

Práctica Profesional I

Código del curso: CC-4901

Informe de Práctica Profesional I

Web Intelligence Centre

Nombre: Gabriel Iturra Bocaz
Carrera: Ingeniería Civil en Computación
Rut: 18.214.940-3
Correo: gabrieliturrab@ug.uchile.cl
Celular: +569-51945376
Profesor: Alexandre Bergel
Fecha de realización: 3 al 31 de Enero de 2017
Fecha de entrega: 1 de Septiembre de 2017
Santiago, Chile

Resumen

El trabajo fue realizado en el centro de investigación Web Intelligence Centre (WIC). Este consistió en el desarrollo del mapa de objetos claves del *Proyecto AKORI*.

El *Proyecto AKORI*, cuyo sigla en inglés es *Advanced Kernel for Ocular Research and web Intelligence*, tiene como objetivo desarrollar una herramienta funcional que permita extrapolar el comportamiento de navegación para usuarios en un sitio web a partir de la extracción de patrones desde datos analizados con técnicas de minería de datos, eye tracking y electroencefalografía.

El mapa de objetos claves realizado tiene la meta de mostrar de forma clara y precisa los objetos más relevantes del *DOM* (*Document Object Model*) cuando un usuario navega e ingresa a un sitio web. Con el objetivo de tener un indicador de los principales objetos que un usuario visualiza en los primeros milisegundos de ingresar a un sitio Web, facilitando así el trabajo de los diseñadores y desarrolladores que desean sitios Web más visitados y atractivos. El mapa de objetos claves fue realizado tres etapas, migración, refactoring y *FrontEnd*, siendo esencial las etapa de migración y refactoring para el desarrollo *FrontEnd*.

El trabajo realizado se encuentra actualmente entregado a la aplicación web oficial del *Proyecto Akori* que se encuentra disponible en el sitio web.

En cuanto a aprendizajes obtenidos en el ámbito técnico se destaca el conocimiento adquirido del framework para desarrollo web *Django*, el lenguaje script *JavaScript* utilizado principalmente por el lado del cliente, y sistema de control de versiones Git. En cuanto otra habilidades aprendidas se destaca el trabajo en un equipo multidisciplinario.

Índice General

1. Introducción	1
1.1. Lugar de Trabajo	1
1.2. Grupo de trabajo	1
1.3. Equipos y Software	1
1.3.1. Software	1
1.3.2. <i>Proyecto AKORI</i>	2
1.3.3. Equipo	3
1.4. Situación Previa	4
1.4.1. Versión <i>Java</i> con <i>Servlet</i>	4
1.4.2. Versión <i>Django</i>	5
1.5. Descripción general del trabajo realizado	5
2. Trabajo Realizado	6
2.1. Etapa 0	6
2.1.1. Algoritmo Mapa de Objetos versión <i>Java</i>	6
2.2. Etapa 1: Migración	7
2.3. Etapa 2: Refactoring	9
2.3.1. Diseño	9
2.3.2. Funcionamiento	9
2.4. Etapa 3: FrontEnd	11
3. Conclusiones	14
Anexo A. Figuras Importantes	15
Referencias	23

Lista de Figuras

1	Organigrama Web Intelligence Centre.	15
2	Mapa de Objetos Claves versión 1.	15
3	Mapa de Fijación.	16
4	Representación matricial del Mapa de Fijación.	16
5	Escala de colores del Mapa de Objetos Claves.	17
6	Mapa de Objetos Claves versión 2.	17
7	Interfaz de la Plataforma del <i>Proyecto AKORI</i>	18
8	Análisis del sitio web de <i>GOOGLE</i>	18
9	Procesamiento de información.	19
10	Mapas desplegados.	19
11	Acercamiento del Mapa de Objetos Claves.	20
12	Mapa de Objetos Claves en escala de rojo.	20
13	Mapa de Objetos Claves en escala de azul.	21
14	Mapa de Objetos Claves en escala de verde.	21
15	Mapa de Objetos Claves en escala de naranja.	22

Lista de Códigos

1	Código mapa de objetos versión 1.	4
2	Obtención <i>WebObjects</i>	6
3	Algoritmo mapa de objetos simplificado.	7
4	Condición de borde.	7
5	Bibliotecas utilizadas para el mapa de objetos.	8
6	Destacación de objetos web	8
7	Consulta al body del sitio web.	8
8	Algoritmo mapa de objetos de simplificado versión <i>Python</i>	8
9	Función que pondera la matrix a valores enteros.	9
10	Forma matricial de una imagen.	10
11	Función de construcción para la matriz de opacidad.	10
12	Fusión de imágenes	10
13	Llamada <i>AJAX</i> para despliegue del Mapa de Objetos.	12
14	Función de despliegue del Mapa de Objetos.	12
15	Función que genera el cambio de colores en el Mapa de Objetos.	13
16	Contenido <i>HTML</i> donde se aloja el Mapa de Objetos.	13

1. Introducción

1.1. Lugar de Trabajo

La práctica profesional se realizó en el Web Intelligence Centre [1], en adelante WIC, es un centro de investigación dependiente de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, dirigido por el académico Juan Velásquez del Departamento de Ingeniería Industrial de la Universidad de Chile. Su misión es desarrollar investigación en el campo de Tecnologías de Información creando soluciones para abordar problemas complejos de ingeniería utilizando herramientas basadas en la Web de las Cosas. Se encuentra ubicado en Beaucheff #993, Santiago, Chile. El trabajo se realizó de forma presencial en las instalaciones del centro, entre el 3 al 31 de Enero de 2017. En figura 1 es posible apreciar el organigrama del centro.

1.2. Grupo de trabajo

En el WIC trabajan investigadores de tiempo completo, desarrolladores con experiencia, profesionales del área de la salud debido a que muchos de sus proyectos son en conjunto con la Facultad de Medicina, alumnos de pregrado que pueden ser memorista sobre algún tema de investigación o trabajadores part-time y estudiantes de magister del Departamento de Ingeniería Industrial. En particular la oficina donde se realizó el trabajo era usada regularmente por 6 personas, de los cuales había un investigador, dos memoristas, un ingeniero de proyectos y otro practicante que también trabajó en el *Proyecto AKORI*. El *Proyecto AKORI* [2] es dirigido por el Ingeniero de Proyecto, Felipe Vera, quien fue el tutor del practicante (pero no se encontraba en la misma oficina).

1.3. Equipos y Software

1.3.1. Software

El software utilizado para desarrollar el trabajo fue *Python* [3] como lenguaje de programación (a través del IDLE Pycharm proporcionado por JetBrains [4]), el framework para aplicaciones Web Python, *Django* [5], *JavaScript* y *AJAX* para la creación de páginas web dinámicas, *Git* para el manejo de control de versiones, el navegador sin interfaz gráfico (conocido como headless browser), PhantomJS, el entorno de pruebas software para aplicaciones web, Selenium [6], dos bibliotecas de Python, una para edición de imágenes, Python Imaging Library, y otra para la generación de un mapa de colores, Matplotlib, y el sistema operativo Ubuntu 16.04 LTS. Además cabe destacar del aplicación del *Proyecto AKORI* [2] en la cual se trabajo. A continuación se presenta una breve descripción de los elementos más relevantes para el desarrollo y comprensión del trabajo realizado:

1. **Python:** Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. En el contexto del trabajo Python sirvió como lenguaje de programación por el lado del servidor para la aplicación web que actualmente funciona como prototipo al *Proyecto AKORI*.

2. **Django:** Es un framework de desarrollo web de código abierto (*open source*), escrito en *Python* que respeta el patrón de diseño conocido como Modelo-Vista-Controlador (MCV) [7], que en pocas palabras separa los componentes más importantes de una aplicación en tres grandes grupos, el modelo donde descansan los datos de la aplicación (Base de Datos), la vista que se encarga de todo el aspecto visual de una aplicación, y los controladores que ejecutan toda la lógica de funcionamiento.

En el caso de *Django* tiene la peculiaridad que las vistas reciben el nombre templates (encargados del *frontend* de la aplicación) y los controladores se llaman vistas (Views en inglés y encargadas del *backend*). Para el trabajo realizado fue necesario tomar la aplicación desarrollada, añadir cambios en los templates y agregar algunas funciones a las vistas para controlar la lógica del mapa de objetos que en las próximas secciones serán explicadas en detalle.

3. **JavaScript y AJAX:** Es un lenguaje de programación que es utilizado para la creación de páginas web de dinámicas por el lado del cliente en una aplicación web. *AJAX* es una extensión a la funcionalidad de *Javascript* que hace llamadas asíncronas al servidor web obteniendo información sin recargar la página por completo. Dentro del trabajo de práctica fueron utilizados para obtener el input de datos para el despliegue del mapa de objetos sin recargar toda la aplicación en la consulta.
4. **PhantomJS:** Es un navegador sin interfaz gráfica que sirve para realizar pruebas a una aplicación que se encuentra en fase de desarrollo. Fue utilizado para realizar web scraping [8], término que describe la recolección de información a través de la Web usando programas automatizados.
5. **Selenium:** Es un entorno de pruebas de software para aplicaciones basadas en la web, con bibliotecas existentes para muchos lenguajes de programación como *Python* y *JAVA*. Se usó para realizar el web scrapping junto a *PhantomJS* sobre los distintos sitios web que fueron testeados.

1.3.2. Proyecto AKORI

Para contextualizar de mayor forma el trabajo de práctica realizado a continuación se explicará brevemente que es el Proyecto AKORI [2] y cuál es su meta a seguir.

Antes de eso es necesario definir que son los objetos web [9], y que se entiende cuando se habla de ellos dentro del mapa de objetos, estos son definidos como la combinación grupal o individual de elementos pertenecientes al *DOM* [10]. En adelante cuando hablemos de objetos claves o simplemente de objetos nos referimos a los de la definición anterior.

El objetivo del proyecto AKORI es desarrollar una plataforma virtual que permita extrapolar el comportamiento de navegación de los usuarios cuando ingresan a sitio web a partir de la extracción de patrones de datos analizados con técnicas de minería de datos, eye tracking, y encefalografía. Para lograr esto, el proyecto se divide en 5 grandes etapas que son las siguientes:

1. Adaptar y refinar un repositorio de datos elaborado con estímulos visuales obtenidos desde sitios web, incorporando los datos de un mayor número de personas y diferenciándolos según variables sociodemográficas.
2. Adaptar e integrar algoritmos de minería de datos masivos (big data) para determinar patrones que permitan caracterizar y extrapolar la navegación de un usuario en un sitio web

mediante análisis de exploración visual utilizando datos provenientes desde eye-tracking, dilatación pupilar y electroencefalografía.

3. Diseñar, construir y evaluar una plataforma prototipo que integre algoritmos de web Intelligence, un repositorio de datos en base a estímulos visuales y electroencefalografía con patrones de comportamiento web para extrapolar las preferencias y el comportamiento de potenciales usuarios web.
4. Testear y evaluar funcionalidades del sitio, capacidad del sistema para lograr lo que se desea y minimizar errores de uso; además de medir el nivel de usabilidad (facilidad de aprender y recordar, eficiencia, minimización de errores de uso y satisfacción del usuario).
5. Prospeccionar y valorizar el mercado, como también la propiedad intelectual. Definir una estrategia para el empaquetamiento y transferencia de la tecnología.

A pesar de que las investigaciones continúan, existe un prototipo de la aplicación web con algunos módulos (mapas esquemáticos) en desarrollo y otros ya implementados. A continuación se describen los módulos ya implementados, incluido el que desarrolla el practicante.

1. **Mapa de Fijación Ocular:** Construye un mapa de calor con las zonas más probables de fijación ocular en un sitio web a partir de una escala de colores. Los clientes pueden visualizar las zonas más vistas en rojo, las menos vistas en azul y las que tienen ninguna probabilidad de ser vistas adquieren un color en escala de grises.
2. **Mapa de Dilatación Pupilar:** Construye un mapa calor siguiendo una escala similar al mapa de fijación ocular, con la diferencia que las zonas que adquieren colores más intensos muestran donde es más probable que un usuario haga un click dentro de la página analizada.
3. **Mapa de Objetos de Claves:** Se construye un mapa que muestra los objetos web más relevantes que un usuario visualiza al ingresar a un sitio web, los objetos más importantes son coloreados con más oscuros que los menos vistos.

Además de los módulos explicados anteriormente, se está trabajando en tres mapas más que aún no han sido llevados a producción, estos son, el índice de claridad, el mapa de probabilidad de clicks, y el mapa de percepción; donde los primeros dos son complementos a los ya implementados, y el último mostrará donde es más probable que el usuario dirigirá la mirada los primeros tres segundos de ingresar al sitio web.

En las figuras 7, 8, 9, 10 puede apreciarse la interfaz de la plataforma mientras se analiza la página de *GOOGLE*.

1.3.3. Equipo

En cuanto a los equipos computacionales, el centro provee tanto equipos estacionarios como portátiles para aquellos que encuentran trabajando e investigando, sin embargo para los practicantes y memoristas deben llevar sus equipos personales para desarrollar sus labores. En particular la práctica realizada solo fue necesario un equipo portátil para programar el mapa de objetos.

1.4. Situación Previa

A continuación se describirá la situación previa al trabajo del practicante. Esta descripción se centrará en dos versiones del *Proyecto AKORI*, una escrita en *Python* utilizando el framework *Django* que actualmente está producción y la otra escrita en *Java* usando *Servlet* que ya no se encuentra en funcionamiento.

1.4.1. Versión Java con Servlet

En esta versión estaban implementados la mayoría de los módulos que fueron explicados en la sección anterior. Además una versión del mapa de objetos claves que sin embargo, resaltaba todos elementos del sitio web (ver figura 2) y no filtraba los elementos de mayor a menor interés. En el siguiente extracto del código se puede apreciar el algoritmo del mapa de objetos claves.

Código 1: Código mapa de objetos versión 1.

```
1 query = driver.findElements(By.tagName(temp.tagName()));
2 for (WebElement temp1 : query) {
3     try{
4         Point po = temp1.getLocation();
5         Dimension d = temp1.getSize();
6         if (d.width <= 0 || d.height <= 0 || po.x < 0 || po.y < 0) {
7             continue;
8         }
9         graph.draw(new Rectangle(po.x, po.y, d.width, d.height));
10        }
11        catch (ElementNotVisibleException ex){
12            System.out.println("Objeto no visible 0:");
13        }
14        catch (StaleElementReferenceException ex){
15            System.out.println("Parece que el objeto ya no existe");
16        }
17        catch (NoSuchElementException ex ){
18            System.out.println("Parece que el objeto ya no existe II");
19            ex.printStackTrace();
20        }
21        ++i;
22    }
23 }
24 }
```

A grandes rangos este código realizaba lo siguiente, almacenaba en la variable *query* todos los objetos web provenientes del *DOM* en una instancia de la clase *ArrayList*, para luego a entrar a un ciclo donde se dibujaba un rectángulo (que encerraba al objeto web) de acuerdo a la posición y las dimensiones de cada objeto web sobre una imagen tomada del sitio web que se estaba analizando. Como en esta versión no existía una clase especial para el mapa de objetos solo se incluye el extracto del código relacionado al trabajo realizado.

Para el practicante, entender todas clases de la versión *JAVA* no fue una tarea difícil debido a que contaba con experiencia previa programando con dichas tecnologías.

1.4.2. Versión *Django*

Previo al trabajo del practicante solo existía una implementación de los mapas de fijación ocular y dilatación pupilar. Como esta versión estaba hecha utilizando el framework *Django*, seguía los principios de programación de *Python*, además de la clara separación de componentes que *Django* administra (patrón MVC), que es muy diferente a la arquitectura que ofrece la programación web *JAVA* y su contenedor de *Servlet*. Sin embargo, la funcionalidad e interfaz gráfica era la misma que la versión previa a la migración del lenguaje que manejaba el servidor web.

1.5. Descripción general del trabajo realizado

Como ya se mencionó en secciones anteriores, el objetivo del trabajo desarrollado fue el proveer un mapa que resaltaré los objetos web que un usuario visualiza a los pocos milisegundos de ingresar a un sitio web. Como no existía una versión del mapa de objetos en la segunda versión de la aplicación, como primera instancia se agregó una versión preliminar del mapa de objetos tal como existía en la primera versión (ver figura 2) que descataba todos los elementos del *DOM* de un documento *HTML*, luego se ideó un mapa de objetos que diferenciaba los elementos más a menos vistos siguiendo un algoritmo de suma de matrices, donde la matriz era gráfícada (ver figura 5) para generar una imagen que era fusionada a otra imagen tomada del sitio web analizado bajando la opacidad en cada una. En la figura 6 es posible visualizar lo descrito previamente.

Para llevar a cabo el trabajo fue dividido por el practicante en tres etapas que se describen brevemente a continuación:

1. **Exploración del código previo:** Dado que el trabajo consistió en agregar el mapa de objetos en una tecnología desconocida, fue necesario estudiar el framework tanto su sintaxis como su filosofía de programación, para luego comenzar a explorar el código existente dentro de la aplicación de forma de identificar aquellas clases y funciones claves involucradas a la partes a extender.
2. **Migración del código:** Como no existía un mapa de objetos en la nueva en la segunda versión de la aplicación, se tomó parte del algoritmo existente en la versión del mapa de objetos, y se realizó una migración del código *JAVA* a *Python* obteniendo un mapa de objetos que encerraba los objetos web tal como en la versión anterior.
3. **Refactoring:** Luego de agregar el mapa de objetos en su versión preliminar, se ideó un algoritmo de sumas de matrices, en donde se tomaban los datos de visualización (obtenidos con algoritmos de minería de datos y las otras técnicas mencionadas) en forma de una matriz de observación para luego fusionarla con una matriz que representaba el sitio (es decir, se tomó una fotografía del sitio web y se hizo una conversión a su representación matricial) en una escala de opacidad tal como se aprecia en la figura 6.
4. **Etapas de FrontEnd:** Una vez listas las funciones y algoritmos de las etapas anteriores, se pasó a la última etapa en la que utilizando *Javascript* y funciones *AJAX* se implementó el mapa de objetos final dentro de la aplicación web, tal como se muestra en la figura 11 y está disponible en su sitio oficial [3].

2. Trabajo Realizado

Como se ha mencionado previamente el trabajo consistió en la creación de un mapa de los objetos web [9] más relevantes que un usuario visualiza mientras navega en un sitio web. Los datos que se usaron para indicar donde se encontraban los objetos claves eran recibidos desde el mapa de fijación ocular (que se explicó en las secciones pasadas) que fueron obtenidos por algoritmos de minería de datos, eye tracking, y electroencefalografía. Estos datos son el punto de partida del mapa de objetos implementado. Los datos eran representados en una imagen basada en un *screenshot* [11] del sitio que se estaba analizando, la imagen tenía una coloración más intensa en los lugares más probables (ver figura 3) que un usuarios fijaría su atención en los primeros milisegundos. Luego se realizaba conversión de la imagen de visualización a sus representación matricial (ver figura 4) con números enteros entre 0 a 255 (que es la escala de colores RGB [12]) los lugares dentro de la matriz que tenían numeros más altos correspondían a las zonas más probables de observación. Finalmente, se gráficaba dicha matriz generando una imagen como la de la figura 5, dicha imagen se fusionaba con un *screenshot* del sitio web produciendo la imagen que se puede apreciar en la figura 6, los colores más oscuros representaban los objetos más vistos, el último paso fue agregar estos cambios a la aplicación oficial.

A continuación se explicará en profundidad el detalle del proceso descrito y las etapas mencionadas en la sección anterior.

2.1. Etapa 0

El primer paso realizar las modificaciones solicitadas fue identificar las clases involucradas en el mapa de objetos de la versión *JAVA*. Esta exploración se realizó al mismo tiempo que el practicante aprendió sobre los elementos específicos para el desarrollo de aplicaciones web *Django*, como lo son los modelos, vistas, templates en especial elementos de *HTML* y *CSS*.

2.1.1. Algoritmo Mapa de Objetos versión *Java*

A pesar de que existía un mapa de objetos en la versión 1 del proyecto, el algoritmo no contaba con una clase propia dentro del código de la aplicación. A continuación se realizará una explicación más detallada del código de la subsección 1.4.1 (ver código 1) siguiendo paso a paso su ejecución:

1. Para obtener todos los elementos del *DOM* se empleaba una consulta al *WebDriver Object* un objeto perteneciente a la biblioteca *Selenium* para *JAVA* encargado de ejecutar llamadas directas al navegador y conseguir información (*Web Scrapping* [8]) del sitio en cual se estaba navegando, luego usando el método *findElements* se llevaba cabo la consulta que era conseguir todos los *WebObject* que poseían un *TagName*, cuyo nombre de etiqueta (*TagName*) era referido a las etiquetas de *HTML*, dichos elementos eran guardados en una variable query que era una instancia de la clase *ArrayList Object*, como su nombre lo indica es el objetos que representa las en *JAVA*.

Código 2: Obtención *WebObjects*.

```
1 query = driver.findElements(By.tagName(temp.tagName()));
```

2. Luego, para dibujar el mapa de objetos se ingresaba a un bucle manejado por un ciclo *for* en el cual se recorría la lista de *Web Objects*, y utilizando los objetos *Point* y *Dimension* ambos provenientes de la biblioteca de *Selenium*, el cual dichos objetos guardaban las posiciones de objeto web en la representación matricial del *screenshot* del sitio de estudio. Dichas posiciones servían para instanciar un objeto *Rectangle* de la clase estándar de *JAVA*, y finalmente con un objeto *Draw* eran dibujados dentro del *screenshot* tomado. En el siguiente código puede apreciarse una versión simplificada de dicho algoritmo.

Código 3: Algoritmo mapa de objetos simplificado.

```
1 for (WebElement temp1 : query) {  
2     Point po = temp1.getLocation();  
3     Dimension d = temp1.getSize();  
4     graph.draw(new Rectangle(po.x, po.y, d.width, d.height));  
5 }
```

3. Algunos puntos que fueron omitidos del algoritmo anterior es que podían existir errores en su ejecución que provenían el sitio web que se estaba analizando. Dichos errores fueron neutralizados con sentencias *try-catch*, aislando así los errores más frecuentes.
4. El último punto importante a mencionar existían algunos objetos web cuyo tamaño escapaba las dimensiones de la pantalla donde se visualizaba el sitio web, dichos objetos no fueron considerados dentro del mapa de objetos debido a que no eran importantes para el análisis de determinar los objetos web más visto por el usuario. Estos se aislaban de acuerdo a sí las posiciones del web object escapaban a una coordenada negativa. Una sentencia *if* se encargaba de lo indicado.

Código 4: Condición de borde.

```
1 if (d.width <= 0 || d.height <= 0 || po.x < 0 || po.y < 0) {  
2     continue;  
3 }
```

2.2. Etapa 1: Migración

Una vez encontradas las clases involucradas y aprendido el funcionamiento de *Django*, se procedió a migrar el algoritmo del mapa de objetos ya existente al lenguaje *Python* testeando el resultado con *PhantomJS* y la biblioteca de *Selenium* para *Python*. La depuración del algoritmo puede ser dividada en las siguientes fases que implementan el mapa:

1. **Importación:** Para migración del código se utilizaron las siguientes bibliotecas de Python:

Código 5: Bibliotecas utilizadas para el mapa de objetos.

```
1 from selenium import webdriver
2 from selenium.webdriver.common.by import By
3 from PIL import Image, ImageDraw
```

2. **Obtención del *Screenshot* del sitio web:** Una vez elegida la url del sitio web a analizar, se obtenía un *screenshot* del sitio web con la ayuda de *PhantomJS* seteando el tamaño de la ventana y la url para luego ser guardadas dentro del servidor. Luego utilizando la biblioteca de edición de imágenes se instanciaban los objetos, *Imagen*, que abría que la imagen a editar y *ImageDraw*, que editaba la imagen abierta, en este caso el *screenshot*.

Código 6: Destacación de objetos web

```
1 driver = webdriver.PhantomJS()
2 driver.set_window_size(1301, 744)
3 driver.get("http://google.com")
4 driver.save_screenshot("/path/screenshot.png")
5 image = Image.open("/path/screenshot.png")
6 draw = ImageDraw.Draw(image)
```

3. **Obtención de objetos web:** Con la biblioteca *Selenium* era posible conseguir todos objetos que poseía el sitio, utilizando una consulta que obtenía todos los elementos dentro del body que manejaban un *tag name*, retornando una lista con los objetos web.

Código 7: Consulta al body del sitio web.

```
1 element = driver.find_element(By.TAG_NAME, 'body')
2 body = element.find_elements(By.XPATH, '.*/*')
```

4. **Destacación de objetos web:** Finalmente, a través de un ciclo se recorría la lista de elementos dibujando un rectángulo en el *screenshot* que destacaba la posición de dicho objetos en cada iteración. La posición exacta era determinada la librería *Selenium*.

Código 8: Algoritmo mapa de objetos de simplificado versión *Python*.

```
1 for b in body:
2     x1 = b.location['x']
3     y1 = b.location['y']
4     x2 = x1 + b.size['width']
5     y2 = y1 + b.size['height']
6     draw.rectangle([x1, y1, x2, y2], outline = 255)
```

El resultado era un mapa igual al de la figura 2. Algunas partes del código han sido omitidas debido a que no formaban parte esencial de la explicación.

2.3. Etapa 2: Refactoring

2.3.1. Diseño

Antes de detallar el funcionamiento del código realizado se explicará el patrón de diseño utilizado. Como se dijo en secciones pasado el framework *Django* sigue el patrón *Modelo Vista Controlador* (MVC) [7].

El MVC consta de las siguientes partes:

1. **Modelo:** Su responsabilidad es el manejar la representación de la información con la cual se está trabajando, por lo tanto le corresponde interactuar y gestionar dicha información.
2. **Controlador:** Es el intermediario entre la vista y el modelo, le corresponde responder a eventos y enviar peticiones al modelo. Sin embargo, para el caso *Django* recibe el nombre de Vistas, *Views* en inglés.
3. **Vista:** Muestra la información de forma gráfica al usuario, en un formato apropiado. La información mostrada corresponde a la información que provee el modelo. En el caso de *Django* reciben el nombre de templates.

Aplicando este modelo a las responsabilidades identificadas en la programación del mapa de objetos se obtuvo lo que cada parte del MVC debe realizar.

- a) El **Modelo** debe encargarse de la extracción de datos, pues el modelo el que debe encargarse de interactuar y gestionar la información.
- b) La **Vista**, o los *templates*, deben encargarse de desplegar el mapa de objetos cuando se inserta una url de un sitio web a analizar.
- c) El **Controlador**, o las *Views*, que se encargan de toda la lógica de programación del mapa de objetos claves.

2.3.2. Funcionamiento

Con la migración del código *JAVA* a *Python* fue posible desarrollar un mapa más completo donde se diferenciarán los objetos más vistos a lo menos vistos. Como ya se había mencionado en secciones previas; los datos que mostraban las posiciones de los objetos web más vistos dentro del sitio de estudio, provenían de la imagen (del sitio web) que generaba el mapa de fijación (ver figura 3). Para trabajar con estos datos, se procedía a transformar la imagen a su representación matricial, en cual donde antes habían existido colores más intensos ahora estaba la concentración de los números más grandes. Sin embargo, estos valores eran entre 0 a 1, para tener un mayor acercamiento a la escala de colores RGB se ponderaba la matriz para obtener valores de 0 a 255, con la función definida en el siguiente código:

Código 9: Función que pondera la matrix a valores enteros.

```
1 def multiplier_matrix(matrix):
```

```
2 matrix = matrix.T * (255.0 / max(matrix))
3 return matrix
```

Para obtener la representación matricial del mapa de fijación ocular se empleó de la biblioteca de *Python*, *matplotlib* [13], (usada para la generación de gráficos) con el método *imread* que retornaba la matriz de la imagen que recibe como entrada.

Código 10: Forma matricial de una imagen.

```
1 matrix = matplotlib.image.imread(screen_path)
```

Sin embargo, la concentración de valores en la matriz producida por el mapa de fijación no era regular, en el sentido que los objetos en el mapa de objetos claves eran representados rectángulos y en dicha matriz la distribución de concentración era irregular, por lo tanto se creó otra matriz, llamada matriz de opacidad, para rellenar sus valores en función de la concentración de la matriz generada por el mapa de fijación y con forma de rectángulo para así respetar la forma de cada objeto web dentro del sitio web. Para ello se utilizó la función *buildMatrix* que rellenaba una matriz de ceros con cierto valor de opacidad en las posiciones de cada objeto web, el código de la función es el siguiente:

Código 11: Función de construcción para la matriz de opacidad.

```
1 def buildMatrix(matrix, x1, y1, x2, y2, alpha):
2     i = x1
3     while (i != x2):
4         j = y1
5         if alpha < matrix[i, j]:
6             break
7         while (j != y2):
8             if alpha < matrix[i, j]:
9                 break
10            matrix[i, j] = alpha
11            j += 1
12        i += 1
13    return matrix
```

La matriz retornaba por la función *buildMatrix* elaboraba un mapa de colores en función de las posiciones de los objetos web, para obtener la imagen proyectada por dicha matriz, se graficaba la matriz gracias a la función *plot* de la biblioteca *matplotlib* [13] y guardaba el archivo con el método *save* de la misma biblioteca. Finalmente se efectuaba una fusión del mapa de colores obtenido y un *screenshot* del sitio web, marcando cierta opacidad para que el mapa de colores resaltará los objetos web previsto en el *screenshot*. El resultado de esto se encuentra en la figura 6 y es realizado por el código de la función *objmapColor*, donde el método *blend* se encarga de la fusión de la imágenes pasadas como argumento, su tercer argumento indica la escala de opacidad, en este caso es 0.5.

Código 12: Fusión de imágenes

```
1 def objmapColor(screenshot, path, matrix, color_string):
2     '''Funcion que retorna la image final para el mapa de objeto realizando un blend
    entre el screenshot y el mapa de colores.
```

```
3 parametro screenshot: imagen de screenshot tomando de la pagina.
4 parametro path: path donde se almacena el screenshot tomado de la pagina web.
5 parametro matrix: matriz de opacidad.
6 parametro: color que se quiere para el mapa de color.
7 ',,'
8 mpimg.imsave(path, matrix, cmap = color_string)
9 img = Image.open(path)
10 output = Image.blend(screenshot, img, 0.5)
11 return output
```

El resto del algoritmo de construcción del mapa de objetos es análogo a los explicado en la subsección anterior por lo que la omitimos. Es importante descartar que el funcionamiento explicado corresponde a los controladores de la aplicación que luego fueron añadidos al archivo *Views* del proyecto, dejando para la subsección siguiente la implementación de las Vistas, es decir el despliegue del mapa de objetos dentro de la aplicación web.

2.4. Etapa 3: FrontEnd

Una vez implementada la lógica del algoritmo, la etapa final fue trasladar el algoritmo a la aplicación y visualizar el mapa de objetos dentro del servidor. Como bien se ha dicho el framework *Django* separa las responsabilidades siguiendo el patrón de diseño MVC, en la subsección anterior se explicó la lógica detrás del mapa de objetos que dentro de la aplicación corresponde al *Controlador* (las *Views* en *Django*), la parte del *Modelo* era proporcionada por los datos del mapa de fijación, por ende, esta etapa se centra en el desarrollo de las *Vistas*, que en *Django* reciben el nombres de *Templates*.

Como el practicante debió agregar nueva funcionalidad a una aplicación existente mucha parte del código que producía la interfaz de usuario ya estaba hecho, por lo que esta parte del trabajo se restringió en añadir la interfaz de usuario del mapa de objetos.

El desarrollo del despliegue del mapa de objetos se divide en dos paso, el primero son las llamadas *AJAX* que se encargan de procesar la información obtenida para generar el mapa de objetos sin recargar la página por completo y el segundo, la creación del contenedor que recibirá el mapa de objetos que es mostrado al usuario. Para simplificar el manejo del código a las llamadas *AJAX* se utilizó la biblioteca *jQuery* perteneciente a *JavaScript*. A continuación se detallan los pasos mencionados dentro de la ejecución del código:

- a) **Llamadas *AJAX***: Se creó una función *JavaScript* nombrada *objmap* encargada de ejecutar la llamadas *AJAX* siguiendo la sintaxis de *jQuery*, dicha función realizaba una solicitud GET del protocolo HTTP para conseguir la información del mapa de fijación, con un *timeout* 120000 milisegundos antes de efectuar la petición, una vez obtenidos los datos se ejecutaba la función de la variable *success* propia de *jQuery*. La sintaxis *\$.ajax{...}* se utiliza para ejecutar llamadas *AJAX*, por lo tanto la funcionalidad del objeto *XMLHttpRequest* (objeto usado para interacción entre servidor y cliente mediante las solicitudes y respuestas generadas entre ellos) [14] quedaba encapsulada por la biblioteca *jQuery* [15]. En el siguiente código se muestra lo descrito.

Código 13: Llamada *AJAX* para despliegue del Mapa de Objetos.

```

1 function objmap(sitio){
2 $.ajax(
3 {
4   type: 'GET',
5   timeout:120000,
6   url:'https://{ request.META.HTTP_HOST }/objmap/',
7   data:{url: sitio},
8   success: function(data){...}
9 }

```

- b) **Visualización:** Una vez obtenidos los datos, se ejecutaba la función dentro de la variable *success*, donde la variable *urlobj* guardaba los datos y mientras se procesaba la información se oculta el *div* contenedor del mapa de objetos (cuyo nombre dentro *DOM* es *objemapdiv*). Luego el elemento *HTML* para carga la imagen (cuya etiqueta es *img* y se añade al *div* contenedor) para a ser visualizado por un botón cuando el mapa ya es operativo.

Código 14: Función de despliegue del Mapa de Objetos.

```

1
2 success: function(data){
3   urlobj = data;
4   $("#objmapdiv").hide();
5
6   var imgdata = '<a id="objmaplink" href="'+urlobj+'_r.png" rel="prettyPhoto"></a>';
8   $("#objmapdiv").html(imgdata);
9   $("#objmaplink").prettyPhoto();
10  $("#objmapdiv").slideDown(800);
11  totalTime = (new Date().getTime()-ajaxTime)/1000;
12  totalTime = totalTime.toFixed(2);
13  $("#objmaptime").html('Se demoró '+totalTime+' segundos!');
14  $(".clickMe").show()
15 },
16 error: function(x, t, m) {
17 }
18 })

```

- c) **Funcionalidad Cambio de Colores:** En secciones previas se dijo que para la versión final de mapa colores existen cuatro tonalidades de colores rojo, azul naranja y verde. Para lograr cambiar el tipo de mapa dinámicamente se programó una función *JavaScript* que se llamaba al presionar botón del color correspondiente, dicha función se ejecuta desde el lado del cliente. Esta función recibía como argumento la ruta de la imagen a modificar, y creaba un etiqueta *HTML* de inserción de imagen, para luego se redireccionada al *div* contenedor del mapa de objetos con al ayuda del método *.slideDown* de la librería *JQuery*. Dicha función recibía la firma *objmap_colorcolor*, donde su argumento correspondía a la ruta de archivo con el mapa de objeto ya creado en el color correspondiente. El siguiente código muestra la funcionalidad descrita.

Código 15: Función que genera el cambio de colores en el Mapa de Objetos.

```

1 function objmap_color(color) {
2   var aux = color;
3   var imgdata = '<a id="objmaplink" href="'+urlobj.concat(aux)+'"
4     rel="prettyPhoto"></a>';
5   $("#objmapdiv").html(imgdata);
6   $("#objmaplink").prettyPhoto();
7   $("#objmapdiv").slideDown(800);
8 }

```

- d) **Código HTML:** El último paso consistió en agregar al templates la etiquetas *HTML* necesarias para desplegar el mapa de objetos, las cuales fueron el *div* contenedor, los 4 botones que modificaban la versión de color del mapa de objetos y una etiqueta que agregaba un pequeño parrafo donde se mostraba el tiempo tardaba en cargar el mapa. Esto se aprecia en el siguiente código *HTML* y se visualiza en la figura 10.

Código 16: Contenido *HTML* donde se aloja el Mapa de Objetos.

```

1 <div class="span4">
2   <div id="objmapdiv" class="mask2">
3     <div class="loadercontainer"><div class="loader"></div></div>
4   </div>
5   <div id="objdiv" class="inside">
6     <hgroup id="group" style="text-align:center;">
7       <h2 class="titleservice">Mapa de Objetos Claves</h2>
8       <p id="objmaptime" class="subtitle"></p>
9       <input class="clickMe" type="button" value="Rojo"
10         onclick="objmap_color('_r.png');"/>
11       <input class="clickMe" type="button" value="Azul"
12         onclick="objmap_color('_b.png');"/>
13       <input class="clickMe" type="button" value="Verde"
14         onclick="objmap_color('_g.png');"/>
15       <input class="clickMe" type="button" value="Naranja"
16         onclick="objmap_color('_o.png');"/>
17     </hgroup>
18   </div>
19 </div>

```

En las figuras 12, 13, 14 y 15 pueden observarse las distintas tonalidades del mapa de objetos cambiados por su respectivo botón.

Sí la última parte es bastante simple está fue la forma indicada por el tutor de práctica, ya que se estaba trabajando en una nueva interfaz a toda la aplicación que no formaba parte del trabajo pedido al practicante.

3. Conclusiones

Los resultados del trabajo presentado en este informe fueron considerados por el tutor de práctica como satisfactorios, siendo estos puestos en producción al poco tiempo de finalizada la práctica, los cuales pueden ser apreciados en la página oficial del *Proyecto AKORI* [3]. Además gracias a la programación hecha por el practicante cuando se ingresa la url del sitio web a analizar no solo es posible generar un mapa de los objetos más relevantes encerrados por rectángulos, sino que existe un mapa en escala de intensidad de colores de los objetos web más a los menos vistos por un usuario (en el sitio web que se está analizando), cambiando la escala en cuatro colores rojo, naranja, azul y verde.

Respecto al futuro de la aplicación hay que destacar que aún se encuentra trabajando en ella realizando actualizaciones para mejorar su llegada al público o a los posibles clientes (si llega a comercializarse) en términos de visualizaciones que le provean más y mejor información al usuario sobre el sitio web a analizar, en particular se seguirá trabajando en las secciones modificadas y agregadas por el practicante, en miras de mejorar la performance de las visualizaciones de los elementos más vistos dentro del DOM en el mapa de objetos mediante optimizaciones en la extracción de datos y algoritmos de búsqueda de objetos web más eficientes.

Dentro del aprendizaje obtenido en el proceso de práctica se destacan a nivel técnico el aprendizaje del framework *Django* para el desarrollo web y sus elementos específicos como vistas, modelos, templates. También el aprendizaje de *Javascript* es especialmente el uso de *AJAX* para el despliegue del mapa de objetos sin actualizar la página completa y *Git* para el manejo de control de versiones. Otro punto no menos importante dentro del aprendizaje de todas las tecnologías fue lo primordial que es leer la documentación antes de utilizar cualquier tecnología ya que simplifica mucho el tiempo de aprendizaje. Dentro de otros aspectos se aprendió sobre el trabajo en equipo, la importancia de mantener constantemente comunicación para realizar un buen trabajo en particular cuando se trabaja en un equipo multidisciplinario. El practicante estuvo en constante contacto con algunos investigadores y el con encargado para obtener un mapa de objeto acorde a los requerimientos y entendible para cualquier usuario use la plataforma del *Proyecto AKORI*. Finalmente queda destacar el aprendizaje de documentar bien el código hecho para que sea fácil de entender y extender a medida que el proyecto crece, y además de contar con buen diseño en el código muchas veces esto puede resultar fácil, pero en la práctica puede ser determinante la continuidad de una aplicación robusta que pueda seguir creciendo y no empezar un sistema que prácticamente hará lo mismo desde 0.

Anexo A. Figuras Importantes

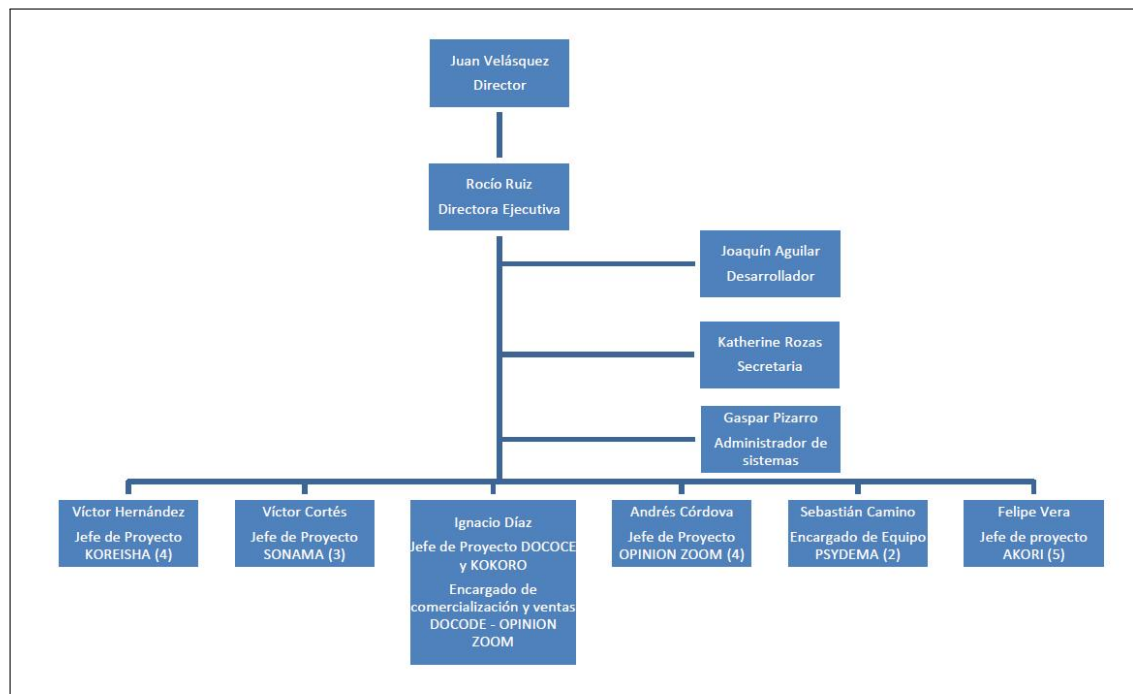


Figura 1: Organograma Web Intelligence Centre.

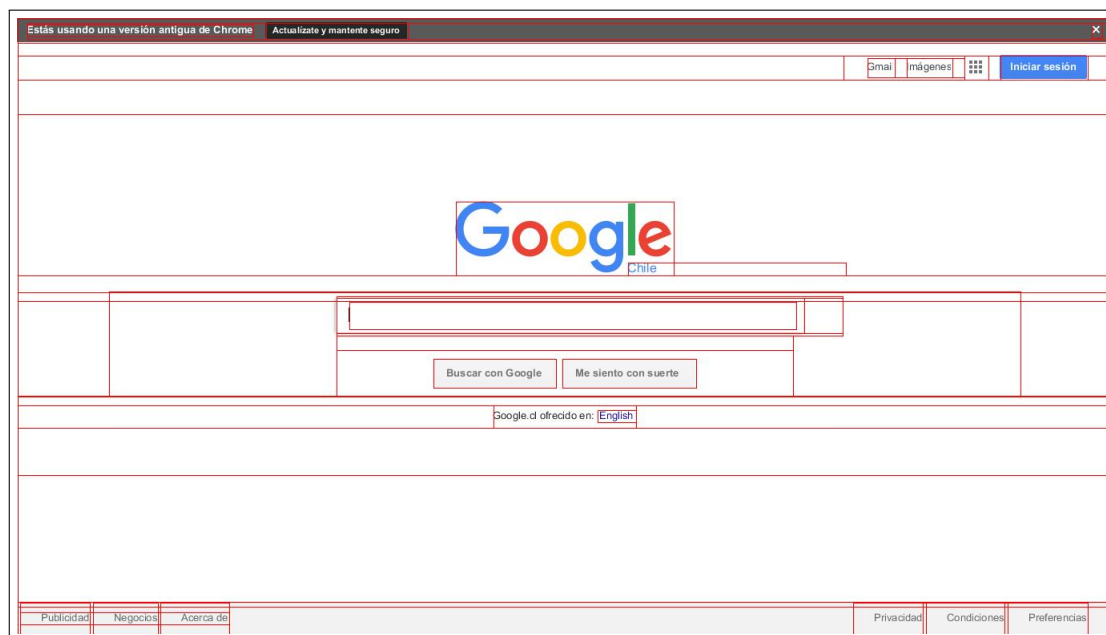


Figura 2: Mapa de Objetos Claves versión 1.

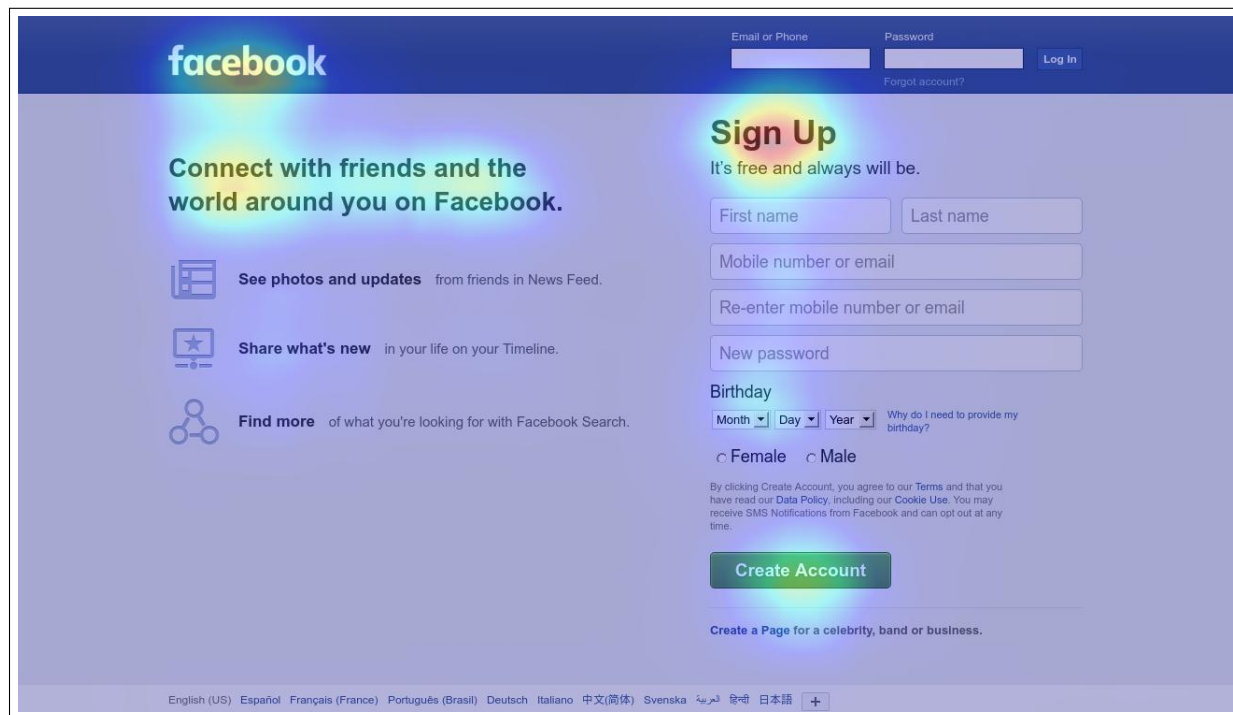


Figura 3: Mapa de Fijación.

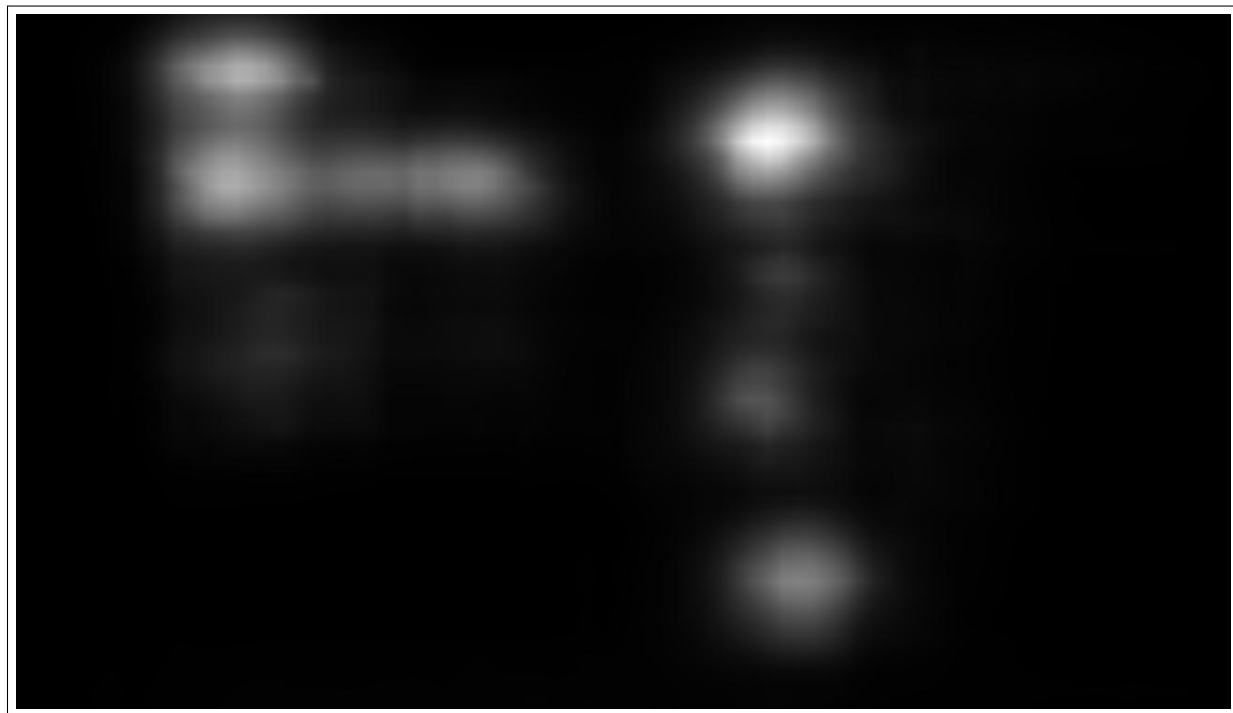


Figura 4: Representación matricial del Mapa de Fijación.

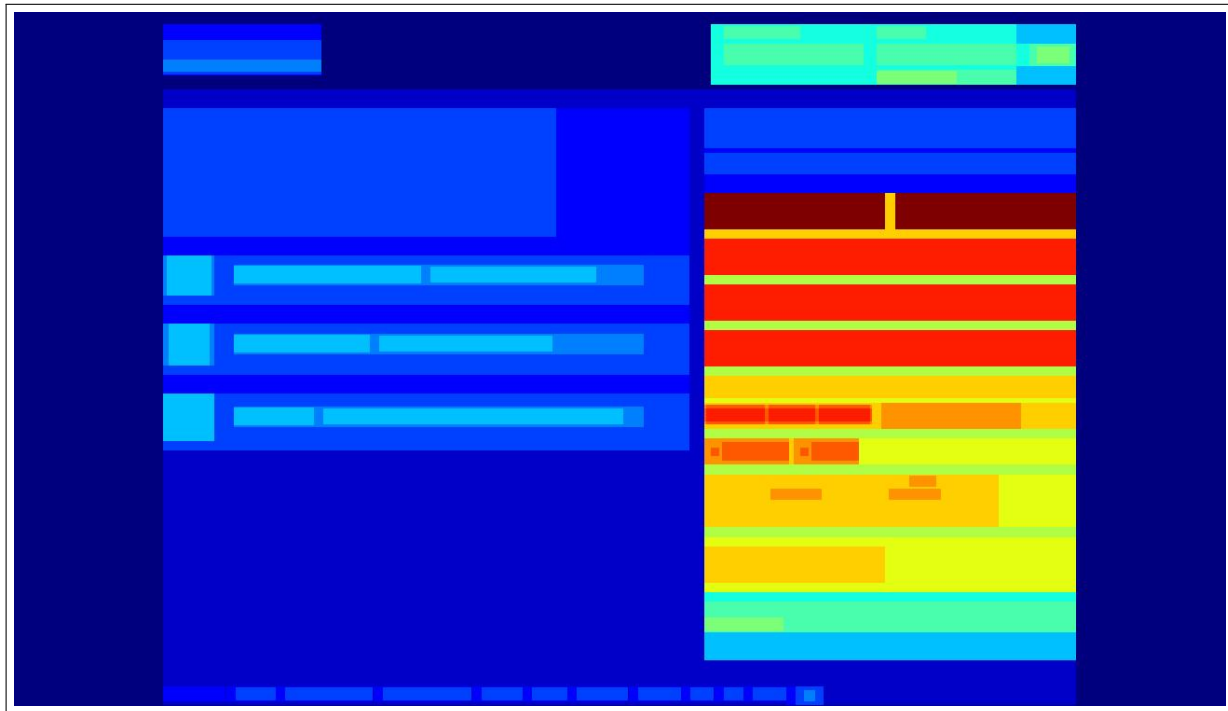


Figura 5: Escala de colores del Mapa de Objetos Claves.

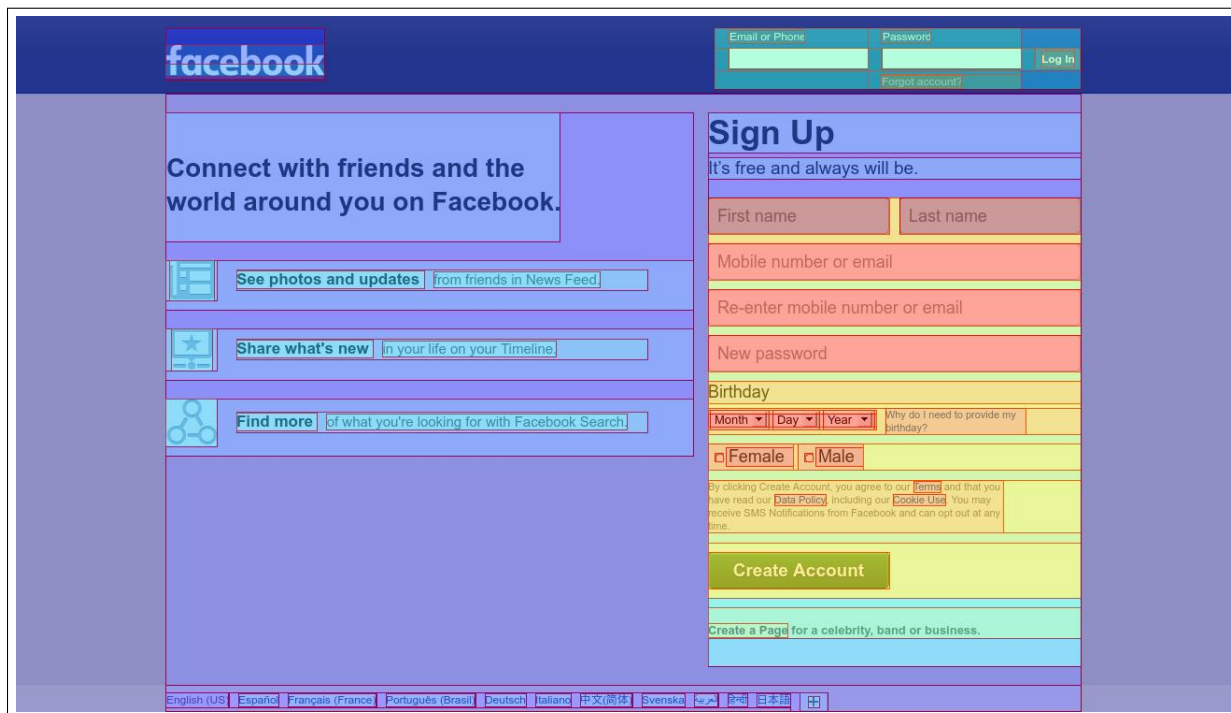
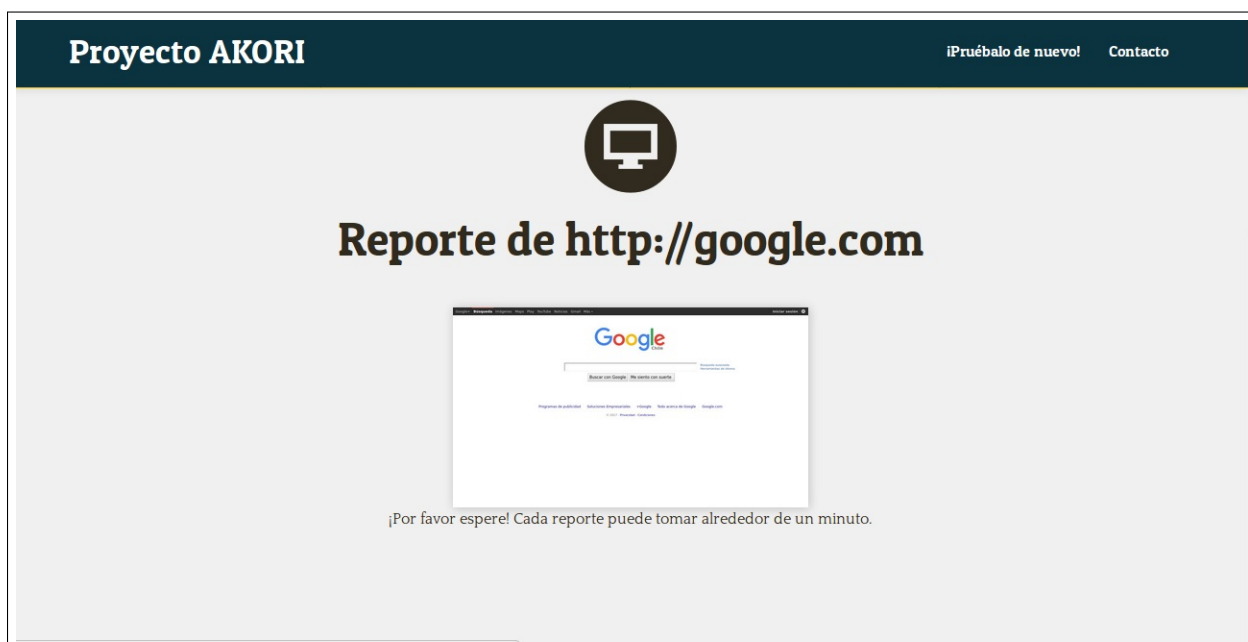


Figura 6: Mapa de Objetos Claves versión 2.



The screenshot shows the main interface of the Proyecto AKORI website. The header is dark blue with the logo 'Proyecto AKORI' on the left and navigation links 'Qué hacemos', 'Ejemplos', 'Equipo', and 'Contacto' on the right. The main content area has a dark blue background with the text 'Proyecto AKORI' in yellow and 'Te ayudamos a mejorar tu sitio' in white. Below this, there is a form with the prompt 'Escribe la URL de tu sitio'. The form includes a text input field containing 'http://', a yellow button labeled '¡Pruébalo!', and two dropdown menus labeled 'Selecciona edad' and 'Sexo'.

Figura 7: Interfaz de la Plataforma del *Proyecto AKORI*.



The screenshot shows a report generated by the Proyecto AKORI platform. The header is dark blue with the logo 'Proyecto AKORI' on the left and navigation links '¡Pruébalo de nuevo!' and 'Contacto' on the right. The main content area has a light gray background. At the top, there is a circular icon of a computer monitor. Below the icon, the text 'Reporte de http://google.com' is displayed in a large, bold, black font. Underneath the text, there is a small thumbnail image of the Google homepage. At the bottom of the report, there is a message: '¡Por favor espere! Cada reporte puede tomar alrededor de un minuto.'

Figura 8: Análisis del sitio web de *GOOGLE*.



Figura 9: Procesamiento de información.

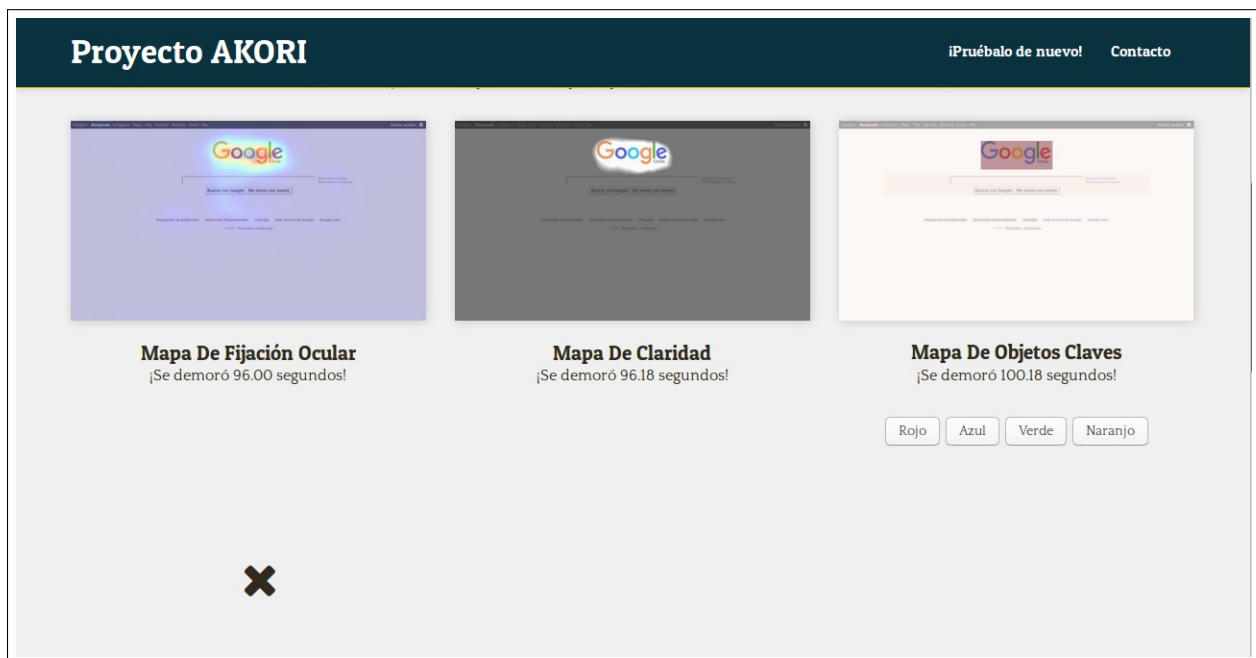


Figura 10: Mapas desplegados.

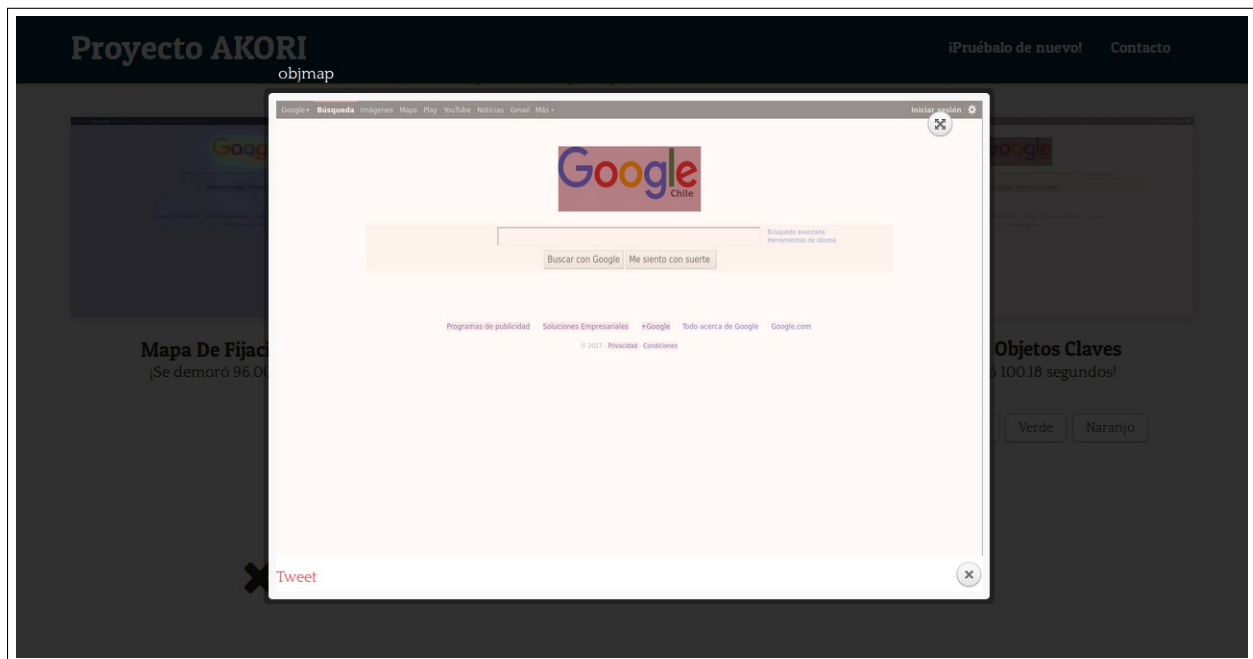


Figura 11: Acercamiento del Mapa de Objetos Claves.



Figura 12: Mapa de Objetos Claves en escala de rojo.

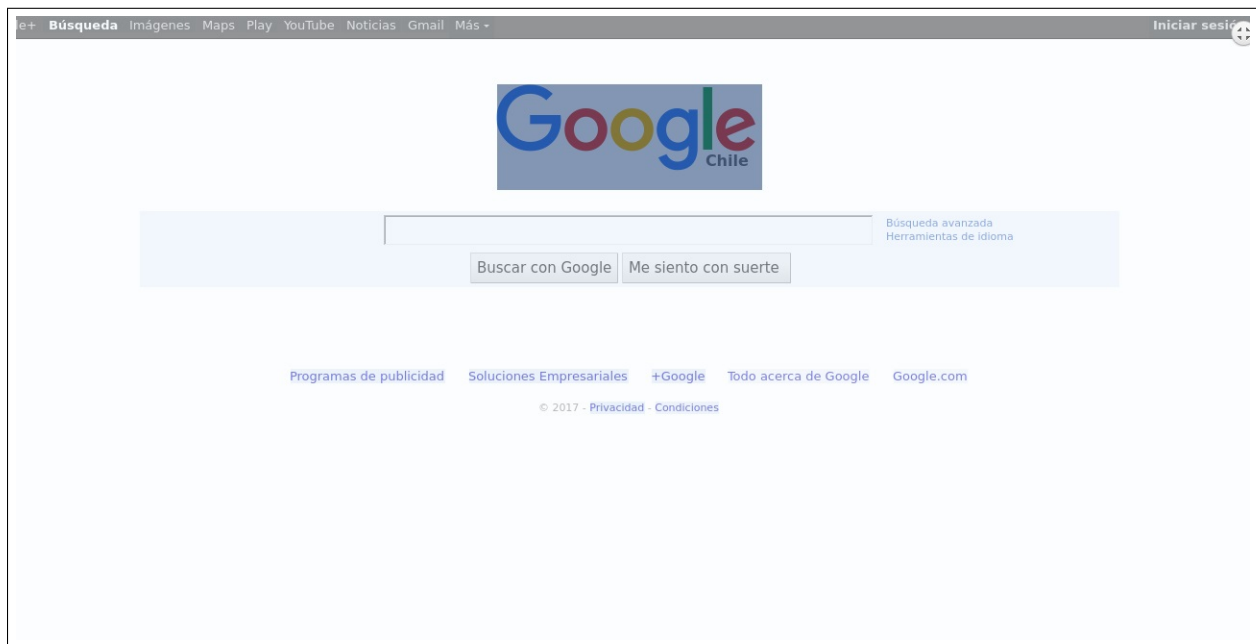


Figura 13: Mapa de Objetos Claves en escala de azul.



Figura 14: Mapa de Objetos Claves en escala de verde.

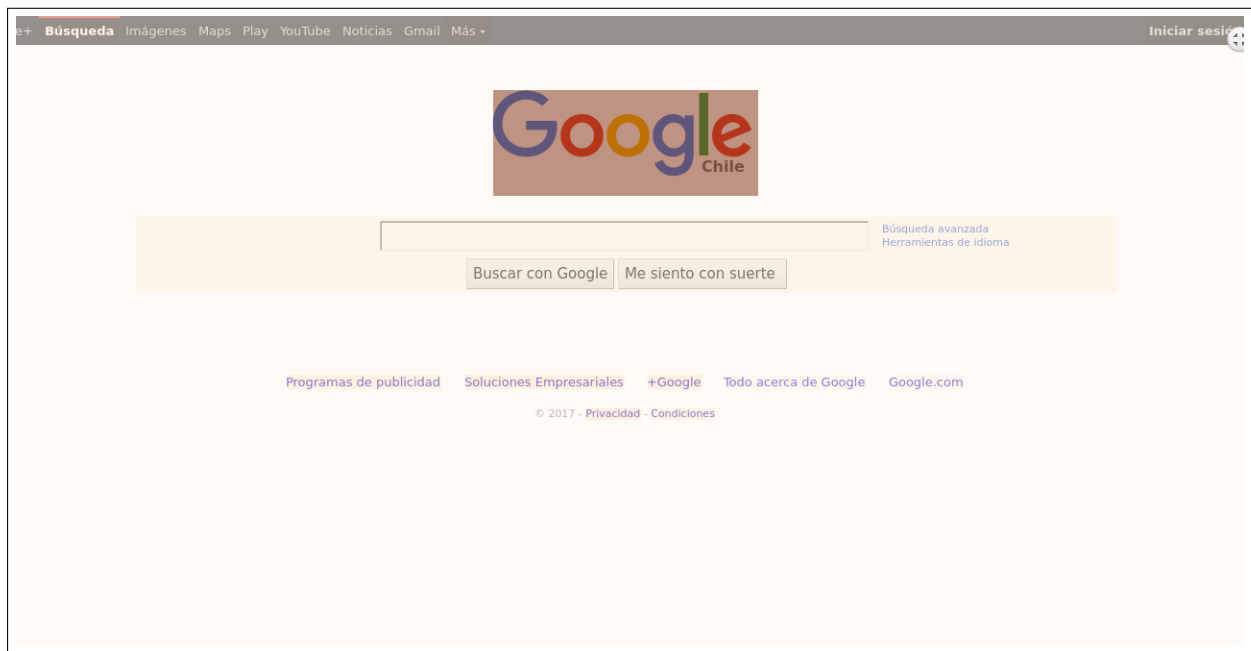


Figura 15: Mapa de Objetos Claves en escala de naranja.

Referencias

- [1] *Sitio Web del WIC* <http://www.wic.uchile.cl>
- [2] *Página Oficial del Proyecto AKORI* <https://www.akoriproject.cl>
- [3] *Sitio oficial de lenguaje de programación Python* <https://www.python.org/>
- [4] *PyCharm IDLE* <https://www.jetbrains.com/pycharm/>
- [5] *El framework Django* <https://www.djangoproject.com/>
- [6] *Biblioteca SELENIUM* <http://www.seleniumhq.org/>
- [7] *Patrón Modelo Vista Controlador* <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- [8] *Definición del término Web scraping* https://es.wikipedia.org/wiki/Web_scraping
- [9] *Publicación del WIC donde son definidos los objetos web* <http://wic.uchile.cl/wp-content/uploads/2015/08/Eye-tracking-and-EEG-features-for-salient-Web-object-identification.pdf>
- [10] *Definición elementos del DOM* https://es.wikipedia.org/wiki/Document_Object_Model
- [11] *Definición de un Screenshot* <https://en.wikipedia.org/wiki/Screenshot>
- [12] *Escala RGB* <https://es.wikipedia.org/wiki/RGB>
- [13] *Biblioteca de generación de gráficos* <https://matplotlib.org/>
- [14] *Objeto XMLHttpRequest* <https://es.wikipedia.org/wiki/XMLHttpRequest>
- [15] *Biblioteca JQuery* <https://jquery.com/>