



INGENIERÍA EN SISTEMAS AUDIOVISUALES Y
MULTIMEDIA

Curso Académico 2019/2020

Trabajo Fin de Grado

(INCOMPLETO) TÍTULO DEL TRABAJO EN
MAYÚSCULAS

Autor : Ángel Perea Arias

Tutor : Dr. Jose María Cañas Plaza

Co-tutor : Dr. David Roldán Álvarez

Trabajo Fin de Grado

(INCOMPLETO) Título del Trabajo con Letras Capitales para Sustantivos y
Adjetivos

Autor : Ángel Perea Arias

Tutor : Dr. Jose María Cañas Plaza

Co-Tutor : Dr. David Roldán Álvarez

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2020, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2020

*Dedicado a
mi familia y amigos*

Agradecimientos

(INCOMPLETO)

Resumen

(INCOMPLETO)

Summary

(INCOMPLETO)

Índice general

1. Introducción	1
1.1. Estructura de la memoria	1
2. Objetivos	3
3. Infraestructura utilizada	5
3.1. Tecnologías Web	5
3.1.1. HTML	5
3.1.2. CSS	6
3.1.3. JavaScript	7
3.1.4. Python	7
3.1.5. Django	8
3.2. Tecnologías de bases de datos	9
3.2.1. MongoDB	9
3.2.2. Elasticsearch	10
3.3. Tecnologías de visualización	11
3.3.1. Matplotlib	11
3.3.2. Kibana	12
4. Analíticas basadas en MongoDB y Matplotlib	15
4.1. Antecedentes	15
4.2. Diseño	19
4.3. Implementación	20
4.3.1. MongoDB en Webserver	20
4.3.2. Matplotlib en Webserver	23

4.4. Validación experimental	27
5. Analíticas basadas en Elasticsearch y Kibana	31
5.1. Diseño	31
5.2. Implementación	32
5.2.1. Elasticsearch en Webserver	33
5.2.2. Kibana en Webserver	38
5.3. Validación experimental	41
6. Conclusiones	51
6.1. Conclusiones finales	51
6.2. Competencias adquiridas	51
6.3. Trabajos futuros	51
Bibliografía	53

Índice de figuras

3.1. Tecnologías web básicas.	5
3.2. Patrón MVT Django.	9
3.3. Stack ELK.	10
3.4. Visualizaciones en Matplotlib.	12
3.5. Visualizaciones en Kibana.	13
3.6. Selector de espacios de trabajo en Kibana.	14
4.1. Editor y simulador en Kibotics.	16
4.2. Arquitectura Kibotics.	17
4.3. Arquitectura Kibotics con funcionalidad de analíticas mejorada usando Mon- goDB y una vista de datos agregados basada en Matplotlib.	20
4.4. Analíticas del primer prototipo parte 1.	28
4.5. Analíticas del primer prototipo parte 2.	29
5.1. Arquitectura Kibotics segundo prototipo.	32
5.2. API Rest Elasticsearch.	37
5.3. Creación de índices en Kibana.	38
5.4. Sección <i>Discover</i> en Kibana.	39
5.5. Menú creación de visualizaciones.	40
5.6. Menú de selección de analíticas en Kibotics	41
5.7. Histograma	42
5.8. Mapa de calor sesiones y simulaciones	42
5.9. Gráfico de barras por día de la semana	43
5.10. Gráfico de barras por hora del día	43

5.11. Gráfico de barras para tiempo total y medio en simulaciones	44
5.12. Mapa geográfico de sesiones	44
5.13. Gráficas circulares para SO, Dispositivo y Navegador	45
5.14. Últimos eventos logueados	45
5.15. Kibana en Kibotics parte 1	46
5.16. Kibana en Kibotics parte 2	47
5.17. Kibana en Kibotics parte 3	47
5.18. Exportación en interfaz Kibana.	50

Capítulo 1

Introducción

(INCOMPLETO)

informacion circunstancial vs información estructural otras plataformas digitales guardado
masivo de datos visualizacion

1.1. Estructura de la memoria

(INCOMPLETO)

Capítulo 2

Objetivos

(INCOMPLETO) **desarrollar la funcionalidad analíticas en una plataforma educativa**

1 expandir los datos grabados 2 grabarlos en una base de datos moderna 3 que se visualicen automáticamente

primera solución - primer prototipo sondas mongo matplotlib

Capítulo 3

Infraestructura utilizada

En este capítulo se describen las diferentes tecnologías web, de bases de datos y de visualización que se han utilizado en el transcurso del proyecto.

3.1. Tecnologías Web

El desarrollo de sitios web o aplicaciones *online* es un ámbito muy amplio con un gran abanico de tecnologías a disposición de los desarrolladores. En esta sección se explorarán las tecnologías web envueltas en el desarrollo del proyecto tanto en el lado cliente como servidor. En la figura 3.1 se muestran las 3 tecnologías web básicas utilizadas.

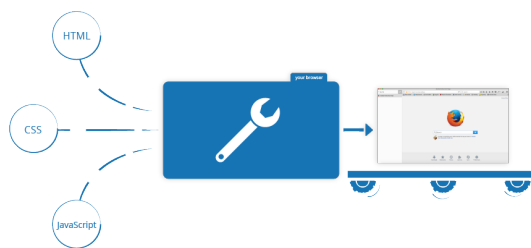


Figura 3.1: Tecnologías web básicas.

3.1.1. HTML

HTML (Hipertextual Markup Language) [1], es un lenguaje de marcado. Actualmente utilizado para la definición de estructura básica de los contenidos de una página web como vídeos, gráficos, texto.

Publicado en 1991, su historia se remonta a 1980, cuando Tim Berners-Lee propuso un nuevo sistema para compartir ficheros. Actualmente se ha impuesto como el lenguaje estándar para dar formato a documentos, definido por el *World Wide Web Consortium* (W3C), el cual ha ido evolucionando versión a versión adoptando todas las nuevas exigencias que ofrecen las webs actuales, tanto en el campo de los recursos multimedia como en el de interactividad.

HTML se desarrolla haciendo uso de una estructura de etiquetas o *tags*, dentro de las cuales se pueden incluir cada uno de los elementos que conforman una página web. Dispone de cierta capacidad para aportar estilo y lógica pero estas generalmente se delegan en CSS y JavaScript respectivamente.

La última versión oficial es HTML5, la cual proporciona soporte nativo para audio y vídeo, inclusión de la etiqueta `canvas` usada para generar gráficos y efectos tanto en 2D como en 3D, entre otras mejoras. Es la versión usada en este proyecto.

3.1.2. CSS

CSS (Cascade Style Sheet), es un lenguaje de reglas en cascada utilizado para dotar de diseño gráfico a elementos de una página web. Define, como se mencionó anteriormente, la estética de un documento HTML y por lo tanto de una página Web.

Permite crear webs atractivas y responsivas, que se adapten al dispositivo en que están siendo vistas, ya sea por ejemplo, tabletas, ordenadores o móviles.

Permite mover todas las reglas de estilo (tamaños de fuente o de imágenes, responsividad de elementos a ciertas resoluciones...) a ficheros `*.css`, evitando así redundancia en un mismo documento `*.html`, mejorando así la modularidad e independencia dentro de un proyecto.

La última versión es CSS3, añade novedades como soporte nativo para transiciones y animaciones. Además de herramientas para una maquetación más precisa. Es la versión usada en este proyecto.

3.1.3. JavaScript

JavaScript [2] es un lenguaje de programación ligero, interpretado, orientado a objetos y dinámico. Utilizado principalmente como lenguaje de *scripting* para paginas web, en este campo su papel principal se centra en el desarrollo de lógica en la parte del cliente *backend*: acceso al *Document Object Model*(DOM) de la web, modificación de etiquetas HTML, generación de gráficos en *canvas* o gestión de *cookies*.

Permite crear nuevo contenido dinámico, así como controlar archivos multimedia y gracias al uso de API's (Aplication Programming Interface), enriquece a JavaScript con más funcionalidades permitiendo crear funcionalidades más complejas con desarrollos más simples.

3.1.4. Python

Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel. Diseñado para un desarrollo de aplicaciones rápido, se utiliza como lenguaje de *scripting* y conexión entre otros componentes de un sistema.

Python es simple, con una sintaxis fácil de aprender centrada en la legibilidad del código, consiguiendo así reducir el coste del desarrollo, mantenimiento y ampliación de proyectos.

Tiene una gran biblioteca de librerías que puede ser fácilmente extendida por módulos personalizados escritos en C/Python. Haciendo uso del instalador de paquetes `PIP` (Python Package Installer), es posible la instalación e integración de paquetería en proyectos de manera muy sencilla, así como el cambio de versiones de las mismas.

Pese a no ser estrictamente hablando una tecnología web, en este proyecto lo hemos empleado en el contexto de Django, para programar la lógica *backend*. El proyecto comenzó a desarrollarse en Python 2.6 y ha terminado en la versión Python 3.6.9.

3.1.5. Django

Django es un entorno Web de alto nivel diseñado para desarrollar servidores en Python. Al igual que este, su filosofía se centra en desarrollos rápidos, limpios y en un diseño pragmático. Sigue el patrón *Model-View-Template* (MVT) [3], donde:

- *Model*, esta capa del patrón tiene toda la información relativa a las bases de datos: cómo se almacenan, cómo se relacionan entre ellas, cómo validarlas... Manejado por la capa de bases de datos de Django. Toda esta información de configuración se desarrolla y almacena en el fichero `Models.py`. Mediante su ORM (Object Relational Model) permite al desarrollador del servidor hacer consultas y manejar los datos de bases de datos relacionales en Python. Los filtros, desarrollados en Python, se traducen automáticamente peticiones SQL.
- *View*, parte lógica del Framework, se puede ver como una unión entre la capa de modelo y plantillas. Formado por dos ficheros: `urls.py`, encargado de llamar a la vista adecuada dependiendo de la URL a la que se acceda y `views.py` con todas esas vistas que devolverán una respuesta HTTP y en las cuales se consultará la capa Model si fuese necesario.
- *Template*, sección que se encargará del qué y cómo mostrar los datos. Manejado por vistas y plantillas de Django que servirán de bases para la parte *frontend* de la Web. Guardado en documentos HTML enriquecidos junto a variables de plantillas Django (`{{ nombre_de_variable }}`), las cuales permite el uso de, por ejemplo, bucles, operaciones condicionales, diccionarios, inserción de bloques... para generar webs enriquecidas y dinámicas en muy pocas líneas de código.

En la siguiente figura 3.2 se pueden ver gráficamente los ficheros python involucrados en el patrón MVT así como las plantillas HTML enriquecidas que Django utiliza.

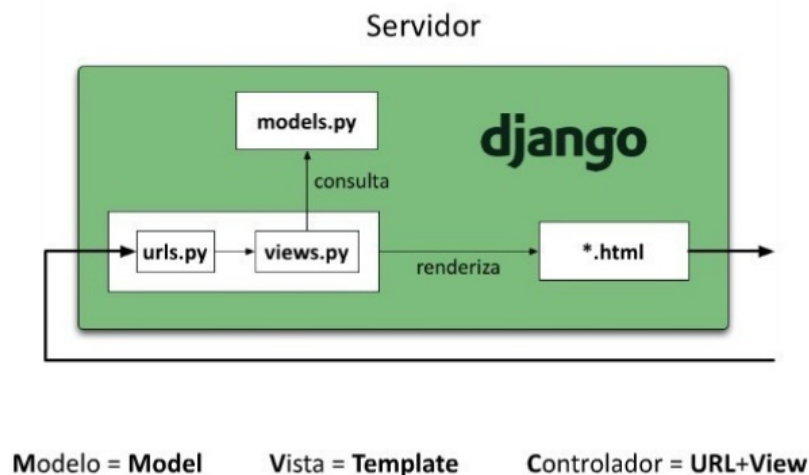


Figura 3.2: Patrón MVT Django.

Diseñado para ser seguro, y evitar errores comunes, Django ofrece una fácil administración de las cuentas de usuarios, así como de sus contraseñas. Además de una sección de administración en la que poder gestionar toda la información almacenada en las bases de datos desplegadas.

El software ya existente de Kibotics Webserver, servicio web sobre el que se ha trabajado, ya estaba basado en Django. El proyecto comenzó a desarrollarse en la versión 1.9 y ha concluido en la Django 1.11 [4].

3.2. Tecnologías de bases de datos

Una base de datos es un conjunto de información, la cual pertenece a un mismo contexto, almacenada de modo sistemático y preservada en distintos formatos, puede ser consultada o alterada posteriormente. En esta sección, se detallarán las distintas tecnologías de bases de datos utilizadas en el desarrollo de este proyecto.

3.2.1. MongoDB

MongoDB [5] es una base de datos NoSQL (Not only SQL) distribuida, documental (almacenando la información en ficheros BSON, muy similares a JSON), de código abierto y

diseñada para ofrecer un nivel productivo alto.

Al ser una base de datos NoSQL, permite almacenar información de forma estructurada, pero flexible, pues los esquemas pueden ser cambiados sin necesidad de parar el servicio.

Debido a esta estructura, la velocidad en las consultas es muy alta, convirtiéndose así en una base de datos ideal para trabajar con grandes cantidades de información que vayan a ser consultados muy frecuentemente.

La escalabilidad de MongoDB es sencilla, puesto que se ejecuta en *clusters*, podrá escalar horizontalmente contratando más máquinas, aumentando así la capacidad de procesamiento. Es una base de datos muy utilizada en la industria [6].

Para el primer prototipo de este proyecto, se ha utilizado la última versión estable de MongoDB, la versión 4.2.6.

3.2.2. Elasticsearch

Elasticsearch [7] es una base de datos. Junto a Logstash y Kibana, los tres proyectos open source, forman el stack ELK. En la siguiente figura 3.3 se muestra un diagrama de flujo de estas tecnologías involucradas en el stack ELK.

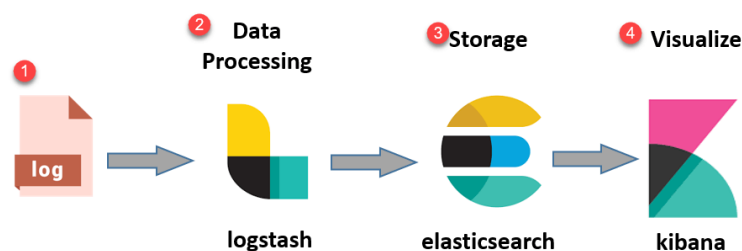


Figura 3.3: Stack ELK.

Basado en Lucene (API para recuperación de información), permite almacenar información compleja como datos de geolocalización así como realizar búsquedas de texto y auto-completado casi en tiempo real. Mediante el uso de una API Rest realiza consultas, borrado y

actualización de documentos.

En vez de usar filas y columnas, Elasticsearch almacena estructuras complejas de datos serializadas como documentos JSON. Para almacenar estos documentos, Elasticsearch hace uso de índices. Cada uno de estos índices dispone de una estructura de datos propia. Estas estructuras JSON son también utilizadas tanto para las consultas y respuestas, lo que la hace sencilla de usar e integrar en sistemas productivos ya existentes.

Dadas estas capacidades de almacenar información preparada en índices, la consulta de documentos es muy ágil puesto que evita búsquedas tanto en índices como en documentos no deseados. Organizado en nodos, permitirá aumentar la potencia a medida que la demanda de recursos crezca.

Se ha convertido en uno de los buscadores por texto más importantes, utilizado por gigantes de Internet como Facebook, Netflix o Github.

La última versión estable de Elasticsearch es la versión 7.8.0 [8], liberada el 18 de Junio del 2020. En proyecto, se ha utilizando la versión 7.6.2 publicada el 31 de Marzo del 2020.

3.3. Tecnologías de visualización

La muestra de datos es una de las bases de este proyecto, en esta sección se explicarán las tecnologías utilizadas tanto en el primer prototipo como en la versión final del módulo de analíticas desarrollado para este proyecto.

3.3.1. Matplotlib

Matplotlib es una librería de Python que se encarga de la generación de visualizaciones tanto estáticas como animadas.

Proporciona gran variedad de visualizaciones como mapas de calor, gráficas de barras, histogramas... recordando a Matlab. Ofrece cierta capacidad de estilo y puede ser utilizada junto

a otras librerías para generar visualizaciones aún más complejas y enriquecidas como mapas geográficos en los que se representa datos mediante latitud y longitud.

En la siguiente figura 3.4 se muestran unos ejemplos de diferentes tipos de visualizaciones que Matplotlib ofrece.

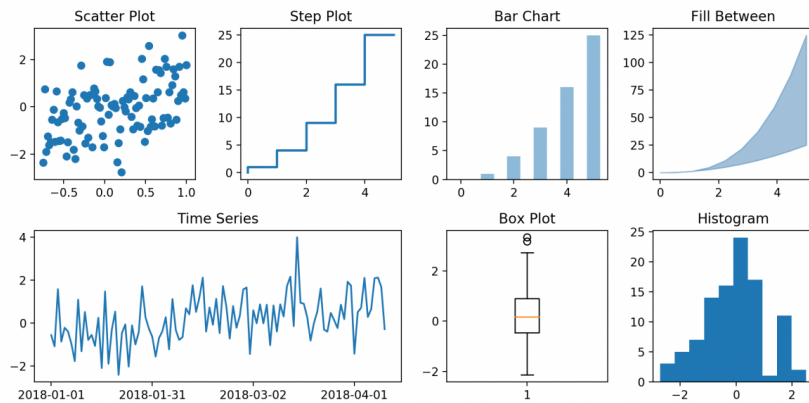


Figura 3.4: Visualizaciones en Matplotlib.

Matplotlib almacena las visualizaciones en figuras, cada una contiene los ejes en que se representarán los datos, estos ejes pueden ser de múltiples tipos, ya sean coordenadas x-y, x-y-x para una representación en tres dimensiones o un eje polar.

Las visualizaciones generadas podrán ser mostradas en una nueva ventana si utilizamos la librería en un *script* o ser renderizadas y devueltas como imagen PNG para su posterior muestra en el servicio web haciendo uso de la etiqueta HTML ``

En el primer prototipo de este proyecto se utilizó la versión 3.1.2 de Matplotlib, publicada el 18 de Marzo del 2020 [9].

3.3.2. Kibana

Como se comentó anteriormente, el stack ELK está compuesto por Kibana [10] como motor de búsqueda, procesador de datos y generador de visualizaciones entre otras funcionalidades. Diseñada para utilizar Elasticsearch como fuente de datos.

Gracias a su aplicación *frontend*, la creación de visualizaciones se simplifica mucho sin ser necesaria la codificación de estas.

Mediante configuración, filtrado y selección de los datos indexados en Elasticsearch se pueden crear múltiples tipos de visualizaciones interactivas (gráficos de barras, circulares, tablas, histogramas y mapas), y posteriormente ser agrupadas en tableros o *Dashboards*, los cuales permiten la visualización y posterior filtrado de grandes cantidades de información de forma simultanea y sencilla.

En la siguiente figura 3.5 se muestran unos ejemplos de diferentes tipos de visualizaciones que Kibana ofrece.

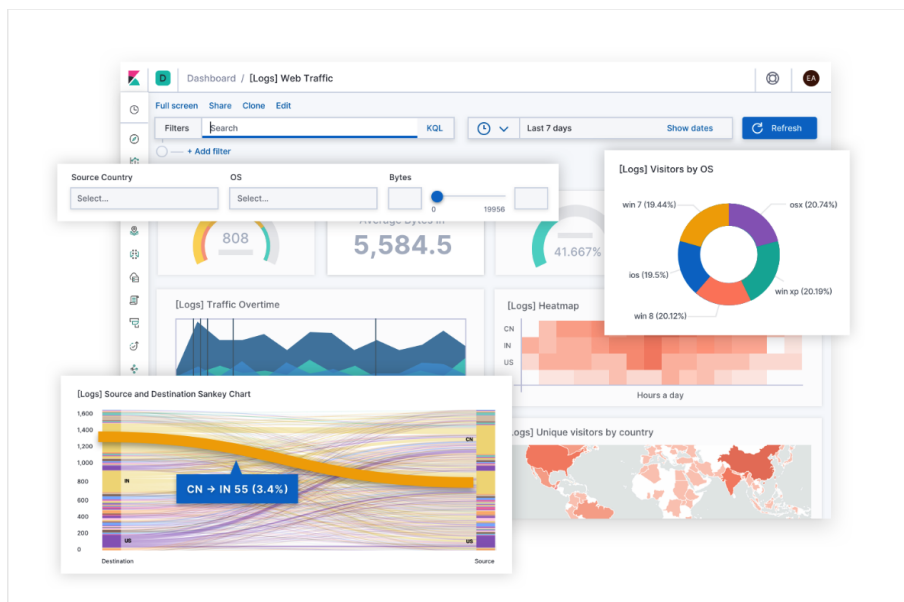


Figura 3.5: Visualizaciones en Kibana.

Permite el procesado de los documentos ya indexados en Elasticsearch para crear nuevos campos dinámicos que podrán ser utilizados y representados posteriormente en visualizaciones y estadísticas.

Ofrece una interfaz segura y organizada, dividida en espacios de trabajo o *spaces* como se muestra en la figura 3.6. Estos espacios pueden ser tratados como instalaciones de Kibana independientes permitiendo a varios equipos de trabajo hacer uso del mismo servicio de Kibana sin

los inconvenientes de compartir información de índices. Aumentando la compartimentalización y escalabilidad del servicio.

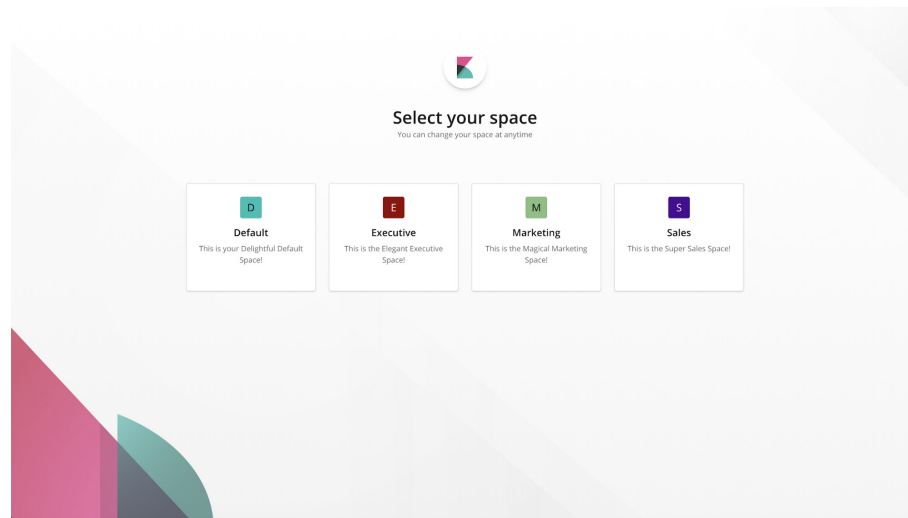


Figura 3.6: Selector de espacios de trabajo en Kibana.

La última versión estable de Kibana es la 7.8.0 [11], liberada el 18 de Junio del 2020. En la versión final desarrollada se ha utilizado Kibana 7.7.0 publicada el 13 de Mayo del 2020.

Capítulo 4

Analíticas basadas en MongoDB y Matplotlib

En este capítulo se describe el estado inicial de la plataforma Kibotics en cuanto a recogida automática de datos y su análisis, tanto arquitectura de la aplicación como las tecnologías en las que está desarrollado. Además, se define el diseño que se ha implementado en el primer prototipo que mejora la funcionalidad de análisis automático en la plataforma Kibotics. Se exponen los pasos realizados para la integración de MongoDB como base de datos, y de Matplotlib como generador de visualizaciones de la herramienta de analíticas que se integrará en Kibotics Webservice.

4.1. Antecedentes

Kibotics es una plataforma educativa *online*, en la que se enmarca este proyecto. Ofrece herramientas centradas en la docencia en robótica y programación para alumnos de secundaria. Con una gran variedad de ejercicios en los que los alumnos pueden aprender conceptos básicos acerca de distintos lenguajes de programación como Scratch o Python, así como introducirse a la visión artificial o la simulación en robots. Estos ejercicios tienen dos secciones principales:

1. Una primera vista con contenidos teóricos, incluye una explicación del ejercicio así como conceptos útiles en la resolución del mismo.
2. La segunda vista es donde los alumnos desarrollen su solución. Dividida en el editor y

en el simulador WebSim, como se muestra en la Figura 4.1. El simulador web ejecuta el código desarrollado en el editor de texto.

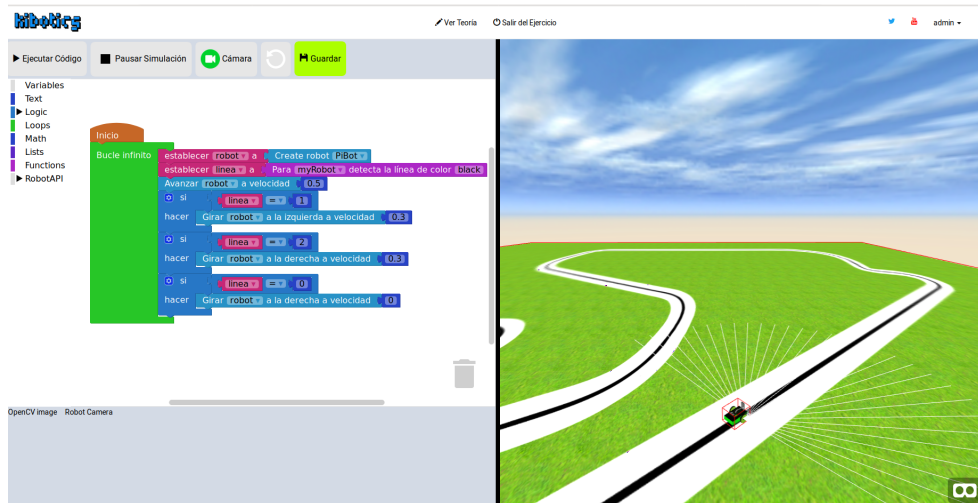


Figura 4.1: Editor y simulador en Kibotics.

Una vez resuelto el ejercicio, Kibotics genera un fichero con la solución que el usuario puede descargar para probarlo en un robot real. En el siguiente enlace se muestra un vídeo de un robot sobre el que se ha cargado código desarrollado en Kibotics.¹

Kibotics es un servicio web complejo y compuesto de muchas tecnologías distintas. La Figura 4.2 muestra la arquitectura que poseía Kibotics 2.0 inicialmente, en ella podemos observar:

- **Webserver:** Servidor web desarrollado en Django, es el centro de Kibotics y donde se generan los logs sobre los que se trabaja en este Trabajo Fin de Grado. Se apoya en una base de datos MySQL para almacenar todos los datos de información estructural de la aplicación y en unos ficheros de log *.txt en los que se almacena información circunstancial de uso de la aplicación.
- **Websim:** Simulador robótico que se ejecuta únicamente en el lado del cliente. Desarrollado en A-Frame, HTML5, JavaScript. Permite a los usuarios aprender los fundamentos de la programación robótica y visión artificial.

¹<https://www.youtube.com/watch?v=a0aIqyyEEnw>

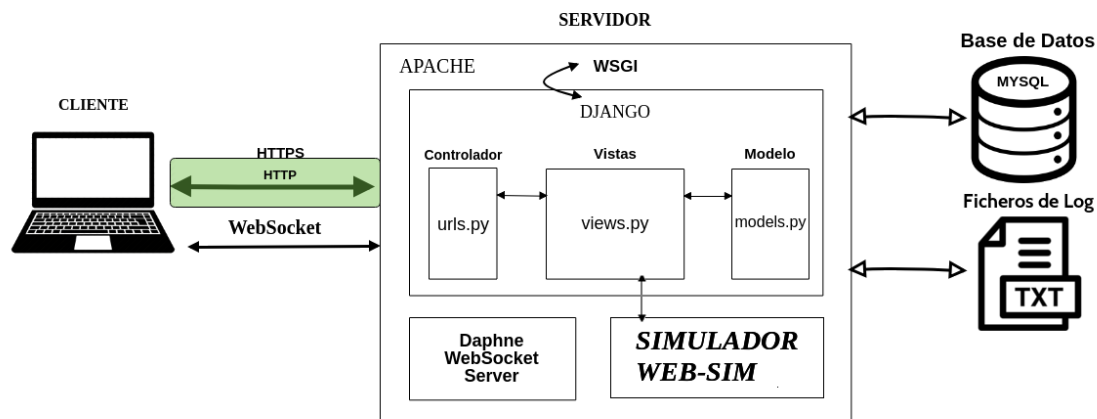


Figura 4.2: Arquitectura Kibotics.

Kibotics 2.0 necesita de datos para ofrecer su servicio, estos datos se pueden dividir en dos tipologías:

- Estructurales: almacenados en la base de datos MySQL, ofrecen información acerca de los usuarios registrados, ejercicios de la aplicación, permisos de acceso a estos ejercicios, etc.
- Circunstanciales: guardados inicialmente en ficheros de texto, proporcionan información acerca de eventos ocurridos en la aplicación por los usuarios como inicio de sesión, fin de sesión, inicio de un ejercicio o salida de un ejercicio. Son los datos que se explotarán en este Trabajo Fin de Grado.

Los datos circunstanciales o logs generados por la aplicación Django se guardan en una serie de archivos indexados en el servidor con formato `yyyy-MM-dd-log.txt`. Teniendo así un fichero por día con todos los eventos registrados.

Inicialmente Kibotics almacenaba 5 eventos distintos: entrada a la plataforma, salida de la plataforma, comienzo de ejercicio, fin de ejercicio y error genérico de la aplicación.

Esta metodología de registro de datos circunstanciales disponía de un sistema numeral de códigos para identificar el evento que había generado cada registro de log. Cada uno de estos registros estaba separado por la cadena de caracteres: " | ". Estos códigos numéricos y su

estructura son los siguientes:

- Log in: "1 | date | user name | user IP | HTTP_USER_AGENT"
- Log out: "2 | date | user name | user IP | HTTP_USER_AGENT"
- Comienzo ejercicio: "3 | date | user name | user IP | simulation type | exercise ID | host IP | HTTP_USER_AGENT"
- Fin ejercicio: "4 | date | user name | user IP | simulation type | exercise ID | host IP | HTTP_USER_AGENT"
- Error 500: "5 | 500 Internal Server Error"

Estos eventos logueados se generaban en el servidor Django haciendo uso de sondas programadas en Python, las cuales registran el evento en distintas partes del código del servidor web para guardarlas en los ficheros. Un ejemplo de estas sondas para el registro de un evento de *log in* es:

```
log = open(DIRECTORY + "/logs/" + str(date.today()) + "-log.txt", "a")

traze = "1 | " + str(datetime.now().strftime("%d/%m/%Y %H:%M:%S"))
+ " | " + username + " | " + client\_ip + " | " + user\_agent + "\\n"

log.write(traze)

log.close()
```

Además de estos logs generados en Django, se disponía de los generados de forma automática por Apache. Apache separa los eventos registrados en dos ficheros:

- Un archivo con la salida general de la aplicación, asociada a los *prints* y excepciones producidas en el servidor.
- Un archivo de acceso al servidor que muestra las peticiones HTTP que este ha recibido.

Inicialmente en Kibotics 2.0 no se explotaban estos ficheros de log extraídos del servicio Django, simplemente se almacenaban sistemáticamente en el servidor y se consultaban a mano si la ocasión lo requería.

4.2. Diseño

Para el desarrollo de este primer prototipo de la herramienta de análisis integrada en Kibotics el primer paso es realizar un diseño con las tecnologías a usar.

Kibotics Webserver disponía de una tecnología primitiva de trazabilidad de logs basada en ficheros TXT con eventos limitados. La apertura y cierre constante de estos ficheros acarrea un gran coste computacional, limitando así la explotación masiva de estos datos para la generación de estadísticas útiles.

Kibotics es una plataforma digital que pretende dar servicio a una gran cantidad de usuarios. Por lo tanto, se espera que la capacidad de procesamiento, almacenamiento y consulta de los logs aumente. Haciendo uso de ficheros de texto plano no se podrá conseguir la velocidad de procesamiento necesaria.

Es por esto, por lo que en un primer prototipo se decidió cambiar a un motor de bases de datos para los datos circunstanciales. Se ha decidido explorar MongoDB, es una base de datos NoSQL que permite almacenar información de forma estructurada tal y como se vio en el capítulo 3. Debido a esto, la velocidad en las consultas es muy alta aún con gran cantidad de datos. Ya que se espera almacenar información de forma masiva, MongoDB es una buena opción para la herramienta de análisis a desarrollar en este primer prototipo.

Para que Kibotics almacene la información de eventos será necesario modificar las sondas ya existentes en Kibotics Webserver que almacenaban información en ficheros de texto plano TXT para que guarden los datos en MongoDB. Además, se ampliará el número de eventos de los que se guarda información en la base de datos circunstanciales.

En este primer prototipo además se desarrollará e incorporará a la plataforma la capacidad de visualización de los datos circunstanciales generados, vistos de manera agregada. Para visualizar estos datos almacenados se usará Matplotlib. Es una librería de Python por lo tanto la integración en Django será simple, se desarrollará toda la funcionalidad en un fichero Pyt-

hon llamado `analitics.py` y se creará una vista específica de Django para mostrar esas visualizaciones. Es a este al que Django accederá para la generación de las visualizaciones que posteriormente se enviarán a las plantillas HTML enriquecidas de Django.

En la Figura 4.3 se muestra de forma esquemática los cambios que se han explorado en este primer prototipo en Kibotics Webserver, tanto el cambio en las sondas para almacenar los datos de log en MongoDB como el uso de Matplotlib como herramienta principal de generación de visualizaciones automáticas.

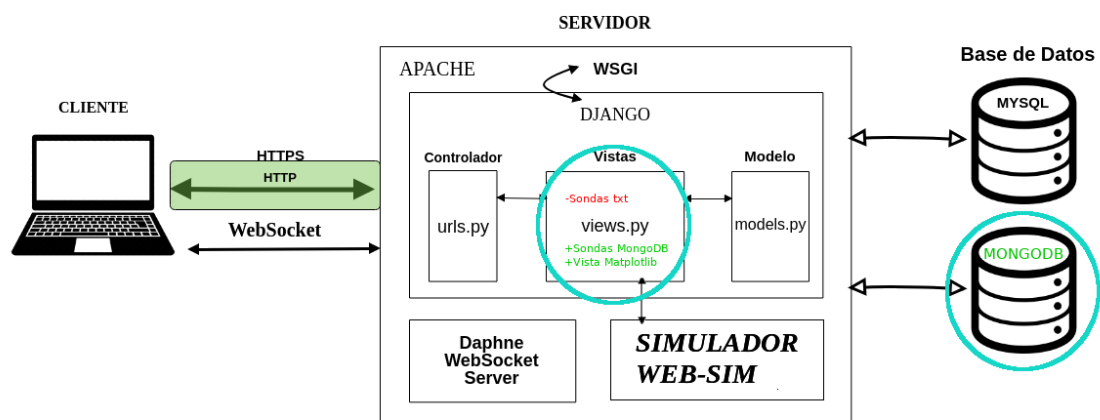


Figura 4.3: Arquitectura Kibotics con funcionalidad de analíticas mejorada usando MongoDB y una vista de datos agregados basada en Matplotlib.

4.3. Implementación

En esta sección se detallan los pasos que ha sido necesario realizar para el desarrollo software del primer prototipo de la herramienta de análisis integrada en Kibotics. Se divide en la integración de MongoDB como base de datos NoSQL de logs y de Matplotlib como generador de visualizaciones.

4.3.1. MongoDB en Webserver

Para la integración de MongoDB en Kibotics Webserver primero se realizó una instalación local del servicio MongoDB para verificar la viabilidad de la tecnología en este primer prototipo-

po. Para ello se ejecuta la siguiente sentencia:

```
$ sudo apt-get install mongodb-org
```

Una vez instalado, se levanta el servicio MongoDB:

```
$ sudo systemctl start mongod
```

Adicionalmente a esto, podemos reiniciar o parar el servicio con las siguientes sentencias respectivamente:

```
$ sudo systemctl restart mongod
```

```
$ sudo systemctl stop mongod
```

Con la base de datos MongoDB ya instalada y configurada el siguiente paso es migrar las sondas previamente definidas. Estas sondas, pasarán de escribir en ficheros a registrar los eventos en MongoDB.

Haciendo uso de la librería `pymongo` para comunicarnos con la base de datos, la tarea se simplifica mucho. Primero importamos la librería y abrimos la conexión con la base de datos, para ello:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["kiboticsDDBB"]
```

Ya con las conexiones necesarias realizadas, como se puede observar en el siguiente código Python, cada una de las sondas almacenará la información de log en una tabla distinta. Las sondas se transforman a, por ejemplo, las de nueva sesión y nueva simulación:

```
# Nueva sesión
mydict = {
    "date" : datetime_object_test,
    "username" : "USERNAME_TEST",
    "client_ip" : "CLIENT_IP_TEST",
    "user_agent" : "USER_AGENT_TEST"
}
mydb["newSession"].insert_one(mydict)
```

```
# Nueva simulación
mydict = {
    "date" : datetime_object_test,
    "username" : "USERNAME_TEST",
    "client_ip" : "CLIENT_IP_TEST",
    "simulation_type" : "SIMULATION_TYPE_TEST",
    "exercise_id" : "EXERCISE_ID_TEST",
    "host_ip" : "HOST_IP_TEST",
    "container_id" : "CONTAINER_ID_TEST",
    "user_agent" : "USER_AGENT_TEST"
}
mydb["newSimulation"].insert_one(mydict)
```

Con estos pasos, el registro de eventos de log en la nueva base de datos MongoDB ya está completo. Para recuperar la información y tratarla en el servidor se hará uso, una vez más, de pymongo. Las sentencias o *queries* de búsqueda para los diferentes eventos logueados serán:

```
# Sentencia nueva sesión
dataNSES = mydb["newSession"].find({
    "username" : {'$regex' : "USERNAME_TEST"},
    "date" : {'$lte': first_day_test, '$gte': last_day_test}
});

# Sentencia fin de sesión
dataESES = mydb["endSession"].find({
    "username" : {'$regex' : "USERNAME_TEST"},
    "date" : {'$lte': first_day_test, '$gte': last_day_test}
});

# Sentencia nueva simulación
dataNSIM = mydb["newSimulation"].find({
    "username" : {'$regex' : "USERNAME_TEST"},
    "date" : {'$lte': first_day_test, '$gte': last_day_test}
});

# Sentencia fin de simulación
dataESIM = mydb["endSimulation"].find({
```

```
"username" : {'$regex' : "USERNAME_TEST"},
"date" : {'$lte': first_day_test, '$gte': last_day_test}
});
```

Como se puede observar, estas sentencias de búsqueda filtran tanto por usuarios como por rangos de fechas. Esto aporta cierta flexibilidad en la obtención de los eventos, evitando así tener que recorrer datos o ficheros extra descartando registros de log, como se haría con la metodología de ficheros que poseía Kibotics inicialmente.

4.3.2. Matplotlib en Webserver

Matplotlib es una librería Python de generación de visualizaciones tanto estáticas como animadas tal y como se presentó en el capítulo 3. Haciendo uso de ella, se han generado todas las visualizaciones necesarias para el primer prototipo de este Trabajo Fin de Grado.

Estas visualizaciones inicialmente se han separado en dos secciones, analíticas de simulaciones y analíticas de sesiones. Ambas pueden ser filtradas tanto por usuarios como por rangos de fechas.

Siguiendo la metodología detallada en las secciones anteriores surgió un problema, los datos de inicio y fin, tanto para las sesiones como para las simulaciones, están separados en distintas tablas de MongoDB. Por lo tanto, para relacionar ambos eventos es necesario cruzarlos en Python para unificarlos en un único evento de sesión o simulación.

Para solventar esta problemática se desarrolló un método con esta funcionalidad de unificación de registros, con el que se consigue extraer información valiosa de duración de eventos:

Este código fuente extrae todos los usuarios contenidos en el campo `username` y posteriormente recorre los registros de apertura de cada uno de ellos. Buscando por el campo `date` de los eventos de cierre hasta encontrar la inmediatamente posterior que será almacenado junto con la duración del evento.

Como salida del método, se devolverá un diccionario de diccionarios con cada uno de los eventos de sesión/simulación para cada usuario del que haya ocurrencias.

```

def formatDatesUser(newData, endData):

    users = newData.distinct("username")
    newData.sort([('Username', -1), ('date', -1)])
    endData.sort([('Username', -1), ('date', -1)])
    Dict = {}

    for user in users:
        for d in newData:
            for dd in endData:
                if(dd['username'] == d['username'] == user and \
                   d['date'] < dd['date']):
                    if(user not in Dict):
                        Dict[user] = {d['date'] :
                                      {
                                          "totalTime" : dd['date']-d['date'],
                                          "endTime" : dd['date']
                                      }
                                      }
                    else:
                        Dict[user].update({d['date'] :
                                           {
                                               "totalTime" : dd['date']-d['date'],
                                               "endTime": dd['date']
                                           }
                                           })
            break;
        endData.rewind()
    newData.rewind()

    return Dict

```


La clave de este diccionario serán los usuarios. El valor será otro diccionario con las fechas de comienzo y fin del evento así como su duración. Un ejemplo de respuesta es:

```
{
  "USERNAME_TEST_1" : {
    start_date_1 : {
      "endTime" : end_date_1,
      "totalTime" : end_date_1 - start_date_1
    },
    start_date_2 : {
      "endTime" : end_date_2,
      "totalTime" : end_date_2 - start_date_2
    },
    ...
    start_date_N : {
      "endTime" : end_date_N,
      "totalTime" : end_date_N - start_date_N
    },
  },
  ...

  "USERNAME_TEST_N" : {
    start_date_1 : {
      "endTime" : end_date_1,
      "totalTime" : end_date_1 - start_date_1
    },
    start_date_2 : {
      "endTime" : end_date_2,
      "totalTime" : end_date_2 - start_date_2
    },
    ...
    start_date_N : {
      "endTime" : end_date_N,
      "totalTime" : end_date_N - start_date_N
    },
  },
}
}
```

Una vez se ha enriquecido la información disponible, ya se puede enviar a métodos de generación de visualizaciones. Se hace uso de la librería Matplotlib para crear diversas visualizaciones típicas, útiles para el conocimiento de las tendencias de los usuarios, la personalización del servicio de la plataforma y su optimización:

1. Se recorrerán los datos de entrada formateándolos a la estructura de ejes necesaria para cada una de las visualizaciones. Generalmente se compondrá de dos listas o *arrays*, uno con los datos del eje-X y otro con los referentes al eje-Y. En ciertos casos, como en el mapa de calor, necesitaremos una matriz de datos para la correcta representación de la información.
2. Ya con los datos formateados, se creará la visualización y se le añadirán los datos que se formatearon en el punto primero. Este es el paso en el que se explicitará qué tipo de visualización se insertará para cada caso. Junto a la primera parte, es lo que más cambiará entre métodos.
3. Se ajustará el diseño gráfico de la visualización para que encaje estéticamente tanto con las otras visualizaciones generadas para la funcionalidad de analíticas, como con el diseño ya existente en la aplicación web.
4. Finalmente, ya generada la visualización, se guardará la figura en un objeto `BytesIO`. Este objeto de bytes se codificará a formato `*.png` y se devolverá por la salida del método. Esta parte de los métodos de creación de visualizaciones es muy lenta, ya que el proceso de renderizado de una imagen con la calidad suficiente para ser mostrada en el servidor no es un proceso instantáneo. Será uno de los motivos por los que se haga un futuro cambio de tecnologías.

Con estos objetos de imagen ya guardados el servidor crea una vista de Django en la que se muestran los resultados de analíticas creadas en Matplotlib y se la envía al navegador web del cliente como una página HTML. En este proceso se utilizan las plantillas HTML enriquecidas que ofrece Django. A continuación, se muestra una sección de una de estas plantillas creadas para la herramienta de analíticas en la que se puede ver cómo se han insertado dos visualizaciones:

```
...
<div class="main">
    <h2>INICIOS POR DIA DE LA SEMANA</h2>
    <hr/>

    <div class=' left' >
        <h4>Sesiones</h4><br>
        
    </div>

    <div class=' right' >
        <h4>Simulaciones</h4><br>
        
    </div>
</div>
...
```

4.4. Validación experimental

En esta sección se muestra el resultado final del primer prototipo de analíticas desarrollado con unos datos de prueba, no productivos. El prototipo muestra información de interés acerca de la actividad en la aplicación web.

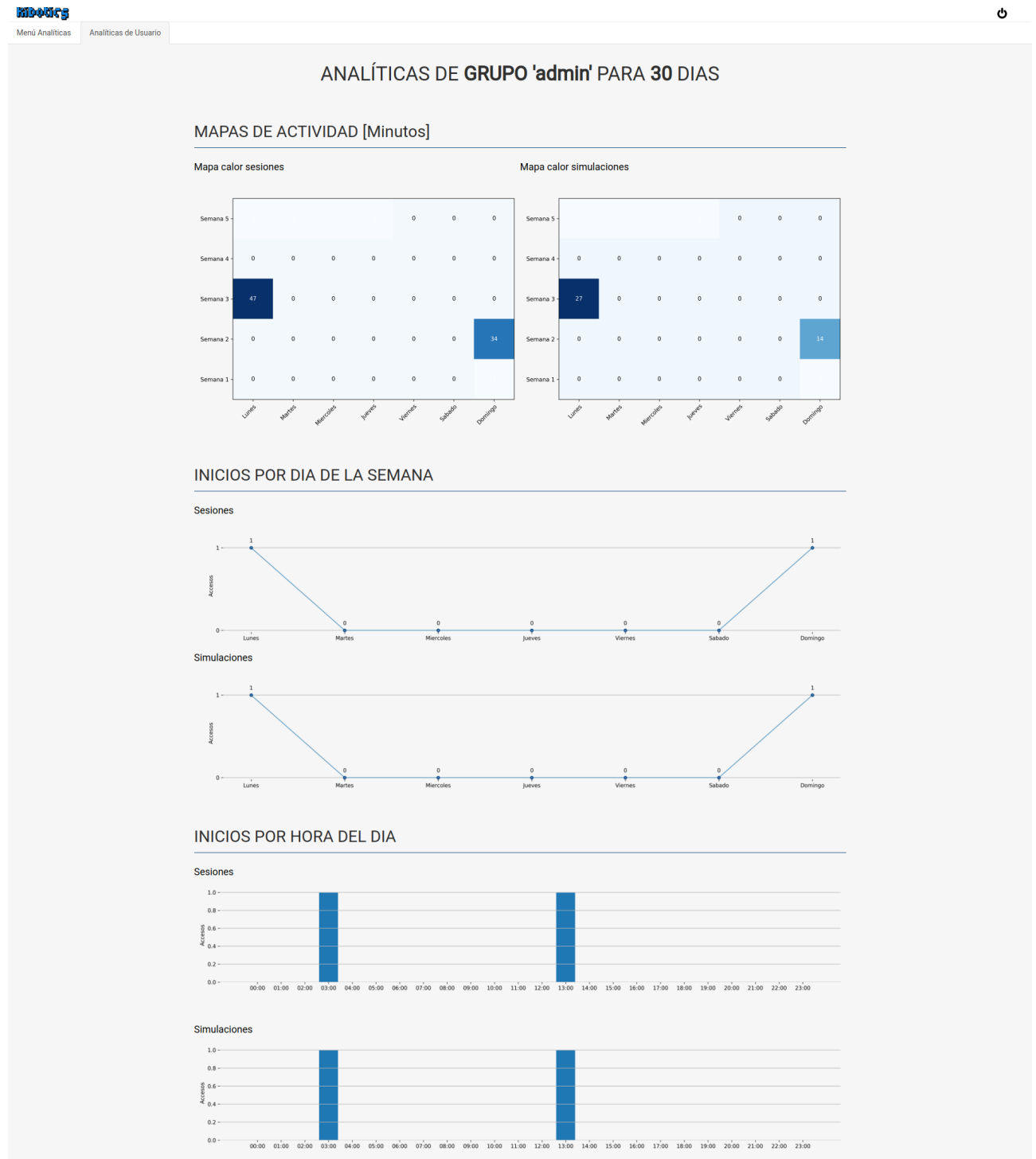


Figura 4.4: Analíticas del primer prototipo parte 1.

En la Figura 4.4, se puede observar una primera parte con dos mapas de calor, los cuales representan la actividad semana a semana (filas) en el servidor web de Kibotics para los distintos eventos de sesión y simulación. El color será más intenso cuantos más sean los minutos

invertidos en ese evento para cada uno de los días o celdas.

En una segunda parte de esta misma figura se representan 4 visualizaciones más con los accesos a sesiones y simulaciones separadas en dos grupos. Una primera agrupación con los accesos por día de la semana, a continuación, el segundo grupo con accesos divididos por la hora del día a la que fueron realizadas.

En la Figura 4.5 se representan las dos últimas secciones de este primer prototipo. Una primera sección con los tiempos totales y medios que el grupo de usuarios o usuario ha pasado en cada uno de los ejercicios a los que ha accedido. Finalmente, una última visualización que representa con un mapa geográfico la localización desde la que cada usuario ha accedido a la plataforma educativa.

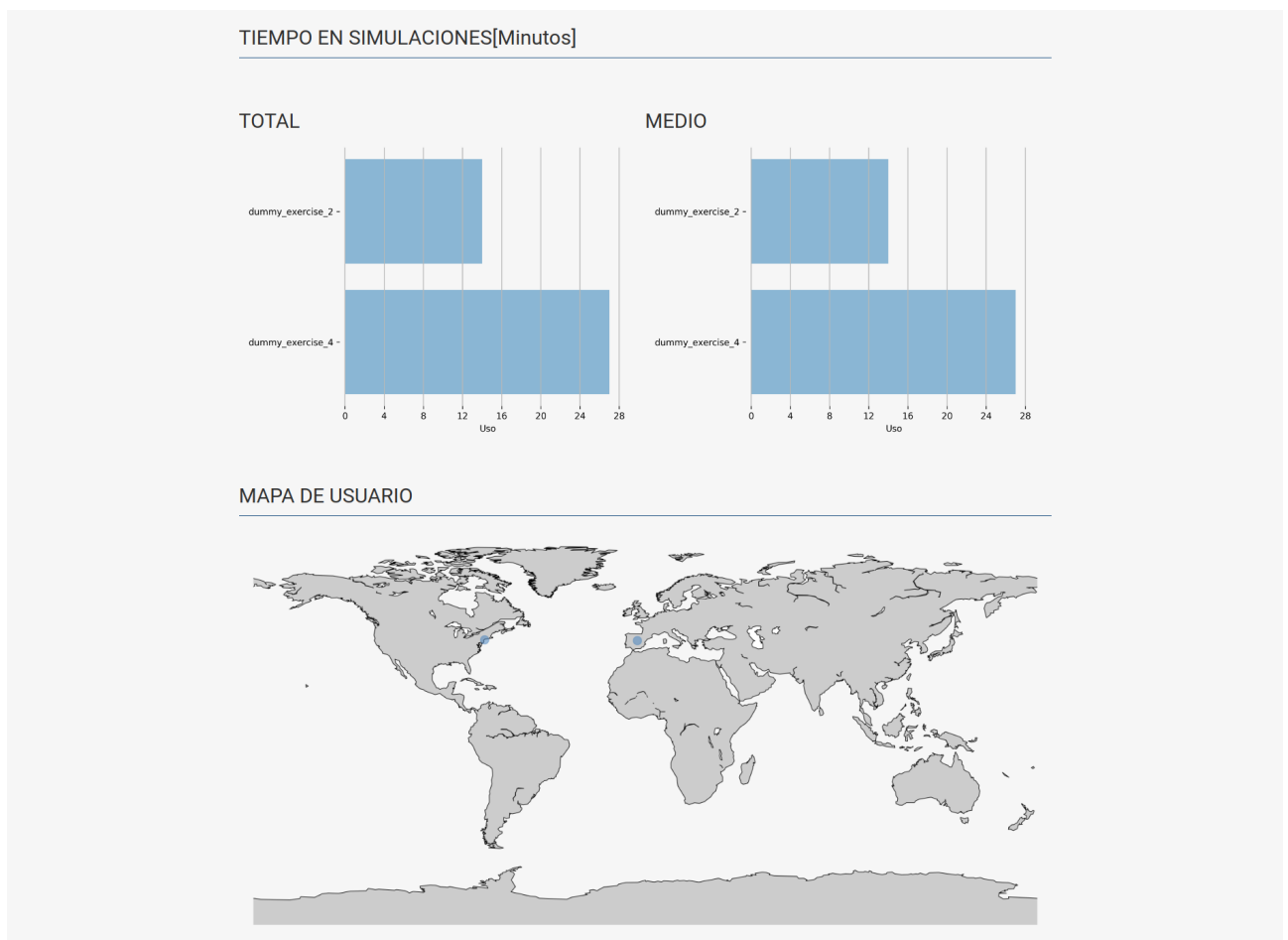


Figura 4.5: Analíticas del primer prototipo parte 2.

Este primer prototipo desarrollado de analíticas automáticas dentro de la plataforma Kibotics es bastante completo pero también tiene ciertos inconvenientes. Primero, falta cierta información útil que podría ser representada, como desde qué dispositivos acceden los usuarios o la actividad de los usuarios no registrados que simplemente visitan la web de la plataforma pero no llegan a registrarse o usarla.

Las visualizaciones generadas, al estar insertadas como imágenes, carecen de interactividad, la cual sería muy útil para tener información extra o poder realizar filtrados más finos de los datos. Además, al tener que renderizar cada una de estas imágenes individualmente, tarda unos segundos en mostrar las visualizaciones.

Capítulo 5

Analíticas basadas en Elasticsearch y Kibana

En este capítulo se describen las nuevas tecnologías utilizadas en la versión final del módulo de analíticas que sustituyen a MongoDB y Matplotlib. Para enriquecer los datos que se estaban almacenando por las sondas del servidor, a esta nueva versión de la herramienta, se han añadido nuevas sondas y enriquecido las existentes. También se detallan la creación de recursos de prueba para que los futuros desarrolladores puedan integrar estas tecnologías en local.

5.1. Diseño

Para el desarrollo de las mejoras necesarias de la herramienta de analíticas, el primer paso es realizar un diseño con los cambios a implementar así como las nuevas tecnologías involucradas.

El primer prototipo estaba basado en MongoDB como base de datos de logs, y con Matplotlib como generador de visualizaciones. Pese a ser bastante completo, carece de cierta información útil como datos de visitantes a la aplicación o acerca del *software* y *hardware* utilizado por los usuarios. Además, debido a la necesidad de renderizar cada una de las imágenes en Python, el subsistema de analíticas era lento.

Es por esto, que se decide cambiar la tecnología de visualización de datos a Kibana. Al ser Kibana parte del stack ELK, la base de datos cambia a Elasticsearch para poder utilizar todo el

potencial que este ecosistema ofrece.

Para enriquecer los datos almacenados y evitar tener que cruzar datos como se detalla en la subsección 4.3.2, se van a añadir más sondas de almacenamiento de datos, esta vez con datos relativos a la actividad de los visitantes de la plataforma educativa. Además, se van a unificar las tablas en las que se almacenarán registros similares, evitando tener así que cruzar datos. Estos cambios reducirán el coste computacional de la herramienta haciéndola así más rápida.

El nuevo diseño y los cambios de tecnología involucrados en este prototipo final se pueden ver representados de forma esquemática en la Figura 5.1.

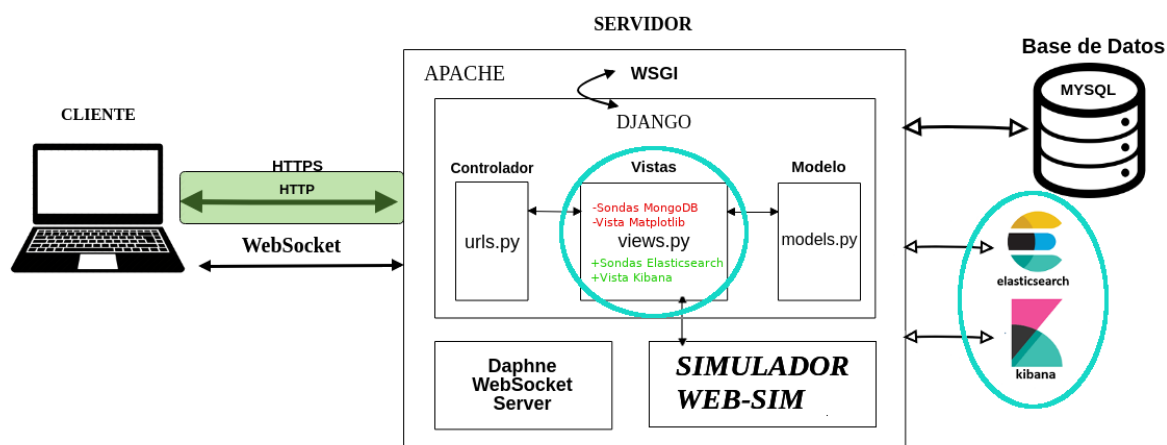


Figura 5.1: Arquitectura Kibotics segundo prototipo.

5.2. Implementación

En esta sección se detallan los cambios involucrados en el desarrollo de la herramienta de analíticas para la integración de Elasticsearch como base de datos y Kibana como herramienta de visualización. Además, de esta versión final de la herramienta, se explica la generación de recursos de prueba para futuros desarrolladores de Kibotics.

5.2.1. Elasticsearch en Webserver

El primer paso en el proceso de mejora de la herramienta es el cambio de base de datos. Para hacer uso del stack ELK, es necesario sustituir MongoDB por Elasticsearch. Para esto lo primero será instalar e iniciar un servicio de Elasticsearch en local ¹.

Una vez iniciado el servicio Elasticsearch, ya dispondremos de la base de datos sobre la que guardar los registros de log o documentos, como se llaman en Elasticsearch. Similar a las tablas de las que hacía uso MongoDB, estos documentos se almacenan en índices, cada uno de los cuales está definido por un esquema de campos y tipologías que definen la estructura de los documentos que almacenan.

Se podrán crear tantos índices como sea necesario, para consultarlos se puede lanzar una sentencia por terminal, o acceder con la IP y el puerto desde un navegador a la API REST configurada en instalación. Un ejemplo de llamada para un índice llamado `index_name_test` es la siguiente:

```
http://127.0.0.1:9200/index_name_test/_search/?size=1000&pretty
```

El siguiente paso es la integración de Elasticsearch en Django. Se realizará haciendo uso de la librería Python `django_elasticsearch_dsl`. Para migrar de MongoDB a Elasticsearch se han realizado dos pasos: creación de los índices y migración de las sondas que almacenan los logs de MongoDB a Elasticsearch.

El primer paso consistió en crear los índices de Elasticsearch. Estos índices son muy similares a los modelos que se utilizan en Django para representar la información de la base de datos. Para esta versión de la herramienta se crean los siguientes índices:

- `kibotics_session_log`: índice en el que se almacenan los eventos referentes a las sesiones.
- `kibotics_simulation_log`: índice en el que se almacena la información relativa a las simulaciones.

¹<https://www.elastic.co/guide/en/elasticsearch/reference/current/targz.html>

- `kibotics_error_log`: índice en el que se almacenan los eventos referentes a los errores.
- `kibotics_visit_log`: índice en el que se almacena la información relativa a las visitas.

Para evitar la problemática que surgió durante el desarrollo del primer prototipo, relativo al cálculo de la duración de los eventos, se ha eliminado el campo que almacenaba la fecha. Para sustituirlo, se han añadido dos nuevos campos a los índices de Elasticsearch los cuales establecerán el inicio y fin de cada evento, unificando así los dos registros de log que se tenían en la primera versión.

Por otro lado, el campo `USER_AGENT`, que se almacenaba anteriormente y ofrecía información acerca del dispositivo y software que utilizaban los visitantes de la web, ha sido dividido y sustituido con la información del navegador, dispositivo y sistema operativo. Cada uno con su propio campo en los esquemas de los índices de Elasticsearch. Ofreciendo así la información de forma más clara y eliminando ciertos datos no útiles para las visualizaciones que se quieren generar.

Para trabajar en Kibana con mapas geográficos es necesario guardar tanto la longitud como la latitud, para lo cual se hará uso de una tipología de campo ya existente en Elasticsearch llamado *Geo Point* que almacenará en un diccionario ambas variables.

Para complementar las nuevas sondas, que se han mencionado anteriormente y se explicarán con más detalle a continuación, es necesario la creación de un nuevo índice de visitas, en el que se registrarán los accesos a la página principal de la aplicación, estén o no registrados.

Un ejemplo de la definición en Django del índice de sesiones de Kibotcis Webserver en el fichero `documents.py` es:

```
from django_elasticsearch_dsl import Document, Text, Date, Double, GeoPoint, Ip

class SessionDocument(Document):
```

```
username = Text()
start_date = Date()
end_date = Date()
duration = Double()
client_ip = Ip()
browser = Text()
os = Text()
device = Text()
location = GeoPoint()

class Index:
    name = 'kibotics_session_log'
    settings = {
        'number_of_shards': 1,
        'number_of_replicas': 0
    }
```

Con todo esto, se tiene una base sólida sobre la que almacenar la información de logs, solo falta modificar el guardado en Elasticsearch, migrar las sondas que almacenan los registros de log.

Para enriquecer las sondas ya existentes y hacer uso de los nuevos campos e índices creados, se han añadido nuevas sondas para almacenar eventos de visitantes, así como sondas para optimizar la monitorización de salida de sesiones y simulaciones y evitar no tener un registro de inicio sin su correspondiente registro de fin de evento por un cierre busco de la aplicación o inactividad.

Las sondas de inicio de los eventos son similares a las que se tenían anteriormente en MongoDB, un ejemplo para la sonda de inicio de simulación es:

```
SimulationDocument(
    username = "USERNAME_TEST",
    start_date = start_date_object_test,
    end_date = start_date_object_test,
    duration = 0.0,
    client_ip = "CLIENT_IP_TEST",
```

```

simulation_type = "SIMULATION_TYPE_TEST",
exercise_id = "EXERCISE_ID_TEST",
browser = "BROWSER_TEST",
os = "OS_TEST",
device = "DEVICE_TEST",
location = {'lat': latitude_double_test, 'lon': longitude_double_test}
).save()

```

Sin embargo, al unificar los registros de entrada y salida en un único documento las sondas de fin de sesión/simulación cambian. Tendrán que buscar en Elasticsearch el último documento no finalizado en el índice para el usuario del cual se quiera cerrar el evento y sustituir tanto los campos `end_date` como `duration`.

Esto se realiza gracias al identificador que cada documento indexado posee al cual se le realiza una operación *update* con los nuevos campos. Un ejemplo de cierre de sesión en Python es:

```

# Búsqueda del ultimo registro de sesión no cerrado
latest_session = Search(index="kibotics_session_log*") \
    .query("match", username=username) \
    .query('match', duration=0) \
    .sort({"start_date": {'order': 'desc'}})[0]

# Actualización de los campos end_date y duration
for hit in latest_session:
    duration = datetime.now() - datetime.strptime( \
        hit.start_date, \
        "%Y-%m-%dT%H:%M:%S.%f")

    Elasticsearch(settings.ELASTICSEARCH_DSL['default']['hosts']) \
        .update(index = 'kibotics_session_log',
                id = hit.meta.id,
                body = {"doc": {
                    'end_date' : datetime.now(),
                    'duration' : duration.total_seconds()
                }}
        )

```

Durante el desarrollo de esta lógica es útil hacer uso de la interfaz de Elasticsearch para comprobar que todos estos campos están siendo creados y actualizados correctamente. A esta interfaz se accede mediante la URL y puerto configurados durante el proceso de instalación.

En esta API, podremos realizar filtrados por campos e índices haciendo uso de expresiones regulares *Regex*, por ejemplo para acceder al índice en el que se almacenan los documentos relativos a los logs de sesiones, podremos acceder mediante la siguiente URL:

```
http://127.0.0.1:9200/kibotics_session_log/_search/?size=1000&pretty
```

En la Figura 5.2 se puede observar la respuesta que se obtiene de la API de Elasticsearch en el navegador a la llamada anterior.

```
▼ 1:
  _index:      "kibotics_session_log"
  _type:       "_doc"
  _id:         "ccCT63EBdVY2vb3Ic8f8"
  _score:      1
  ▼ _source:
    username:   "student_dummy"
    start_date: "2020-01-29T06:00:00"
    end_date:   "2020-01-29T06:31:00"
    duration:   1860
    client_ip:  "161.185.160.93"
    browser:    "dummy_browser_5"
    os:         "dummy_os_5"
    device:     "dummy_device_5"
    ▼ location:
      lat:      40.7654
      lon:      -73.8174
▼ 2:
  _index:      "kibotics_session_log"
  _type:       "_doc"
  _id:         "a8CT63EBdVY2vb3Ic8f0"
  _score:      1
  ▼ _source:
    username:   "teacher_dummy"
    start_date: "2020-01-28T05:00:00"
    end_date:   "2020-01-28T05:32:00"
    duration:   1920
    client_ip:  "195.235.232.206"
    browser:    "dummy_browser_4"
    os:         "dummy_os_4"
    device:     "dummy_device_4"
    ▼ location:
      lat:      40.4
      lon:      -3.6833
```

Figura 5.2: API Rest Elasticsearch.

5.2.2. Kibana en Webserver

Para acceder a Kibana, deberemos primero instalar y ejecutar el servicio². Una vez Kibana está ejecutando podremos acceder a su interfaz gráfica mediante la URL y puerto configurado en la instalación:

`http://127.0.0.1:5601/app/kibana#/home`

Ya con datos en Elasticsearch almacenados y como se muestra en la Figura 5.3, Kibana nos pedirá que añadamos los índices sobre los que queremos obtener soporte y seleccionemos el campo sobre el que se filtrarán temporalmente estos logs. En este caso se utilizará el campo `start_date`.

Step 1 of 2: Define index pattern

Index pattern

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, |.

> Next step

✓ **Success!** Your index pattern matches **6 indices**.

kibotics_classification_log
kibotics_error_log
kibotics_ranking_log
kibotics_session_log
kibotics_simulation_log
kibotics_visit_log

Figura 5.3: Creación de índices en Kibana.

Configurados todos los índices en Kibana se tiene una primera visualización de los documentos indexados en Elasticsearch en la pestaña *Discover* de Kibana. En esta pestaña podremos ver todos los logs que estén almacenados en los índices, así como un histograma en el que se podrá ver gráficamente la evolución de estos índices en el tiempo. Además, esta sección *Discover* proporciona la posibilidad de filtrar por rangos de fechas y campos así como se tenía en el primer prototipo de este proyecto.

²<https://www.elastic.co/guide/en/kibana/current/targz.html>

Esta sección ya nos da una primera pincelada de la potencia de procesamiento de Kibana, así como la sencillez de implementación y despliegue. Las consultas son instantáneas, notablemente más rápidas que las realizadas en el primer prototipo desarrollado.

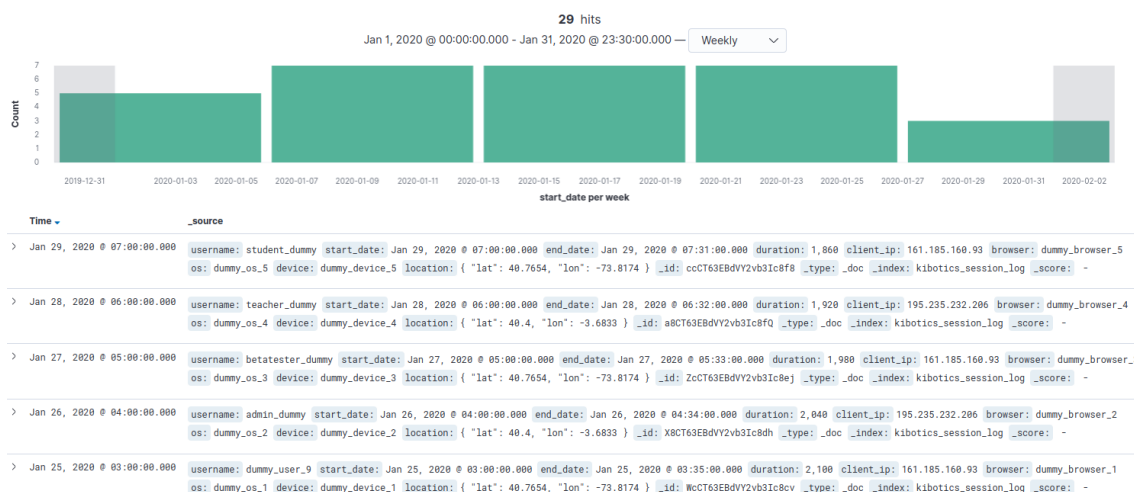


Figura 5.4: Sección *Discover* en Kibana.

Kibana ofrece la posibilidad de creación de *scripted fields*, campos cuyo valor deriva de otros campos o datos ya indexados. Generados en un lenguaje muy similar a C llamado *painless*.

Para el módulo de analíticas que se quiere desarrollar, y en especial para las visualizaciones que filtran por día de la semana y por hora del día, se han creado dos *scripted fields* para cada uno de los índices utilizados. Estos son *day_of_week* y *hour_of_day*, el código *painless* utilizado para la creación de estos campos es el siguiente:

```
// day_of_week
["", "1 Lunes", "2 Martes", "3 Miercoles", "4 Jueves", "5 Viernes",
"6 Sabado", "7 Domingo"] [doc['start_date'].value.dayOfWeek]

// hour_of_day
doc['start_date'].value.hourOfDay
```

Con estos *scripted fields* además de los campos ya almacenados en los documentos, Kibana dispone de todos los datos necesarios para la creación de las visualizaciones. Para ello, en la sección *Visualize*, Kibana tiene una colección de distintas plantillas gráficas. Estas plantillas

serán las que se configuren con los índices, documentos y campos a usar para la generación de los distintos tipos de visualizaciones. En la Figura 5.5 se muestran algunas de las visualizaciones que Kibana ofrece.

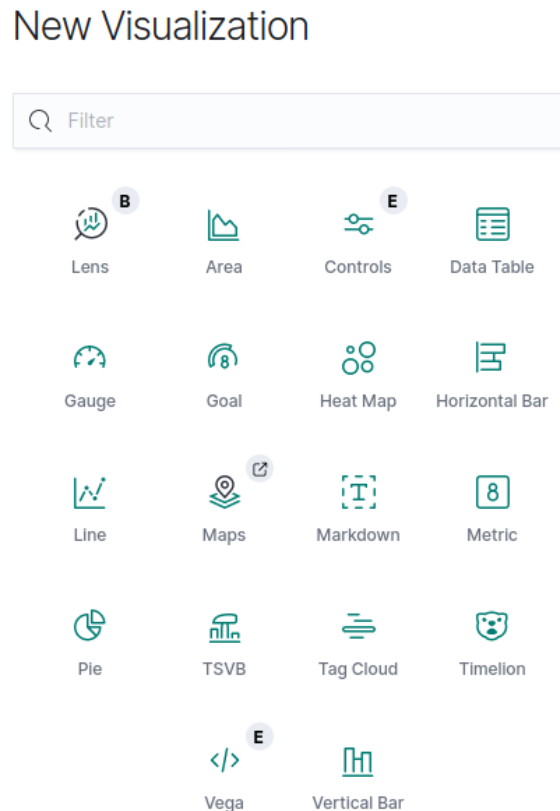


Figura 5.5: Menú creación de visualizaciones.

Ya creadas las visualizaciones, solo quedará integrarlas en Kibotics sustituyendo las generadas en Matplotlib. Para ello se creará una vista `menú` con la que se seleccionará a que tipo de analíticas se quiere acceder. Este menú selector de analíticas es el mostrado a continuación en la Figura 5.6.

SELECTOR DE FECHAS

20200512/20200610 ▼

ANALÍTICAS DE USUARIOS

NOMBRE DE USUARIO GRUPO TODOS LOS USUARIOS

admin_dummy Buscar Admin Buscar Buscar

ANALÍTICAS VISITANTES

TODOS LOS VISITANTES

Buscar

Figura 5.6: Menú de selección de analíticas en Kibotics

En esta vista se filtrará tanto por usuarios y grupos, como por fechas de las cuales se quieren analíticas. Una vez filtrado, Django generará automáticamente una URL con los datos seleccionados en el Menú de Kibotics. Esta URL dinámica apuntará a las visualizaciones de nuestro servicio de Kibana y será devuelta por el contexto de Django hasta las plantillas Django que lo insertarán en un elemento HTML `iFrame`.

5.3. Validación experimental

Para esta versión final del módulo de analíticas, se han creado una amplia selección de visualizaciones de las que obtener información de actividad de los usuarios. Divididas en dos *Dashboards* o tableros, el primero reservado a los visitantes no registrados y el segundo para analíticas de sesiones y simulaciones de usuarios registrados. A continuación, se mostrarán las visualizaciones creadas así como una explicación de lo que representan.

Para tener un control del número de accesos a lo largo del tiempo se creó el histograma representado en la Figura 5.7. Este histograma muestra los eventos almacenados en el índice `kibotics_session_log` dividido día a día en el eje de abscisas.

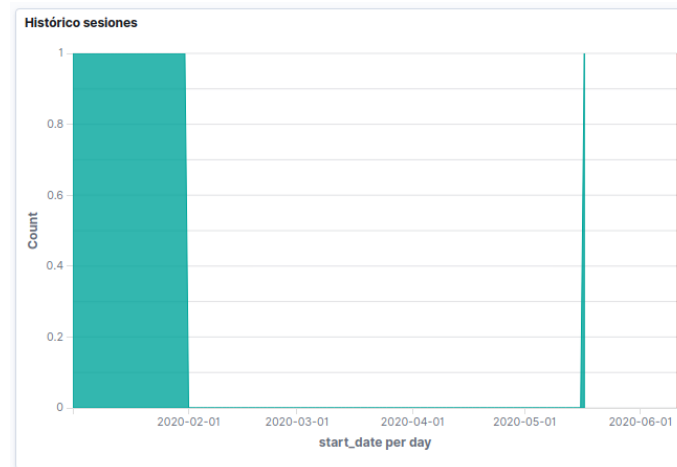


Figura 5.7: Histograma

Se ha creado un mapa de calor, que al igual que el histograma antes mencionado, representa la actividad de inicio de eventos. Dividido en columnas, cada una representa una semana con sus 7 días correspondientes. Un color más oscuro en la celda indica mayor actividad para ese día. Como se puede observar en la Figura 5.8, hay una visualización para las sesiones y otro para las simulaciones, división recurrente que se observa en distintas secciones del módulo Kibana desarrollado.

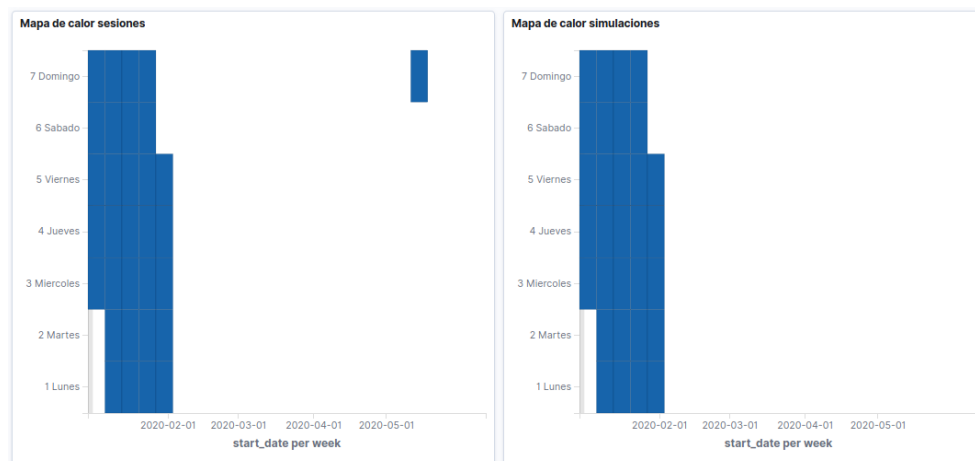


Figura 5.8: Mapa de calor sesiones y simulaciones

La Figura 5.9, dividida en dos visualizaciones, representa la actividad tanto de sesiones como de simulaciones almacenada en los índices de Elasticsearch `kibotics_session_log` y `kibotics_simulation_log` respectivamente. En formato gráfico de barras vertical, los datos, filtrados por el día de la semana en que se registraron en el campo inicio de evento

start_date.

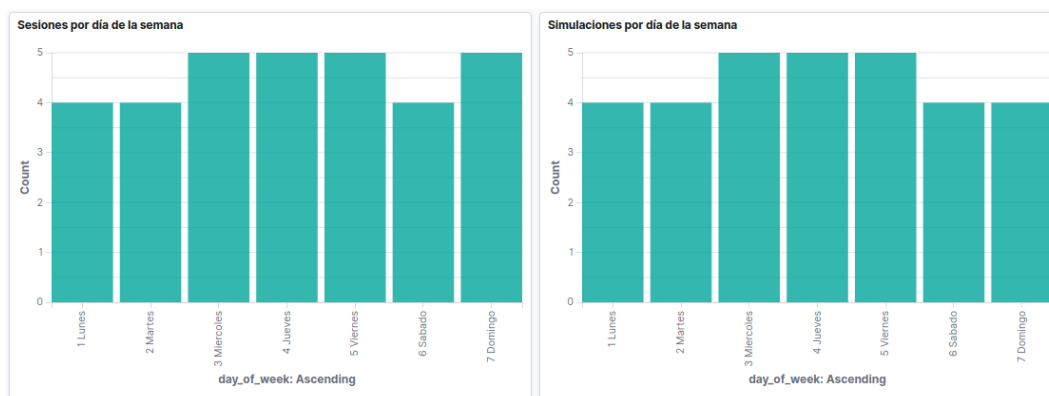


Figura 5.9: Gráfico de barras por día de la semana

La Figura 5.10 muestra, en dos visualizaciones con gráficas de barras verticales, la actividad en el servicio web Kibotics. Filtrados, esta vez, por la hora del día en que se registraron los eventos. Como se puede observar, cada una de estas visualizaciones corresponde a uno de los eventos que han sido logueados en Elasticsearch y almacenados en los índices `kibotics_session_log` y `kibotics_simulation_log`.

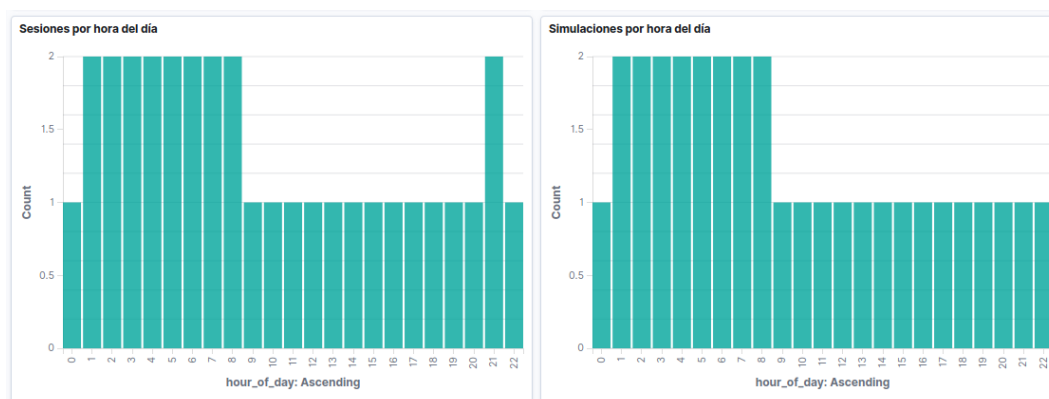


Figura 5.10: Gráfico de barras por hora del día

Para mostrar la actividad en las simulaciones que Kibotics ofrece, se han desarrollado las visualizaciones representadas en la Figura 5.11. Dividido en dos gráficos de barras que representan el tiempo invertido por los usuarios. Esta información está almacenada en el campo `duration` del índice `kibotics_simulation_log`. La primera visualización muestra el tiempo total invertido y la última representa el tiempo medio invertido.

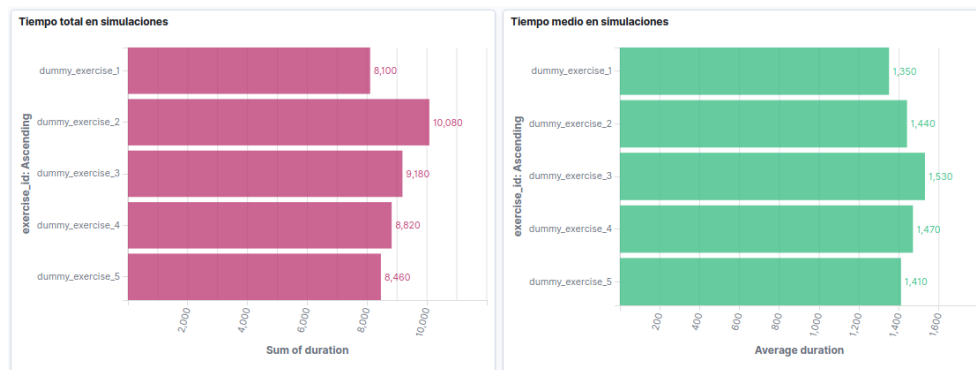


Figura 5.11: Gráfico de barras para tiempo total y medio en simulaciones

En la Figura 5.12, se representa gráficamente la superficie terrestre, en ella se muestran eventos de inicio de sesión ocurridos para el rango de fechas seleccionado. Hace uso de los datos de latitud y longitud almacenados en formato `Geo Point` del índice `kibotics_session_log`.

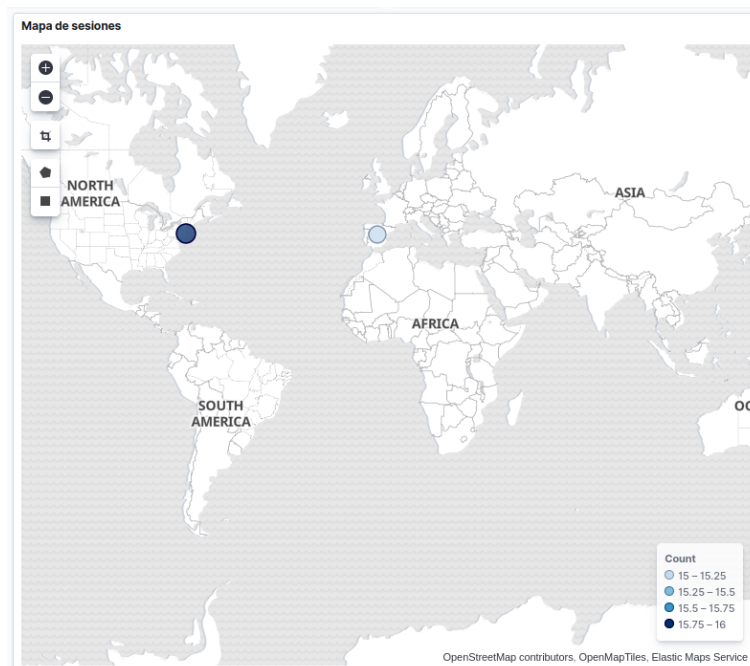


Figura 5.12: Mapa geográfico de sesiones

Para monitorizar el *hardware* y *software* utilizado por los usuarios de la web, se han desarrollado las tres gráficas circulares mostradas a continuación en la figura 5.13. Cada una de ellas representa porcentualmente campos almacenados en el índice de sesiones `kibotics_session_log` que proporcionan información acerca del sistema operativo, dispositivo y navegador utilizados.

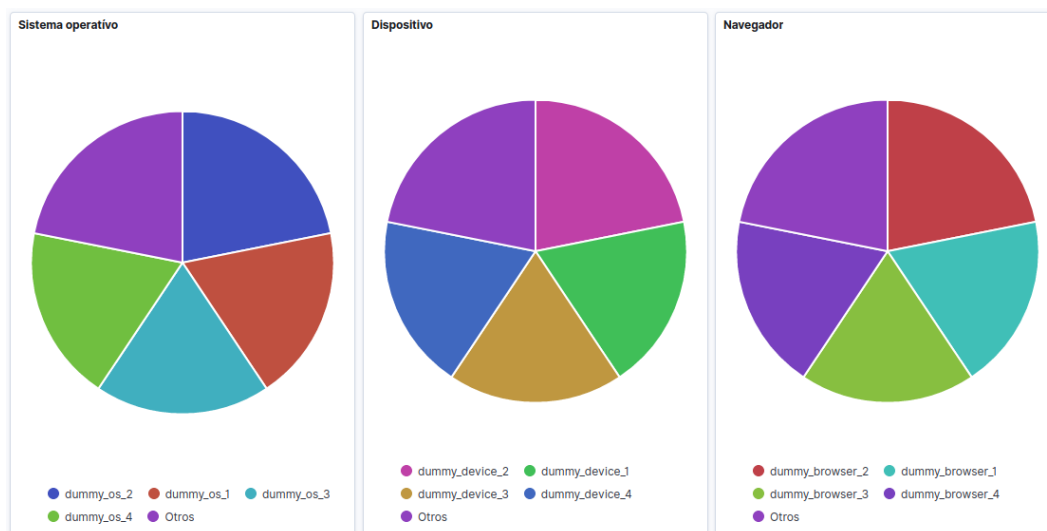


Figura 5.13: Gráficas circulares para SO, Dispositivo y Navegador

En la figura 5.14, se muestra una última sección con dos tablas de datos con los últimos eventos de sesión y simulación almacenados en los índices `kibotics_session_log` y `kibotics_simulation_log` de Elasticsearch para cada uno de los usuarios filtrados.

username: Descending	start_date: Descending	Count
admin_dummy	May 17, 2020 @ 23:21:37.702	1
betatester_dummy	Jan 27, 2020 @ 05:00:00.000	1
dummy_user_1	Jan 30, 2020 @ 08:00:00.000	1
dummy_user_2	Jan 31, 2020 @ 09:00:00.000	1
student_dummy	Jan 29, 2020 @ 07:00:00.000	1
teacher_dummy	Jan 28, 2020 @ 06:00:00.000	1
dummy_user_3	Jan 19, 2020 @ 20:00:00.000	1
dummy_user_4	Jan 20, 2020 @ 21:00:00.000	1
dummy_user_5	Jan 21, 2020 @ 22:00:00.000	1

username: Descending	start_date: Descending	Count
betatester_dummy	Jan 27, 2020 @ 05:05:00.000	1
dummy_user_1	Jan 30, 2020 @ 08:05:00.000	1
dummy_user_2	Jan 31, 2020 @ 09:05:00.000	1
student_dummy	Jan 29, 2020 @ 07:05:00.000	1
teacher_dummy	Jan 28, 2020 @ 06:05:00.000	1
admin_dummy	Jan 26, 2020 @ 04:05:00.000	1
dummy_user_3	Jan 19, 2020 @ 20:05:00.000	1
dummy_user_4	Jan 20, 2020 @ 21:05:00.000	1
dummy_user_5	Jan 21, 2020 @ 22:05:00.000	1

Figura 5.14: Últimos eventos logueados

Todas estas visualizaciones son interactivas y se puede filtrar por sus campos simplemente pulsando sobre ellas. Funcionalidad muy útil que no poseían las imágenes renderizadas que se generaban en el primer prototipo. Además, cada una de las visualizaciones del módulo desarrollado tiene una opción para ver los datos representados en texto plano e incluso descargarlos en formato CSV para su posterior tratamiento.

Se han mostrado las visualizaciones referentes a la sección de sesiones y simulaciones de usuarios registrados ya que es la que más información proporciona, pero hay otra sección de analíticas de visitantes con unas visualizaciones de monitorización similares a las mostradas anteriormente.

A continuación, en las siguientes figuras, se muestra Kibana ya integrado en la nueva vista de analíticas del servicio web Kibotics haciendo uso del `iFrame`.

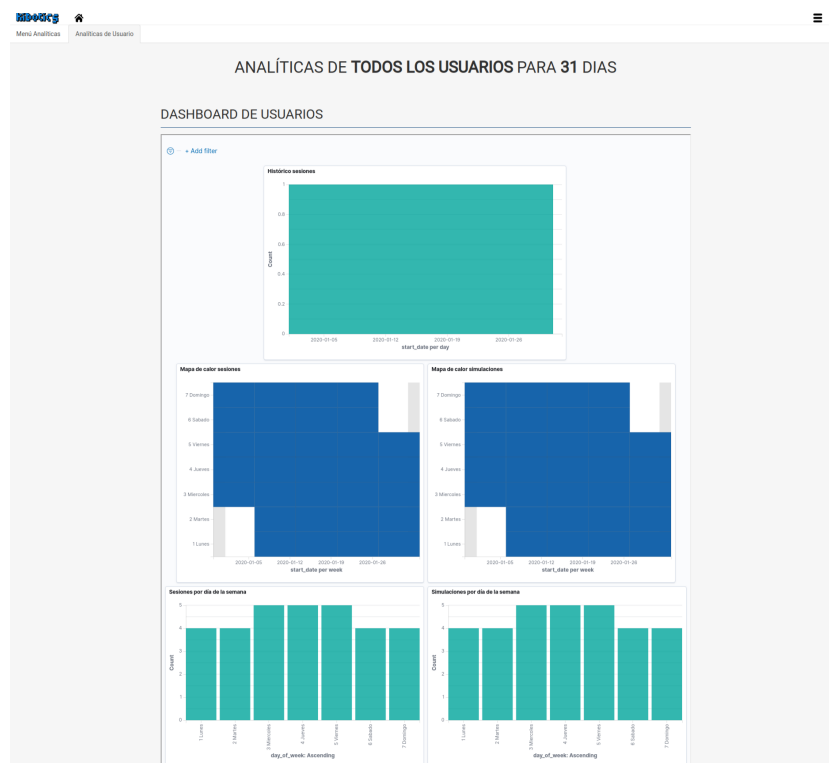


Figura 5.15: Kibana en Kibotics parte 1

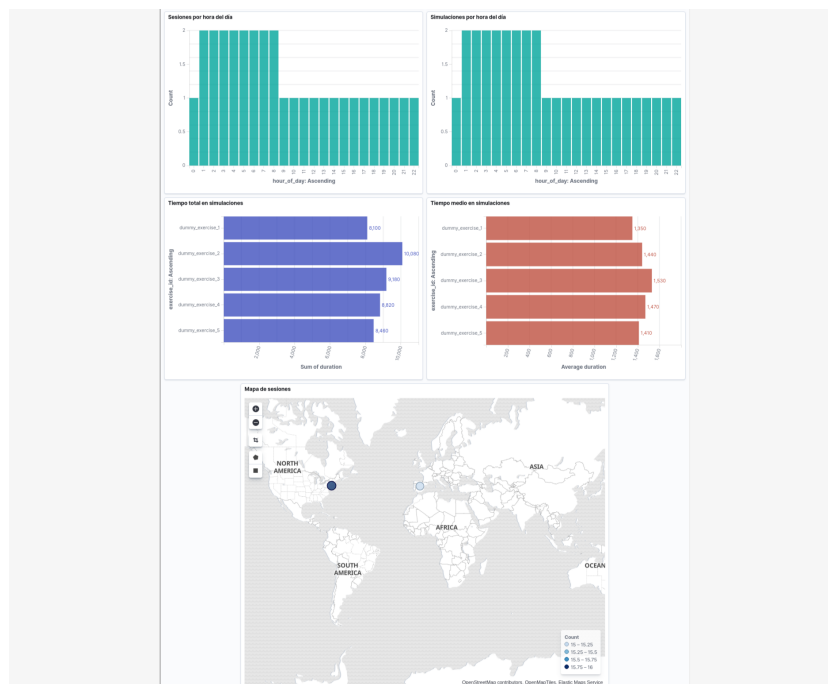


Figura 5.16: Kibana en Kibotics parte 2

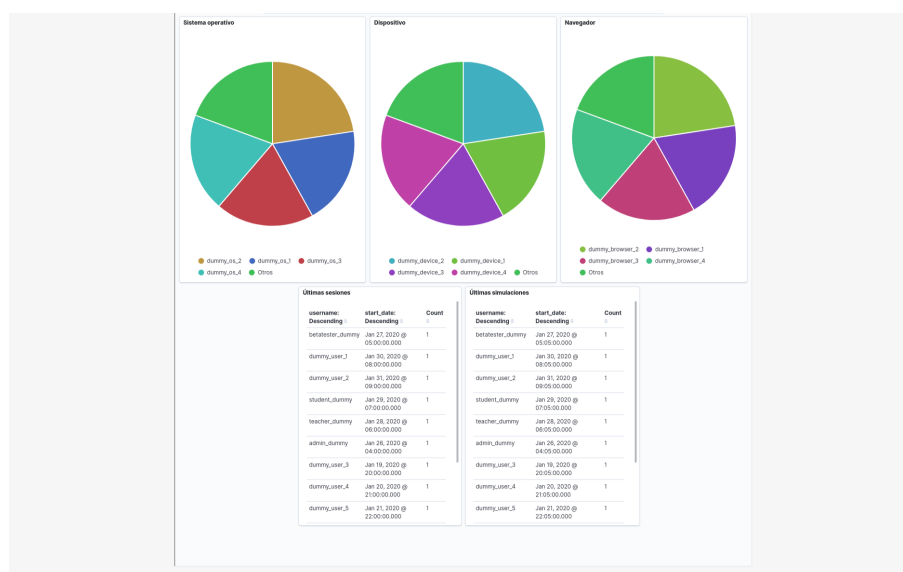


Figura 5.17: Kibana en Kibotics parte 3

Para proporcionar a los futuros desarrolladores los recursos necesarios con los que poder empezar a trabajar sin tener que crearlos de manera manual, se ha creado una base de datos dummy la cual ofrece una variedad de datos para todos los índices utilizados tanto en este proyecto, como en otros desarrollos paralelos.

La creación de esta base de datos de pruebas se generó mediante un *script* de Python que crea los índices y su estructura haciendo uso de la librería que Elasticsearch proporciona para Python.

El *script* creará todos los índices con sus respectivas estructuras y tipologías de datos. Un ejemplo para el índice de sesiones es:

```
# Import librería Elasticsearch y conexión
from elasticsearch import Elasticsearch
client = Elasticsearch()

# JSON con la estructura del índice
session_mapping = {
    "settings": {
        "number_of_shards": 1,
        "number_of_replicas": 0 },
    "mappings": {
        "properties": {
            "username": { "type":"keyword" },
            "start_date":{ "type":"date" },
            "end_date":{ "type":"date" },
            "duration": { "type":"double" },
            "client_ip":{ "type":"ip" },
            "browser": { "type":"keyword" },
            "device":{ "type":"keyword" },
            "location": { "type": "geo_point" },
            "os":{ "type":"keyword" }
        }
    }
}

# Creación del índice en Elasticsearch
client.indices.create( index="kibotics_session_log",
                      body=session_mapping,
                      ignore=400 )
```


Una vez creadas las estructuras de los índices, el siguiente paso es guardar todos los objetos con la información de prueba que se desee guardar.

Finalizada la ejecución del *script* ya tendríamos los datos en nuestro servicio local de Elasticsearch. Para simplificar más la instalación de la base de datos, se han generado una serie de documentos JSON con la estructura y datos que otros usuarios importarán a su servicio Elasticsearch.

Estos documentos JSON se han generado haciendo uso de la herramienta `elasticdump`, la cual tiene una instalación muy sencilla:

```
$ sudo npm install elasticdump -g
```

Para la generación de los documentos de datos y mapeo se han ejecutado las siguientes sentencias para cada uno de los índices usados en Elasticsearch:

```
$ elasticdump --input=http://127.0.0.1:9200/"INDEX_NAME"  
--output="./mapping_elasticsearch.json" --type=mapping
```

```
$ elasticdump --input=http://127.0.0.1:9200/"INDEX_NAME"  
--output="./data_elasticsearch.json" --type=data
```

Para la importación de estos ficheros en la base de datos, el desarrollador simplemente tiene que instalar `elasticdump` y ejecutar un *script bash* el cual recorre y almacena todos los ficheros al servicio Elasticsearch local:

```
indexes="session simulation visit error classification ranking"  
directory="./kibotics_dummy_es/"  
  
for index in $indexes; do  
    elasticdump --output=http://127.0.0.1:9200/"kibotics_"$index"_log"  
    --input=$directory"mapping_"$index"_es.json" --type=mapping  
  
    elasticdump --output=http://127.0.0.1:9200/"kibotics_"$index"_log"  
    --input=$directory"data_"$index"_es.json" --type=data  
done
```

El proceso de exportación e importación de datos de prueba para Kibana es sencillo pues la propia interfaz gráfica de Kibana nos proporciona una herramienta para realizarlo como muestra la Figura 5.18.

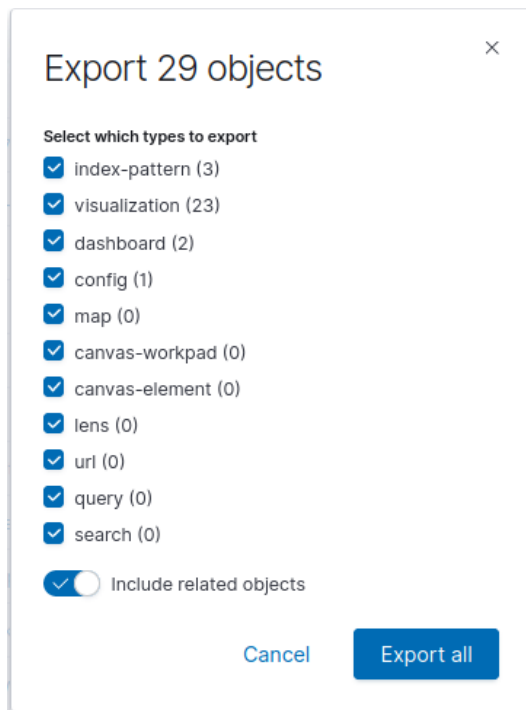


Figura 5.18: Exportación en interfaz Kibana.

Esta herramienta nos generará un fichero NDJSON similar a la estructura JSON con los patrones de índices creados, así como las visualizaciones, tablas o *scripted fields* guardados en Kibana.

Para que un futuro desarrollador importe estos datos simplemente podrá hacerlo por la interfaz gráfica de Kibana. Para unificar la metodología y ya que en el servidor de pre-producción/producción no se dispone de esta interfaz gráfica, esta también se puede realizar mediante la siguiente sentencia:

```
$ curl -X POST "localhost:5601/api/saved_objects/_import" -H "kbn-xsrf: true"
--form file=@kibotics_dummy_kibana.ndjson
```

Capítulo 6

Conclusiones

6.1. Conclusiones finales

6.2. Competencias adquiridas

6.3. Trabajos futuros

Bibliografía

- [1] *Estándar HTML*: <https://html.spec.whatwg.org/>
- [2] *Documentación JavaScript MDN web docs*: <https://developer.mozilla.org/es/docs/Web/JavaScript>
- [3] *Patrón Django Model-View-Controller*: <https://uniwebsidad.com/libros/django-1-0/capitulo-5/el-patron-de-diseno-mtv>
- [4] *Documentación Django Project*: <https://www.djangoproject.com/>
- [5] *Documentación MongoDB*: <https://docs.mongodb.com/guides/server/introduction/>
- [6] *Quién usa MongoDB*: <https://www.mongodb.com/who-uses-mongodb>
- [7] *Documentación Elasticsearch*: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>
- [8] *Histórico de versiones para Elasticsearch*: <https://www.elastic.co/downloads/past-releases#elasticsearch>
- [9] *Versiones de Matplotlib*: <https://github.com/matplotlib/matplotlib/releases>
- [10] *Documentación Kibana*: <https://www.elastic.co/guide/en/kibana/current/index.html>
- [11] *Histórico de versiones para Kibana*: <https://www.elastic.co/downloads/past-releases#kibana>

[12] *Documentación Matplotlib*: <https://matplotlib.org/>

[13] *Documentación Elasticdump*: <https://www.npmjs.com/package/elasticdump>