i) { if (i==19) return true; },
```

Para filtrar las imágenes que se muestran como resultado de la selección de categorías y características se probó a organizar las imágenes en carpetas correspondientes a su categoría/estilo y realizar una función que detecte la categoría y características que se han seleccionado y busque en las carpetas. De este modo, si hay varias características marcadas busca el nombre de estas y selecciona solo las imágenes con el mismo nombre que se encuentran en las carpetas con el nombre de las características.

Para ello lo primero necesario es una función que añade una imagen (img) a un div (contenedor):

```
function anadir imagen(){
        var img = new Image();
        var div = document.getElementById('contenedor');

        img.onload = function()
                {
                        div.appendChild(img);
                };

        img.src = 'categorias/animales/1.png';
 }
```

Para detectar si una imagen determinada existe o no se uso la función que detecta si el elemento tiene altura o no:

```
var img2 = new Image();
      img2.src = 'categorias/animales/9.png';
function buscar(){
      if(img2.height == 0){
  alert("no hay imagen");
      }
}
```

Para añadir todas las imágenes de una carpeta se empleó la siguiente función:

```
 var numFotos=8   // Cantidad de fotos que tenemos
 var t=new Array(numFotos)   //
 function prueba(){
    for(i=1;i<=numFotos;i++){   // Precarga de imágenes
       zed='categorias/animales/'+i+'.png';
       anadir(zed);
 //t[i-1]=new Image()
 //t[i-1].src='categorias/animales/'+i+'.png'
 }
}
```

Y para añadir todas las imágenes comunes en dos o más carpetas manualmente:

```
var numFotos=1000   // Cantidad de fotos que tenemos
var t=new Array(numFotos)
  function prueba(){
     for(i=1;i<=numFotos;i++){   // Precarga de imágenes
        imgct1=new Image()
        imgca1=new Image()
        cat1='categorias/animales/'+i+'.png';
        car1='caracteristicas/color/'+i+'.png';
        imgct1.src=cat1;
        imgca1.src=car1;
        if(imgct1.height != 0 && imgca1.height != 0)
           anadir(cat1);
           //t[i-1]=new Image()
           //t[i-1].src='categorias/animales/'+i+'.png'
         }
  }
```

Para identificar la categoría seleccionada y crear un string con la variable que debe ser comparada con la condición para seleccionar la imagen (!=0) se uso la siguiente función:

```
var id = null
function getId(ele) {
   if (id == null)
   id = ele.id;
 else
 id = id + "&&" + ele.id;
}
```

Para realizar la búsqueda de categorías en función de las opciones que marca el usuario se uso un formulario con checkboxes y una función que para cada imagen busca si esta se encuentra en todas las carpetas de las categorías marcadas devolviendo si es así la imagen en cuestión.

La declaración del formulario en el html:

```
<form>
<input type="checkbox" id="caract1" name="caract1" value="Animal">Animal<br>
<input type="checkbox" id="caract2" name="caract2" value="Color">Color<br>
<input type="checkbox" id="caract3" name="caract3" value="Grueso">Grueso<br>
</form>
<input type="button" value="Buscar" onclick="SelCat()"/>
```

Y la función :

```
function SelCat(){
var imgCar=new Image;
var miDir;
var numCaract=3
aux=true;
for(nImag=1;nImag<=13;nImag++){
    for(i=1;i<=numCaract;i++){
            if (document.getElementById('caract'+i).checked){
                    if (i==1){
                            imgCar.src='caracteristicas/animales/'+nImag+'.png';
                            aux=aux&&(imgCar.height!=0);
                    }else if (i==2){
                            imgCar.src='caracteristicas/color/'+nImag+'.png';
                            aux=aux&&(imgCar.height!=0);
                    }else if (i==3){
                            imgCar.src='caracteristicas/grueso/'+nImag+'.png';
                            //alert(imgCar.src);//Hay que poner algo para que funcione bien
                            aux=aux&&(imgCar.height!=0);
                    }
            }
    }
    if(aux==true)
    anadir(imgCar.src);
    }
    alert('Busqueda con exito');//Para que funcione bien
}
```

Sin embargo el código no es interpretado correctamente en el ordenador y hace falta mostrar alert en determinadas partes dejando tiempo suficiente para completar la solicitud AJAX.

Para corregir el problema anterior al realizar la búsqueda de categorías se volvió al código anterior. Este se centra en la idea de carpetas con el nombre de las características que contienen las imágenes que cumplen esa característica. El código compara si cada imagen existe en todas las categorías y en caso afirmativo devuelve la imagen, inicialmente busca en la carpeta base de datos que contiene todas las imágenes y al ir seleccionando características va cambiando la ruta correspondiente.

Para las rutas de las características hace falta una variable global para cada una de las características:

```
var nomCar1='basedatos';
var nomCar2='basedatos';
var nomCar3='basedatos';
```

Para asignar la ruta correcta al seleccionar una categoría y cambiar el color del elemento hace falta la siguiente función:

```
function getId(ele) {
  id = ele.id;
    if(id=='cat1'){
            if (nomCar1=='animales'){
                    nomCar1='basedatos';
                    ele.style.color = "#4F4F4F";
            }else{
                    nomCar1='animales';
                    ele.style.color = "#29ABE2";}
    }else if (id=='car1'){
            if (nomCar2=='color'){
                    nomCar2='basedatos';
                    ele.style.color = "#4F4F4F";
            }else{
                    nomCar2='color';
                    ele.style.color = "#29ABE2";}
    }else if (id=='car2'){
            if (nomCar3=='grueso'){
                    nomCar3='basedatos';
                    ele.style.color = "#4F4F4F";
            }else{
                    nomCar3='grueso';
                    ele.style.color = "#29ABE2";}
    }
}
```

Con sus correspondiente botones de activación en el html:

```
<label id="cat1" type="button" onClick="getId(this); changeColor(this.id)">Animal</label><br/>
<label id="car1" type="button" onClick="getId(this)">Color</label><br/>
<label id="car2" type="button" onClick="getId(this)">Grueso</label><br/>
<button type="button" onClick="prueba()">Buscar</button>
```

Para añadir elementos al contenedor se usa la siguiente función:

```
function anadir(elem){
        var img2 = new Image();
        var div = document.getElementById('contenedor');
        img2.onload = function() {
                div.appendChild(img2);
        };
        img2.src = elem;}
```

Y para eliminarlos:

```
function borrar(){
    var div = document.getElementById('contenedor');
    //div.parentNode.removeChild(div2);
    div.innerHTML = "";
}
```

Y la función que busca las imágenes que cumplen los criterios de búsqueda:

```
// Funcion que anade fotos de categoria
var numFotos=40   // Cantidad de fotos que tenemos
function prueba(){
    borrar();
     for(i=1;i<=numFotos;i++){   // Precarga de imágenes

     imgct1=new Image()
     imgct1.src='caracteristicas/'+nomCar1+'/'+i+'.png';
     imgca1=new Image()
     car1='caracteristicas/'+nomCar2+'/'+i+'.png';
     imgca1.src=car1;
     imgca2=new Image()
     car2='caracteristicas/'+nomCar3+'/'+i+'.png';
     imgca2.src=car2;

    var filtros=imgca1.height && imgct1.height && imgca2.height;
    if(filtros != 0)
    anadir(car1);
    }
 }
```

Por último, en el contenedor de las imágenes hay que especificar que muestre el scroll:

```
<div id="contenedor" style="height:500px; overflow-y: scroll;">
</div>
```

Para corregir el código anterior que no cargaba inmediatamente las fotos la mayoría de las veces, se creo una función que carga las imágenes de la base de datos al seleccionar una categoría.

Para las rutas de las características hace falta una variable global para cada una de las características:

```
function precarga(){
      var numFotos=40   // Cantidad de fotos que tenemos
  var t=new Array(numFotos)   //
  for(i=1;i<=numFotos;i++){   // Precarga de imágenes
    t[i-1]=new Image()
      t[i-1].src='caracteristicas/'+nomCarp+'/'+i+'.png'}
}
```

Habiendo definido la variable global:

```
var nomCarp='basedatos';
```

Y modificando la función que seleccionaba la categoría:

```
function getId(ele) {
   id = ele.id;
      if(id=='cat1'){
                  if (nomCar1=='animales'){
                              nomCar1='basedatos';
                              ele.style.color = "#4F4F4F";
                  }else{
                              nomCar1='animales';
                              nomCarp='animales';
                              precarga();
                              ele.style.color = "#29ABE2";}
      }else if (id=='car1'){
      .....................
```

También se re-hizo el código que analiza si una imagen existe o no, reescribiéndolo mejor:

```
//Funcion que comprueba si una imagen existe o no
function checkImage(src) {
  var img = new Image();
  img.onload = function() {
    alert("existe imagen");// code to set the src on success
  };
  img.onerror = function() {
    alert("no existe imagen");// doesn't exist or error loading
  };
  img.src = src; // fires off loading of image
}
```

Por último, se hizo una función que devuelve la última foto de una carpeta suponiendo que la primera se llama 1.png y las demás van consecutivas, sin saltarse un número, pero necesita de alerts para su correcto funcionamiento y no es útil para el código por ahora:

```
Funcion que analiza si existe o no una imagen
function checkImage(src) {
  var img = new Image();
  img.onload = function() {
    ExisteImagen=true;
  };
  img.onerror = function() {
    ExisteImagen=false;
      //numFotos=5;//Para que funcione el bucle for
  };
  img.src = src; // fires off loading of image
}



//Funcion que devuelve la ultima carpeta que existe ne la carpeta
function contar(){
      var src;
      for(i=1;ExisteImagen!=false;i++){
                  imgAux.src='caracteristicas/'+nomCarp+'/'+i+'.png';
                  src=imgAux.src;
                  checkImage(src);
                  //alert(ExisteImagen);
                  alert(src);//Si no no funciona
      }
      alert(i-2);
}
```

Para centrar todos los elementos tanto horizontalmente como verticalmente (este último era lo dificil) hizo falta un div contenedor con unas diensiones fijas (sin definirlo como position:absolute?) y el siguiente código:

En el CSS:

```
html, body {
   margin: 0;
   padding: 0;
   height: 100%;
   }

#container {
   display: table;
   height: 100%;
   width: 100%;
   margin: 0;
   }

#content {
   display: table-cell;
   vertical-align: middle;
   position: relative;
   }

#inner {
   border: 1px dashed #F00;
   /* width: 50%;*/
   width: 300px;
   margin: 0 auto;
   padding: 0 20px;
   }

* html #content {
   top: 50%;
   left: 0;
   height: 1px;
   }

* html #content #inner {
   position: relative;
   top: -50%;
   }
```

Y en el body del html:

```
<div id="container">
  <div id="content">
    <div id="inner">

      (Contenido)

    </div><!-- fin inner -->
  </div><!-- fin content" -->
</div>
```

Para organizar las capas en el código sin usar el atributo position:absolute (que tendría que declararse en todos los divs padres) se puede realizar la siguiente forma de organizar, teniendo en cuenta que los divs los coloca uno debajo de otro y con el atributo float:right/left podemos centrar en un lado del contenedor el div:

En el CSS:

```
#contenedor {
    background-color: #666;
    height:600px;
    width: 1024px;
    z-index: 1;
}

#BaseDatos {
    height:100px;
    width:100px;
    background-color:#FFF;
    z-index:1;
}

#ContenedorResultados {
    height:300px;
    width:100px;
    background-color: #06C;
    z-index:1;
}

#Resultados {
    float:right;
    height:300px;
    width:50px;
    background-color: #0F0;
    z-index:1;
}
```

Y en el body del html:

```
<div id="contenedor">
    <div id="BaseDatos"></div>
    <div id="ContenedorResultados">
            <div id="Resultados"></div>
    </div>
</div>
```

Para corregir el problema de las tildes, que no se mostraban correctamente, se modificó la encriptación añadiendo en el head una línea de código.

Para las rutas de las características hace falta una variable global para cada una de las características:

```
<meta charset="UTF-8">
```

Para añadir la funcionalidad de poder seleccionar la categoría mediante una lista se añadió en el html el siguiente formulario:

```
<form id="Formul" >
    <select id="SelCat" onchange="checkForm()">
        <option value="n0">Sin Categoría</option>
        <option value="n1">Animales</option>
        ...
                    </select>
            </form>
```

Acompañado de la correspondiente función para interpretar los resultados (además de actualizar la función prueba declarando la nueva variable Cat e iniciándola con el valor de base de datos globalmente):

```
function checkForm() {
    var f = document.getElementById('Formul');
    var s = document.getElementById('SelCat');
    if( s.selectedIndex == 0 ) {
            nomCat='basedatos';
            nomCarp='basedatos';
            precarga();
    }if( s.selectedIndex == 1 ) {
            nomCat='animales';
            nomCarp='animales';
            precarga();
    }if( s.selectedIndex == 2 ) {
            alert("2");
    }
}
```

También se hizo una función que detecta si un botón está o no presionado (leyendo el nº del id y comprobando si la característica correspondiente vale base de datos(no seleccionado) o no (seleccionado) y remplaza su color al pasar por encima:

```
function ColorSel(elem){
    var aux=elem.id;
    var Num = aux.match(/\d/g);          //Esta funcion lee todos los numeros
    Num = Num.join("");                          //No olvidar quitar espacios
    Cond=eval("nomCar"+Num);
    if (Cond=='basedatos')
            elem.style.color = "#404040";
}
```

Y en el html, para cada característica:

```
<label id="Car1" type="button" onClick="getId(this)" onMouseOver="ColorSel(this)" onMouseOut="ColorNorm(this)">Línea fina</label><br/>
```

Para crear una función que muestra el numero de una imagen del cuadro de resultados se creó la siguiente función:

```
//Funcion que añade imagenes a un div
function anadir(elem){
	var img2 = new Image();
	var div = document.getElementById('contenedor');
	img2.onload = function() {
		div.appendChild(img2);
	};
	img2.src = elem;
	//Y para que muestre el numero de imagen clicado
	img2.onclick = function(){
		var aux=img2.src;
		var Num = aux.match(/\d/g);
		Num = Num.join("");
		Cond=eval(Num);
		alert(Cond);
	};
}
```

Y para aplicar la anterior función a la base de datos antigua para que al hacer click en una imagen la añada al canvas, además de juntar el .js moverImagen en un solo script, se necesita la siguiente función:

```
//Funcion que añade imagenes a un div
function anadir(elem){
	var img2 = new Image();
	var div = document.getElementById('contenedor');
	img2.onload = function() {
		div.appendChild(img2);
	};
	img2.src = elem;
	//Y para que muestre el numero de imagen clicado
	img2.onmouseup = function(){
		var aux=img2.src;
		var Num = aux.match(/\d/g);
		Num = Num.join("");
		rutmos = 'Images/BaseDatos/'+Num+'.png';
	};
	img2.onclick = function(){
		addRect(275, 175, 150, 150);
	};
}
```

Y para crear la capa con opacidad que oscurece en fondo para que resalte la aplicación se requiere el siguiente css:

```
#Oscuro {
  top:0px;
    bottom:0px;
    right:0px;
    left:0px;
  position: absolute;
    z-index:2;
    background-color: #000;
    opacity: 0.8;
}
```

Y para ubicar el div, va dentro del div container y antes del div content tal y como se indica a continuación:

```
<div id="container" class="oculta">
    <div id="Oscuro"></div>
    <div id="content">
            <div id="inner">
                    <div id="ContUniv">
                            <div id="ContenedorBaseDatos">
                                    <div id="CabeceraBaseDatos">
                                            ……..
```

Para crear la plantilla de la página web se quiso añadir un degradado a los bordes del contenedor fijo de texto, para ello se creo el contenedor:

```
#DifDer {
 position:absolute;
 top:0px;
 bottom:0px;
 right:-8px;
 width:8px;
 background-image: url(Images/Diseno/difum.png);
 z-index:1;
}
```

Y se situó en la posición correcta del html:

```
<body>
<div id="container-page">
<div>
<div id="DifDer"></div>
<div id="DifIz"></div>
 <div id="container-head">
  <div id="head"><a href="./index.html"><img src="Images/Diseno/Logo.png"></a>
   <div id="content-sec">
    ..............
```

Para cambiar las tipografías del documento por una descargada, se convirtió mediante un programa a los formatos adecuados y se uso el siguiente código en el css para declararla:

```
@font-face {
     font-family: 'Conv_GeosansLight';
     src: url('fonts/GeosansLight.eot');
     src: local('☺'), url('fonts/GeosansLight.woff') format('woff'), url('fonts/GeosansLight.ttf') format('truetype'),
     url('fonts/GeosansLight.svg') format('svg');
     font-weight: normal;
     font-style: normal;
}
```

Y para aplicarla a un div, en el css:

```
body {
 color: #333;
 font-family:'Conv_GeosansLight',Sans-Serif;
 font: 12px;
 text-align: center;
 background-color:#EFEFEF;
}
```

Para añadir las imágenes adecuadas (una vez seleccionados los filtros) al cuadro de resultados y que estas puedan seleccionarse (quedándose marcadas en un borde azul), muestren un mensaje informativo al pasar por encima, y se puedan añadir al canvas (con dobleclick) se necesita el siguiente código:

```
//Funcion que añade imagenes a un div
function anadir(elem){
	var img2 = new Image();
	var div = document.getElementById('contenedor');
	img2.onload = function() {
			div.appendChild(img2);
	};
	img2.src = elem;
	img2.className='imgNorR';
	img2.title='Click para seleccioonar, doble click para añadir';
	//Y para que muestre el numero de imagen clicado
	img2.onclick = function(){
			if(img2.className=='imgNorR')
	    img2.className='imgSelR'
	else
	    img2.className='imgNorR'
	};
	img2.ondblclick = function(){
			var aux=img2.src;
			var Num = aux.match(/\d/g);
			Num = Num.join("");
			Cond=eval(Num);
			alert(Cond);
	};
}
```

Además de la siguiente función que cambia la clase de un elemento (para mostrar o no el borde de color):

```
// Funcion que cambia la clase de un elemento por otra
function CambiaClase(elemento,clase1,clase2){

  if(elemento.className==clase1)
	elemento.className=clase2
  else
	elemento.className=clase1

}
```

Y la correspondiente definición de las clases:

```
/* Para las imagenes de caracteristicas */
.imgNorR {
	border:3px solid #FFF;
	margin-left:15px;
	margin-top:18px;
	margin-bottom:0px;
	margin-right:5px;
}
.imgSelR {
	margin-left:15px;
	margin-top:18px;
	margin-bottom:0px;
	margin-right:5px;
	border:3px solid #29ABE2;}
```

Para añadir las imágenes al cuadro de resultados y que estas puedan seleccionarse (quedándose marcadas en un borde azul), muestren un mensaje informativo al pasar por encima, y se puedan añadir al canvas (con dobleclick) se necesita el siguiente código en el script del html, declarando la clase como una variable para que no cree interpretaciones erróneas en el return:

```
<script type="text/javascript" charset="utf-8">
//Funcion de la base de datos / scroll infinito
   var imgNor='imgNor';
var imgSel='imgSel';
   $(function() {
    $('ul#list').endlessScroll({
     fireOnce: false,
                ceaseFire: function(i) { if (i>=70) return true; },
     insertAfter: "ul#list div:last",
     data: function(i) {
      return '<p><a><img class="imgNor" src="Images/Tumb/caracteristicas/basedatoso/' + i + '.png" title="Click
     para seleccionar, doble click para añadir" onmouseup="GetSrc(this)" onclick="CambiaClase(this,imgNor,imgSel);"
     onDblClick="AdImageCanv();"></a>
     .....
```

Y para mostrar la aplicación de búsqueda de imágenes al presionar un botón y que esta aparezca en el centro de la pantalla y se oscurezca el resto hace falta organizar los divs de la siguiente manera:

```
<body onload="init();iniBasdat()">

<div id="ContPagUniv"><div id="container" class="oculta">
     <div id="Oscuro"></div>
     <div id="content">
                <div id="inner">
                     .....
                </div><!-- fin inner -->
     </div><!-- fin content" -->
</div></div>

<div id="container-page">

<div>
<div id="Iconos"></div>
<div id="DifDer"></div>
<div id="DifIz"></div>
 <div id="container-head">
  .......
 </div>
 <div id="container-content">
  <div id="content-pri">
   <div id="ContCanvas">
                <canvas id="canvas1" width="630" height="400">
                </canvas>
                <div id="CuadroBotones2" class="visible">
                     .........
                </div>
      </div>
  </div>
 </div>
</div>
 <div id="container-foot">
  ..........
 </div>
</div>

</body>
```

Para hacer aparecer y desaparecer un elemento durante un tiempo determinado se usaron las siguiente funciones:

```
//Funciones que hacen aparecer y desaparecer un elemento durante X tiempo.
function toggleOpacity(id) {
   var el = document.getElementById(id);
      el.style.visibility = 'visible';
                  fadeIn(el, 150);
                  //Para desvanecerce cuando haya aparecido y no antes
                  setTimeout(function () {fadeObject(el, 1, 0, 150);},500);
                  setTimeout(function () {el.style.visibility = 'hidden';},650);
}

function fadeIn(el, duration) {
   fadeObject(el, 0, 1, duration);
}

function fadeOut(el, duration) {
   fadeObject(el, 1, 0, duration);
}

function fadeObject(el, start, end, duration) {
   var range = end - start;
   var goingUp = end > start;
   var steps = duration / 20;   // arbitrarily picked 20ms for each step
   var increment = range / steps;
   var current = start;
   var more = true;
   function next() {
      current = current + increment;
      if (goingUp) {
         if (current >= end) {
            current = end;
            more = false;
         }
      } else {
         if (current <= end) {
            current = end;
            more = false;
         }
      }
      el.style.opacity = current;
      if (more) {
         setTimeout(next, 20);
      }
   }
   next();
}
```

Y para detectar si el navegador que abre la página es Chrome se uso el siguiente código capaz de detectar los navegadores más usados actualmente y que escribe en el body el nombre.

```
<body>

   <script language="javascript" type="text/javascript">
//Detect browser and write the corresponding name
if (navigator.userAgent.search("MSIE") >= 0){
   document.write('MS Internet Explorer ');
}
else if (navigator.userAgent.search("Chrome") >= 0){
document.write('Google Chrome ');// For some reason in the browser identification Chrome contains the word "Safari" so when detecting for Safari you need to include Not Chrome
}
else if (navigator.userAgent.search("Firefox") >= 0){
   document.write('Mozilla Firefox ');
}
else if (navigator.userAgent.search("Safari") >= 0 && navigator.userAgent.search("Chrome") < 0){//<< Here
   document.write('Apple Safari ');
}
else if (navigator.userAgent.search("Opera") >= 0){
   document.write('"Opera ');
}
else{
   document.write('"Other"');
}
   </script>

</body>
```

Para añadir y eliminar elementos en el cuerpo de la página se uso la función .before() de jquery y el siguiente código:

Para añadir las dos cajas (nombre elemento y botón cerrar) antes del elemento con id Anadir:

```
function anadirDelante ()
{
    //i=i+1;
    $("#Anadir").before('<div id="'+i+'bu" style="width:150px; height:31px;"><div id="'+i+'a" style=" float:left;
    width:100px; height:31px;">'+caract+'</div><div id="'+i+'b" onclick="restar1();asigID(this);eliminarP()"
    style=" float:right; width:35px; height:31px; background-image:url(Close.png)"></div></div>');
}
```

Y para eliminar el elemento seleccionado:

```
function eliminarP ()
{
    $("div").remove("#"+j+"");
}
```

Todo esto junto con las siguientes funciones que se encargan de llamar correctamente al nuevo id y recoger la información del id para borrar el seleccionado:

```
var i = 1;
var j = 1;
var caract = "caract1";

function asigID(ele)
{
    j=ele.id+"u";
    //alert (j);
}

function asigNom(ele)
{
    caract=ele.id;
    //alert (caract);
}

function sumar1()
{
    i=i+1;
}

function restar1()
{
    i=i-1;
}
```

Por último, se añadieron funciones para el desvanecimiento y cambiar la visibilidad de los elementos que ya antes se habían comentado.

Para obligar a que el usuario solo pueda seleccionar una imagen de la base de datos por defecto al hacer click se realizó el siguiente código, que comprueba si la búsqueda está restringida o es libre y reinicia todas las imágenes o no en función de esta:

La definición de la variable debe ser global:

```
var simpleSearch=true;
```

Para cambiar la clase de las imágenes, y con ello su color de borde en función de la variable anterior:

```
// Funcion que cambia la clase de un elemento por otra
function CambiaClaseImg(elemento,clase1,clase2)
{
    var elems = document.getElementsByClassName('imgSel');

    if(simpleSearch == false){
            if(elemento.className==clase1){
                    elemento.className=clase2
            }else{
                    elemento.className=clase1
            }
    }else{

            for(var i = 0; i < elems.length; i=i) {//Si se pone i++ salta imagenes no se por que
                    elems[i].className=imgNor
            }
            elemento.className=imgSel
    }
}
```

Y para cambiar de búsqueda simple a múltiple y a su vez reiniciar las imágenes que pudieran estar seleccionadas:

```
//Función que cambia de busqueda simple a múltiple
function BusqSimpl() {
    var elems = document.getElementsByClassName('imgSel');
    for(var i = 0; i < elems.length; i=i) {//Si se pone i++ salta imagenes no se por que
            elems[i].className=imgNor
    }
    if (simpleSearch ==false){
            simpleSearch = true;
    }else{
            simpleSearch = false;
    }
    alert ("Necesita este mensaje para que le de tiempo a interpretarlo");
}
```

Para crear dos divs contenedores que se puedan redimensionar de manera que al hacer uno más grande el otro se haga más pequeño correctamente y viceversa se utilizó el siguiente código en el cuerpo:

```
<script>
  var myheight  = ($(window).height()-50);//Pie de pagina
  var mywidth  = $(window).width();
  $(window).load(function () {
    $("#resizable").height(220); //Altura por defecto del cuadro de arriba, puse 220
    $("#content").height(myheight-$("#resizable").height());
            $( "#accordion" ).accordion( "refresh" );
  });
</script>
<div id="resizable" style="border:1px solid #333; overflow:auto">resizable</div>
<div id="content" style="border:1px solid red; overflow:auto">content</div>
<script>
  $("#resizable").resizable({
    handles: 's',
    maxHeight: myheight,
    resize: function() {
    $("#content").height(myheight-$("#resizable").height());
            $( "#accordion" ).accordion( "refresh" );//Para actualizar el menu desplegable
    }
  });
    $("#accordion-resizer").resizable({
    handles: 's',
    maxHeight: myheight,
    resize: function() {
    $("#content").height(myheight-$("#accordion-resizer").height());
            $( "#accordion" ).accordion( "refresh" );//Para actualizar el menu desplegable
    }
  });
</script>
</body>
```

Y llamando a los siguientes divs en el head:

```
<link rel="stylesheet" href="js/jquery.ui.all.css">
    <script src="js/jquery-1.9.1.js"></script>
    <script src="js/ui/jquery.ui.core.js"></script>
    <script src="js/ui/jquery.ui.widget.js"></script>
    <script src="js/ui/jquery.ui.mouse.js"></script>
    <script src="js/ui/jquery.ui.resizable.js"></script>
  <script src="js/ui/jquery.ui.accordion.js"></script>
    <link rel="stylesheet" href="js/demos.css">
```

Y para ajustar el número de divs a la pantalla se usó la siguiente modificación en la función de WebAppStandAlone.js:

```
var mywidth  = ($(window).width()-50);
var tamanoImg = 100;//88 mas margenes
var NELEMTS_PER_ROW=Math.floor(mywidth/tamanoImg);

var NROWS_INITIAL=5;
```

Para crear un div escondido que al presionar un botón se despliega con una animación de desplazamiento se uso el siguiente código:

En el html se definen el div (mas el boton):

```
<div class="pollSlider">
    <div>
        ........
    </div>
</div>
```

En el css se define la posición y margen para ocultarlo y colocarlo en la posición inicial:

```
.pollSlider{
    position:absolute;
    bottom:100px;
    height:250px;
    background-color:#FFF;
    width:249px;
    left:0px;
    margin-left: -249px;
    z-index:3;
}
```

Y en el .js se escribe la función que llamará a la animación, con una variable que comprueba si el botón esta presionado o no:

```
//Funcion que muestra el desplegable de AddFeature
var IsPres1=false;
function AddFeatYN() {
  if(IsPres1 ==true)
  {
    $('.pollSlider').animate({"margin-left": '-=249'});
            IsPres1=false;
    //$('#pollSlider-button').animate({"margin-right": '-=200'});
  }
  else
  {
    $('.pollSlider').animate({"margin-left": '+=249'});
            IsPres1=true;
    //$('#pollSlider-button').animate({"margin-right": '+=200'});
  }
}
```

Para hacer un formulario con opciones no seleccionables se uso el siguiente código (la clase solo la lee mozilla pero contenía (option.select-hr { border-bottom: 1px dotted #000; }):

```
<form id="FormAddFeat">

        <select id="SelAddFeat" onchange="checkFormAddFeat();anadirDelante(); sumar1();ReseteoFormAddFeat()">
            <option selected val="a" class="select-hr" disabled="disabled">Add Feature</option>
          <option class="select-hr" disabled="disabled">Stroke</option>
          <option value="Car1">Thin Line</option>
          <option value="Car2">Thick Line</option>
          <option value="Car3">Irregular Line</option>
          <option value="Car4">No Contour Line</option>
          .....
           </select>
      </form>
```

Para asignar la característica correspondiente a cada opción del formulario de addfeature:

```
function checkFormAddFeat() {
    var f = document.getElementById('FormAddFeat');
    var s = document.getElementById('SelAddFeat');
    if( s.selectedIndex == 2 ) {
            caract="Thin Line";
    }else if( s.selectedIndex == 3 ) {
            caract="Thick Line";
    .....................
    }else{
            caract="Filling Out";
    }
}
```

Y para los colores de los labels(1-18) al pasar por encima o hacer click en ellos:

```
function ColorSelSimple(elem){
    if (elem.className=="TextNor"){
            elem.style.color = "#076F93";
    }else{
            elem.style.color = "#1188BC";
    }
}
function ColorNormSimple(elem){
    if (elem.className=="TextNor"){
            elem.style.color = "#101010";
    }else{
            elem.style.color = "#28ABE2";
    }
}
function PressLabel(elem){
    if (elem.className=="TextNor"){
            elem.className="TextSel";
            elem.style.color = "#28ABE2";
    }else{
            elem.style.color = "#101010";
            elem.className="TextNor";
    }
}
```

Y para el desplegable al presionar AddFeature que añade la categoría seleccionada al menú:

En el js:

```
< //Funciones para hacer aparecer o desaparecer el cuadro gris de Add Feature
function Aparecer(id) {
   var el = document.getElementById(id);
      fadeObject(el, 0, 0.4, 200);
}

function Desaparecer(id) {
   var el = document.getElementById(id);
      fadeObject(el, 0.4, 0, 200);
}

var FeatI = 1;
var FeatJ = 1;
var caract = "caract1";

function asigID(ele)
{
      FeatJ=ele.id+"u";
      //alert (j);
}

function asigNom(ele)
{
      if (ele.id == "Car1"){
                  caract="Thin Line";
                  //alert (caract);
      }else if (ele.id == "Car2"){
                  caract="Thick Line";
      }else if (ele.id == "Car3"){
                  caract="Irregular Line";
      }else if (ele.id == "Car4"){
                  caract="No Contour Line";
      }else if (ele.id == "Car5"){
                  caract="Black and White";
      }else if (ele.id == "Car6"){
                  caract="Many colors";
      }else if (ele.id == "Car7"){
                  caract="Few Colors";
      }else if (ele.id == "Car8"){
                  caract="Hard Shading";
      }else if (ele.id == "Car9"){
                  caract="Hard Transition";
      }else if (ele.id == "Car10"){
                  caract="Smooth Transition";
      }else if (ele.id == "Car11"){
                  caract="3D Aspect";
      }else if (ele.id == "Car12"){
                  caract="Flat Aspect";
      }else if (ele.id == "Car13"){
                  caract="Complex Shape";
      }else if (ele.id == "Car14"){
```

(Continua a la derecha)

(Continua aquí)

```
                  caract="Simple Shape";
      }else{
                  caract="Filling Out";
      }
}

function sumar1()
{
      FeatI=FeatI+1;
}

function restar1()
{
      FeatI=FeatI-1;
}

function anadirDelante ()
{
      //i=i+1;
      $("#pollSlider-button").before('<div id="'+FeatI+'bu"
style="width:130px; height:31px;"><div id="'+FeatI+'a"
style="padding-top:4px; float:left; width:100px;
height:31px;">'+caract+'</div><div id="'+FeatI+'b"
onclick="restar1();asigID(this);eliminarP()" style=" float:right;
width:20px; height:20px; background-image:url(Images/CSS/Close.
png)"></div></div>');
}

function eliminarP ()
{
      $("div").remove("#"+FeatJ+"");
}
```

Para añadir el botón que enciende y apaga el autoplay del slider de los planetas:

En en css se declara el div:

```
/*Boton que enciende y apaga el autoplay*/
#dg-container div.dg-aut{
	position:absolute;
	top:450px;
	left:440px;/*465-width/2*/
	width:30px;
	height:30px;
	background-color:#F00;
	z-index:2;
}
```

En el .js se declara el botón:

```
this.$navAut		= this.$el.find('.dg-aut'); //Boton que para o enciende el autoplay
```

Y también en el .js se crea la función en el correcto lugar (donde los botones de prev y next):

```
//Aqui empieza mi codigo para cambiar el autoplay
				this.$navAut.on( 'click.gallery', function( event ) {
					if( _self.options.autoplay == false) {
						//clearTimeout( _self.slideshow );
						_self.options.autoplay		= true;
						_self._startSlideshow();
					}
					else{
						clearTimeout( _self.slideshow );
						_self.options.autoplay		=false;
					}
				});
//Aqui termina mi codigo para cambiar el autoplay
```

Para redimensionar la aplicación cuando el usuario redimensiona la pantalla del navegador se usó la siguiente función:

Al principio del html:

```
//Para redimensionar la aplicacion correctamente al redimensionar la ventana
$(window).resize(function() {
                myheight  = ($(window).height()-160);
                $("#ContenedorBaseDatos2").height(myheight-$("#resizable").height());
                //$( "#accordion" ).accordion( "refresh" );
    });
```

Aunque no se usaron se muestran las siguientes funciones con comportamientos parecidos que se barajaron:

```
//Realiza la funcion de los comentarios al redimensionar por primera vez
$(window).one("resize", function () { /* */ });

//Realizar una funcion al redimensionar
$(window).resize(function() { ... });

//Realizar funcion al terminar de redimensionar
var waitForFinalEvent = (function () {
  var timers = {};
  return function (callback, ms, uniqueId) {
    if (!uniqueId) {
      uniqueId = "Don't call this twice without a uniqueId";
    }
    if (timers[uniqueId]) {
      clearTimeout (timers[uniqueId]);
    }
    timers[uniqueId] = setTimeout(callback, ms);
  };
})();

$(window).resize(function () {
    waitForFinalEvent(function(){
      alert('Resize...');
      //...
    }, 500, "some unique string");
});
```

Apéndice H

Contenido multimedia

El CD con nombre Contenido Multimedia que se encuentra en la contraportada de este anexo contiene:

- SitioWeb: Fichero comprimido con el contenido de la página web.
- Vídeo: Vídeo realizado para explicar la aplicación.

Apéndice I

Planificación

En la Figura 5 se muestra el resumen de la evolución temporal del proyecto, desde su comienzo a principios de enero hasta su finalización en septiembre. A continuación se describen las fases principales:

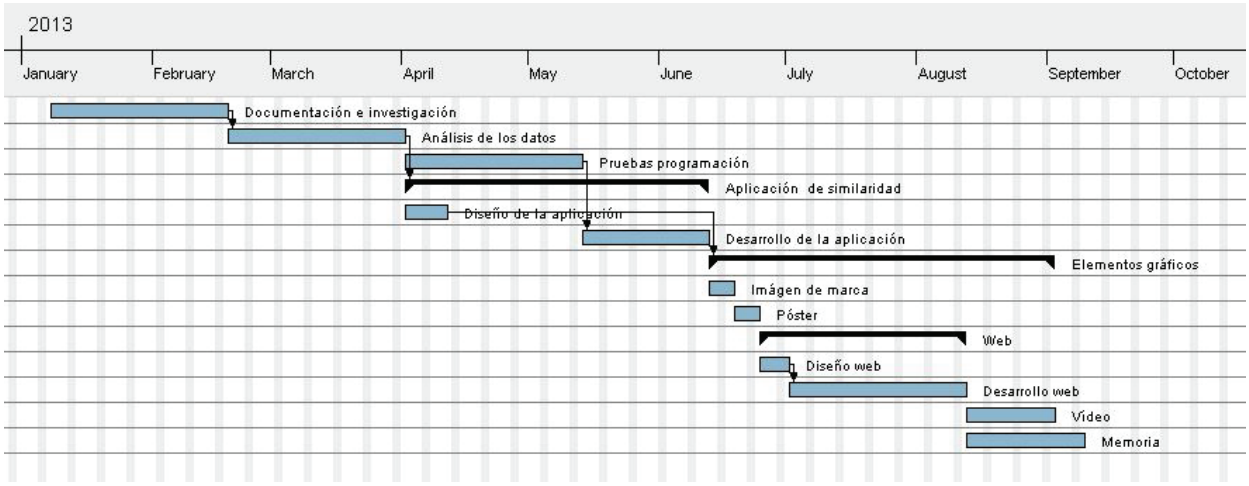


Figura 5: *Diagrama Gantt con la evolución temporal del proyecto.*

- **Documentación e investigación.** Una vez definido el proyecto, tuvo lugar una periodo de intensa documentación e investigación sobre aplicaciones existentes de búsqueda mediante similaridad visual y principios de diseño gráfico.
- **Análisis de los datos.** Después de analizar el mercado y los principios gráficos se analizaron todos los datos y se seleccionó los que se consideró adecuados para realizar pruebas con usuarios y la aplicación.
- **Pruebas programación.** Antes de comenzar la programación de la aplicación de búsqueda de imágenes similares y la web se realizaron pruebas con html y javascript para comprobar las limitaciones existentes.
- **Aplicación de similaridad.** En esta fase se desarrollaron las funcionalidades de la aplicación y se creó la interfaz de esta.
- **Elementos gráficos.** Esta parte es la más amplia del proyecto y en ella se crearon todos los elementos gráficos para la promoción de la aplicación.

    1. Imagen de marca. En esta fase se creó el imagotipo de la aplicación así como su manual de marca.

    2. Póster. En esta fase se desarrolló el póster de la aplicación.

3. Web. En esta fase de diseñó y desarrolló el sitio web de la aplicación, incluyendo el prototipo de búsqueda de imágenes similares de la aplicación.

4. Vídeo. En esta fase se creó el vídeo de promoción de la aplicación.

- **Memoria.** En este último periodo coincidente con el desarrollo del vídeo se redactó la memoria y el anexo.