¡¡¡¡¡ Título temporal !!!!!

Ángel Perea Arias

24 de Mayo del 2020

Índice general

Índice de figuras

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

Capítulo 1

INTRODUCCIÓN

1.1. Resumen

1.2. Motivación

El objetivo principal es dotar a la Web Kibotics.org de sondas de almacenamiento de datos y herramientas de visualización para el análisis de estos, ya sean de visitantes a la web, registros de usuarios o uso de los ejercicios.

Ofreciendo así a la web y sus administradores de capacidades para recoger, estudiar y valorar los datos aportados para tener una mejor visión global de que rumbo tomar, como está funcionando el servicio, como mejorarlo.

Estos datos son imprescindibles para cualquier decisión importante que se deba tomar para mejorar la satisfacción de los usuarios, aumentar la retención, mejorar los contenidos y su distribución etc... (INCOMPLETO)

Capítulo 2

OBJETIVOS

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

3.1.1. 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 facil 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 módulos que puede ser fácilmente extendida por módulos personalizados escritos en C/Python. Haciendo uso del instalador de paquetes PIP, es posible la instalación e integración de paquetería en proyectos de manera muy sencilla, así como el cambio de versiones de las mismas.

El proyecto comenzó a desarrollarse en Python 2.6 y ha terminado en la versión Python 3.6.9.

3.1.2. Django

Django es un framework Web de alto nivel diseñado para desarrollar aplicaciones 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), donde:

- Model, esta capa del patrón tiene toda la información relativa a las bases de datos: como se almacenan, como se relacionan entre ellas, como 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.

- View, parte lógica del Framework, se puede ver como una unión entre la capa de modelo y de templates o 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 complejas, altamente enriquecidas y dinámicas en muy pocas lineas de código.

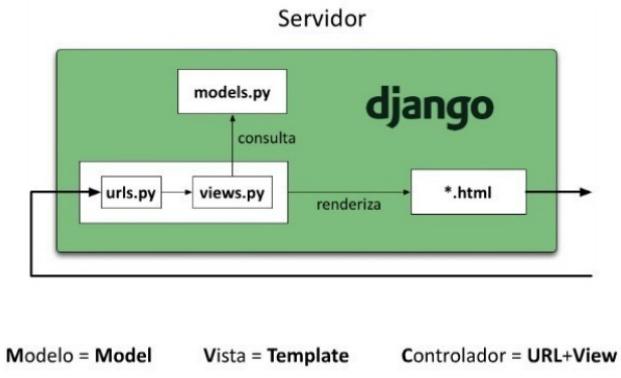


Figura 3.1: Patrón MVT Django.

El proyecto comenzó a desarrollarse en Django 1.9 y ha terminado en la versión Django 1.11.

3.1.3. HTML

HTML (Hipertextual Markup Lenguaje), es un lenguaje de marcado. Actualmente utilizado para la definición de estructura básica de los contenidos de una página Web como vídeos, figuras, iframes...

Publicado inicialmente en 1991, su historia se remonta a 1980, cuando Tim Berners-Lee propuso un nuevo sistema para compartir documentos. Actualmente se ha impuesto como el estándar, 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 en el campo de los recursos multimedia y de interactividad. Actualmente la última versión oficial es HTML 5, la cual proporciona soporte nativo de audio y vídeo, inclusión de la etiqueta canvas, entre otras mejoras.

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

3.1.4. CSS

CSS (Cascade Style Sheet), es un lenguaje de reglas en cascada utilizado para dotar de diseño a elementos. El cual 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, esto es, que se adapten al dispositivo en que están siendo vistas, ya sean, por ejemplo, tabletas, ordenadores o móviles.

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

3.1.5. JavaScript

JavaScript 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: 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), proporciona a JavaScript de más funcionalidades.

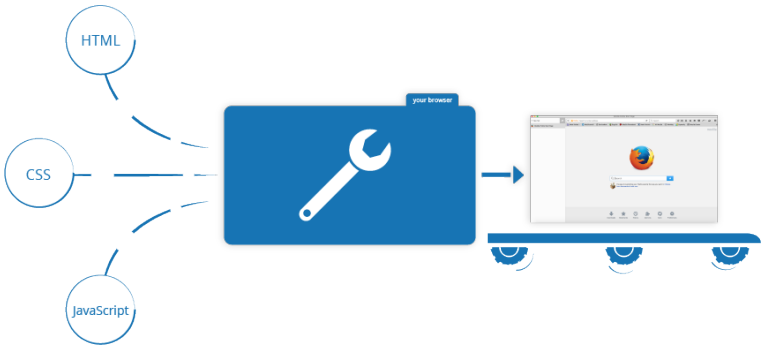


Figura 3.2: Tecnologías web.

3.2. Bases de datos

3.2.1. SQLite

SQLite es una librería ligera, rápida y fiable desarrollada en C. Siendo actualmente el motor de bases de datos SQL más usado en el mundo, utilizado en gran parte de los dispositivos móviles y ordenadores, además de venir de serie en muchas aplicaciones, por ejemplo, Django.

No necesita de de un servidor para funcionar, hecho por el cual su integración y despliegue es sencillo, basado en lectura y escritura en un fichero *.sqlite para almacenar toda la información de una base de datos, este fichero es multiplataforma pudiendo así ser migrado entre distintos sistemas de manera muy sencilla. Con un tamaño máximo de 140 terabytes.

3.2.2. MongoDB

MongoDB es una base de datos NoSQL distribuida, documental (almacenando la información en ficheros BSON, muy similares a JSON), de código abierto y diseñada para ofrecer un nivel productivo alto.

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 muy 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.

3.2.3. ElasticSearch

Elasticsearch es una base de datos. Junto a LogStash y kibana, los tres proyectos open source, forman el Stack ELK. Mediante una simple API Rest realiza consulta, borrado y actualización de documentos. Haciendo uso de objetos JSON tanto para las consultas como para las respuestas de estas, lo que la hace muy fácil de usar e integrar en sistemas productivos ya existentes.

Basada en Lucene(API para recuperación de información), gracias a esto, permite almacenar información como datos de geolocalización así como realizar búsquedas de texto y auto-completado es muy sencillo.

Organizado en nodos, permitirá aumentar la potencia a medida que la demanda de recursos crezca.

Datas sus capacidades de almacenar información preparada en índices previamente creados, la consulta de documentos es muy ágil puesto que evitamos búsquedas en indices no deseados, gracias a esto, se ha convertido en uno de los buscadores de texto más importantes, utilizado por gigantes de Internet como Facebook, Netflix o Github.

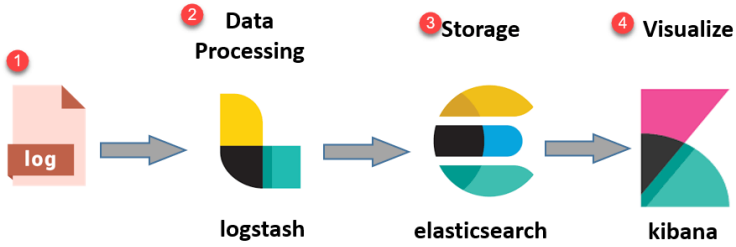


Figura 3.3: Stack ELK.

3.3. Tecnologías de visualización

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 gráficas 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 gráficos aún más complejos y enriquecidos como mapas geográficos en los que se representarán

datos mediante datos de latitud y longitud.

Matplotlib almacena las gráficas en Figuras, cada una contiene los ejes en los 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.

Estas visualizaciones o gráficos podrán ser mostrados 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 <img></img>

3.3.2. Kibana

Como se comentó anteriormente, el Stack ELK está compuesto por Kibana como motor de búsqueda, procesador de datos y generador de visualizaciones entre otras muchas funcionalidades.
Gracias a su aplicación frontend, la creación de gráficas 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, gráficos 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.
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.

Capítulo 4

INTEGRACIÓN DE MONGODB Y MATPLOTLIB EN KIBOTICS

En este capítulo se describe el estado inicial de Kibotics Webserver, tanto arquitectura de la aplicación web como tecnologías ya utilizadas. Además, se explica la implementación de MongoDB como base de datos, y de Matplotlib como generador de gráficas.

4.1. Estado inicial de Kibotics Webserver

En esta sección se describe la arquitectura que poseía Kibotics Webserver al comienzo del desarrollo de este proyecto.

4.1.1. Arquitectura

Kibotics Webserver es un servicio web desarrollado en Django. Compuesto por varias partes:

- Kibotics Webserver: Servicio web desarrollado en Django, es el centro de Kibotics y el generador de los logs sobre los que se trabajará en este proyecto fin de carrera.

- Kibotics websim: Simulador robótico que se ejecuta unicamente en el lado del cliente. Desarrollado en herramientas como A-Frame, HTML5, JavaScript, entre otras, permite a los usuarios aprender los fundamentos de la programacón robótica y visión artificial.

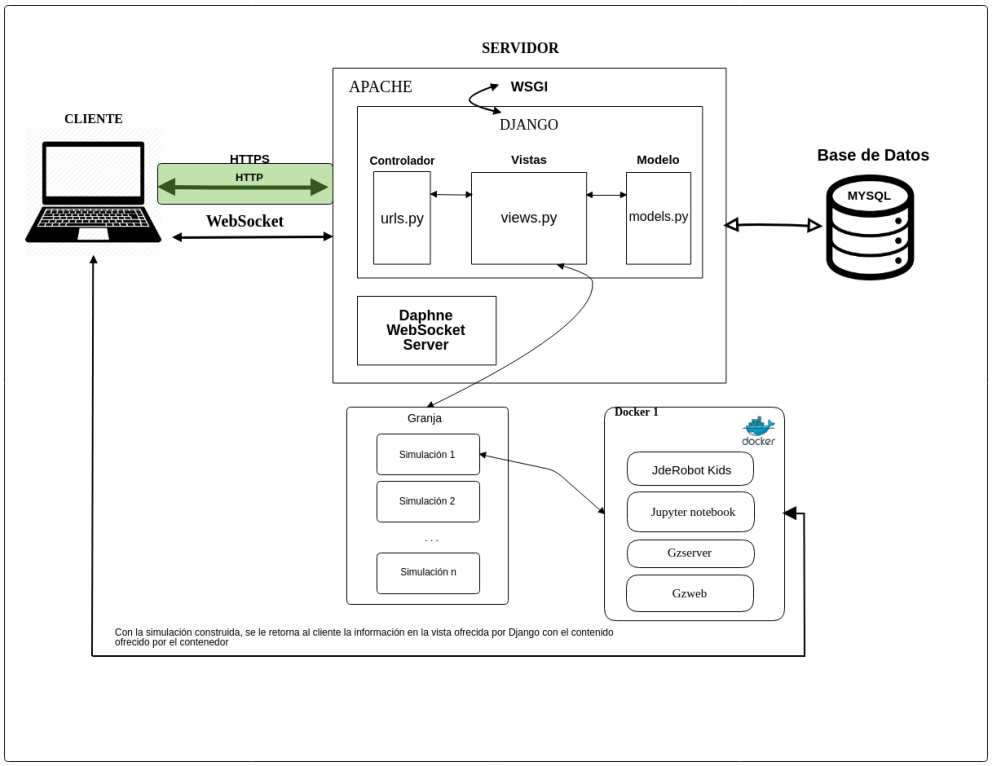


Figura 4.1: Arquitectura Kibotics.

Cuando un usuario accede a la plataforma lo hace a través del protocolo HTTPS hacia el servidor Django principal. Este servidor es el orquestador de eventos que tienen lugar en el resto de máquinas.

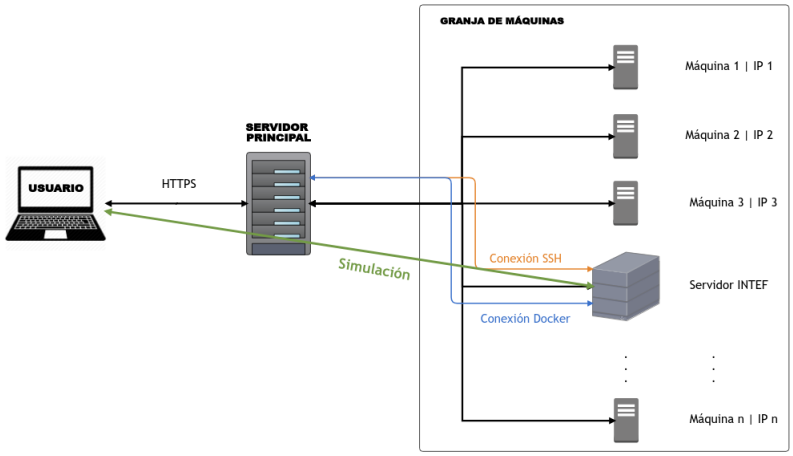


Figura 4.2: Infraestructura Kibotics.

El centro de Kibotics son las simulaciones. Una simulación está compuesta principalmente de dos partes: el código que programa el estudiante (izquierda) y la ventana de simulación (derecha), en la que el usuario verá su código ejecutado en tiempo real.

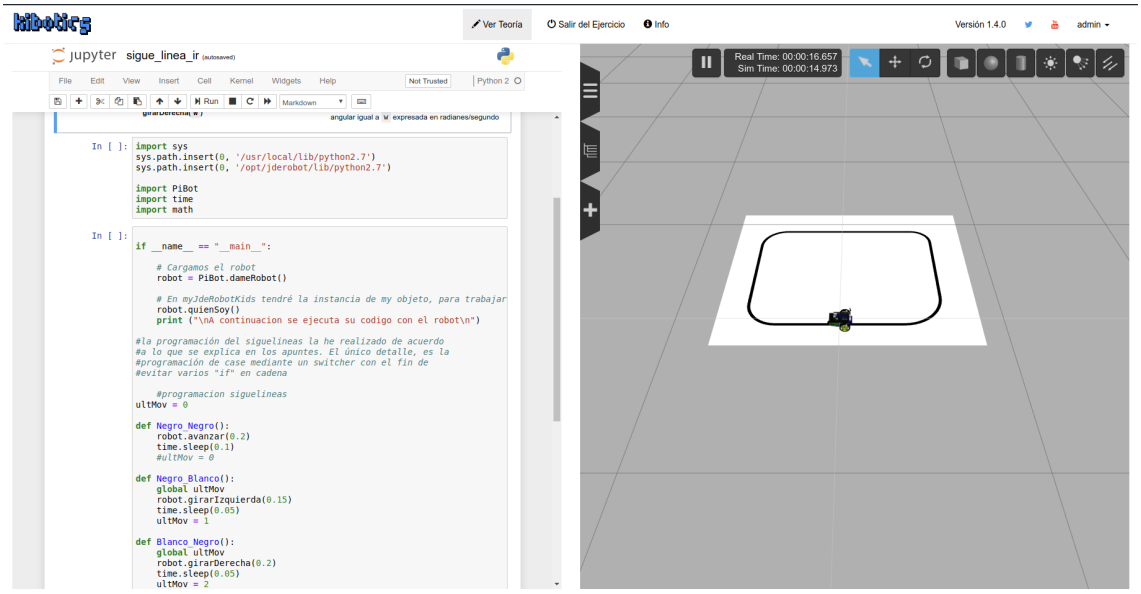


Figura 4.3: Simulador Kibotics.

4.1.2. Logs

Los logs generados por la aplicación Django se guardan en una serie de archivos indexados en el servidor con el formato YYYY-MM-DD-log.txt. Teniendo así, un fichero por día con todos los eventos registrados.

Esta metodología disponía de un sistema numeral de códigos para identificar el evento que había generado cada registro de log. Cada campo de un mismo registro estaba separado por la cadena de caracteres " | ".

Estos códigos 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 ejecicio: "3 | date | user name | user IP | simulation type | exercise ID | host IP | HTTP_USER_AGENT"
- Fin ejecicio: "4 | date | user name | user IP | simulation type | exercise ID | host IP | HTTP_USER_AGENT"
- Error 500: "5 | 500 Internal Server Error"

Estos log, se generaban en el servicio Django yse guardaban en los ficheros con sentencias Python similares a la siguiente:

```
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, se disponía de los generados de forma automática por Apache. El cual, separa los eventos registrados en dos ficheros, uno con la salida general de la aplicación, asociada a los prints y excepciones producidas. Y el otro, es un archivo de acceso al servidor que muestra las peticiones HTTP que este ha recibido.

4.2. Desarrollo local

En está sección se describe la evolución que han sufrido los logs de la aplicación. Así como una primera prueba de concepto de la herramienta de analíticas.

4.2.1. MongoDB en Kibotics Webserver

Kibotics webserver disponía de una tecnología muy primitiva de trazabilidad, con limitados eventos y una distribución en ficheros txt. La cual, limitaba la explotación masiva de estos datos para la generación de estadísticas útiles.

En Kibotics se espera un crecimiento muy notable en los usuarios. Por lo tanto, la capacidad de procesamiento, almacenamiento y consulta de los logs debía aumentar, haciendo uso de ficheros de texto plano no se podrá conseguir la velocidad de procesamiento necesaria.
Se decide cambiar a un motor de bases de datos, MongoDB, una base de datos externa a Django, a la que se realizarán consultas de Python mediante la librería pymongo. La cual ofrece las herramientas de consulta y guardado necesarias para una interacción ágil con MongoDB.

Para realizar esta migración de tecnología, es necesario primero instalar MongoDB:

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

Una vez instalado, es necesario iniciar el proceso para levantar el servicio MongoDB:

```
$ sudo systemctl start mongod
```

Adicionalmente a esto, podemos reiniciar o parar el servicio con los siguientes comandos respectivamente:

```
$ sudo systemctl restart mongod
$ sudo systemctl stop mongod
```

Con el servicio MongoDB ya instalado lo único necesario para completar la migración es modificar las sondas para evitar que escriban en ficheros. Haciendo uso de la librería pymongo la tarea se simplifica mucho. 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, las sondas se transforman a, por ejemplo, las de nueva sesión y 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,
    "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 logueo en la nueva base de datos MongoDB está completa, para recuperar la información y tratarla en python, una vez más, haciendo uso de pymongo las Queries de búsqueda serán:

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

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

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

# Query 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 queries o sentencias de búsqueda filtran tanto por usuarios como por rangos de fechas. Esto aporta mucha flexibilidad para posteriormente simplemente los datos necesarios y evitar tener q recorrer ficheros extra descartando registros de log.

4.2.2. Matplotlib en Kibotics Webserver

Matplotlib es una librería Python de generación de gráficos. Haciendo uso de ella, se han generado todas las visualizaciones necesarias para el primer prototipo.

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

Con el guardado de datos que se ha realizado surge un problema inicial, 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 hacer una relación entre ellos es necesario cruzarlos en Python para unificarlos en un único evento de sesión o simulación.
Para esto, se creó la funcionalidad para unificar estos registros para que proporcionasen información más útil:

```
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):
                    if(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
```

Este código extrae todos los usuarios y posteriormente recorre los registros de principio a fin buscando por el campo hora hasta encontrar la inmediatamente siguiente de cierre. Nos devolverá un diccionario de diccionarios con cada uno de los eventos sesión/simulación para cada usuario del que haya ocurrencias.

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 sera:

```
{
    "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 con estos datos más completos, ya se puede enviar a métodos de generación de gráficas, estos tienen una estructura muy similar entre ellos:

- Primero, recorrerán los datos de entrada formateandolos a la estructura de ejes necesaria para cada una de las gráficas. 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 el mapa de calor, necesitaremos una matriz de datos para la correcta representación de la información.

- Segundo, se creará la gráfica 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 gráfica se insertará para cada caso. Junto a la primera parte, es lo que más cambiará entre métodos.

- Tercero, se ajustará el diseño de la gráfica para que encaje estéticamente tanto con las otras gráficas generadas para la funcionalidad de analíticas, como con el diseño ya existente en la aplicación.

- Finalmente, ya generada la gráfica, 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.

Con estos objetos de imagen ya guardados, lo único que queda es devolverlos por el contexto de la respuesta HTML de Django, que es simplemente un diccionario de variables. Es a este contexto al que la platilla HTML enriquecida de Django accederá para mostrar las imágenes.

Un ejemplo de una sección en estas plantillas será:

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

    <div class='left' >
        <h4>Sesiones</h4><br>
        <img class='analyticsIMG'src=" {{ imageWeekSesion }}"
        alt="Data not loaded properly" width=500 height=auto />
    </div>

    <div class='right'>
        <h4>Simulaciones</h4><br>
        <img class='analyticsIMG'src=" {{ imageWeekSimulation }}"
        alt="Data not loaded properly" width=500 height=auto />
    </div>
</div>
...
```

A continuación se puede ver el resultado final, con unos datos de prueba, no productivos. El prototipo nos muestra mucha información acerca de la actividad en la aplicación web.

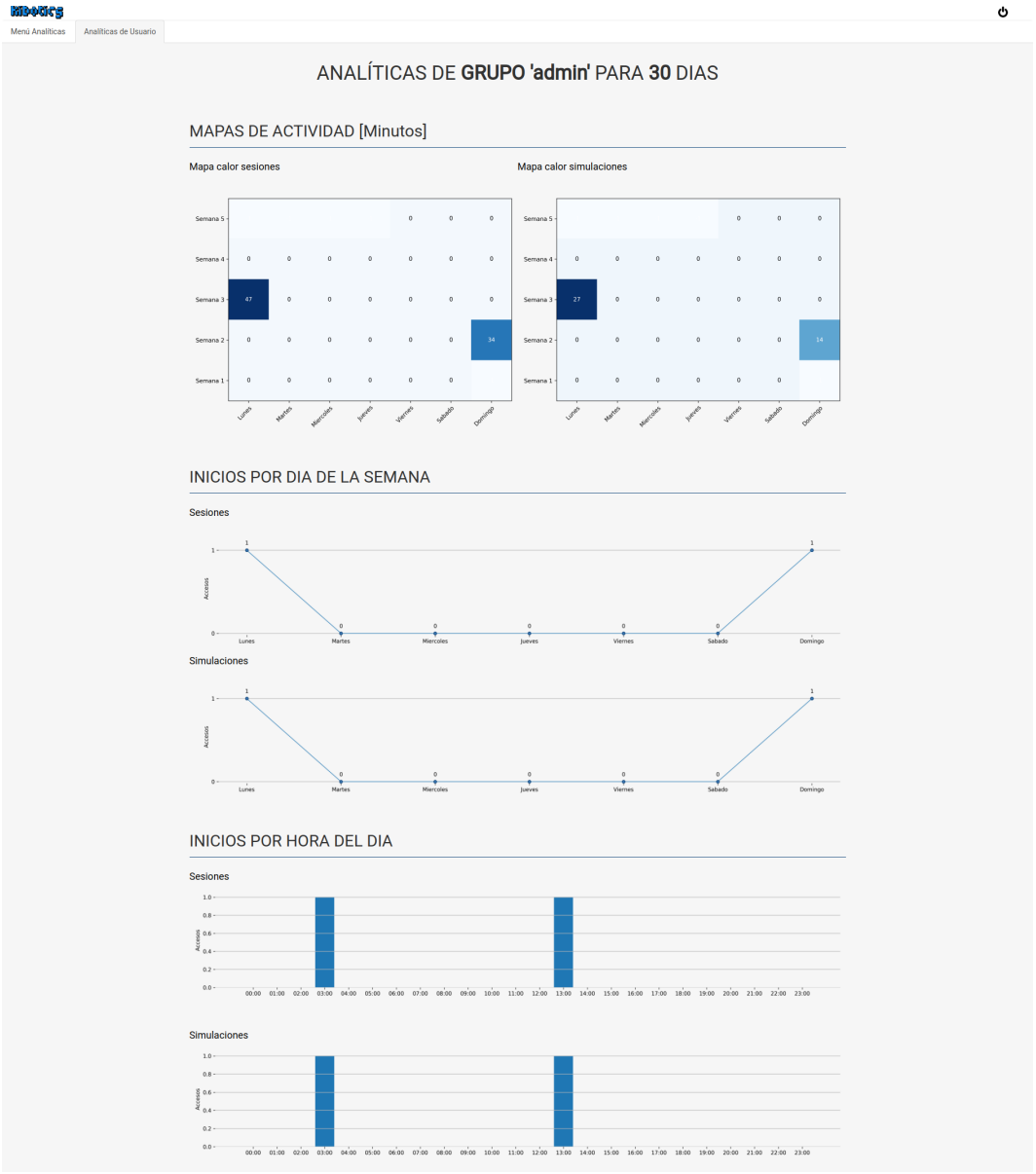


Figura 4.4: Primer prototipo parte 1.

En la primera figura, se puede observar una primera parte con un mapa de calor tanto para simulaciones como para sesiones, la cual representa la actividad en minutos para el grupo de usuarios analizado.

En una segunda parte, se representar 4 gráficas 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.

Por último en la siguiente figura, se representan las dos últimas gráficas. Una primera 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, la última gráfica representa con un mapa geográfico la localización desde la que cada acción se ha realizado.

Este primer prototipo es bastante completo pero tiene ciertos inconvenientes. Primero, falta cierta información útil que podría ser representada, como desde que dispositivos acceden los usuarios.

Esta gráficas, al estar insertadas como imágenes, carecen de interactividad, la cual sería muy útil para tener información extra o poder realizar mas filtrado de los datos.

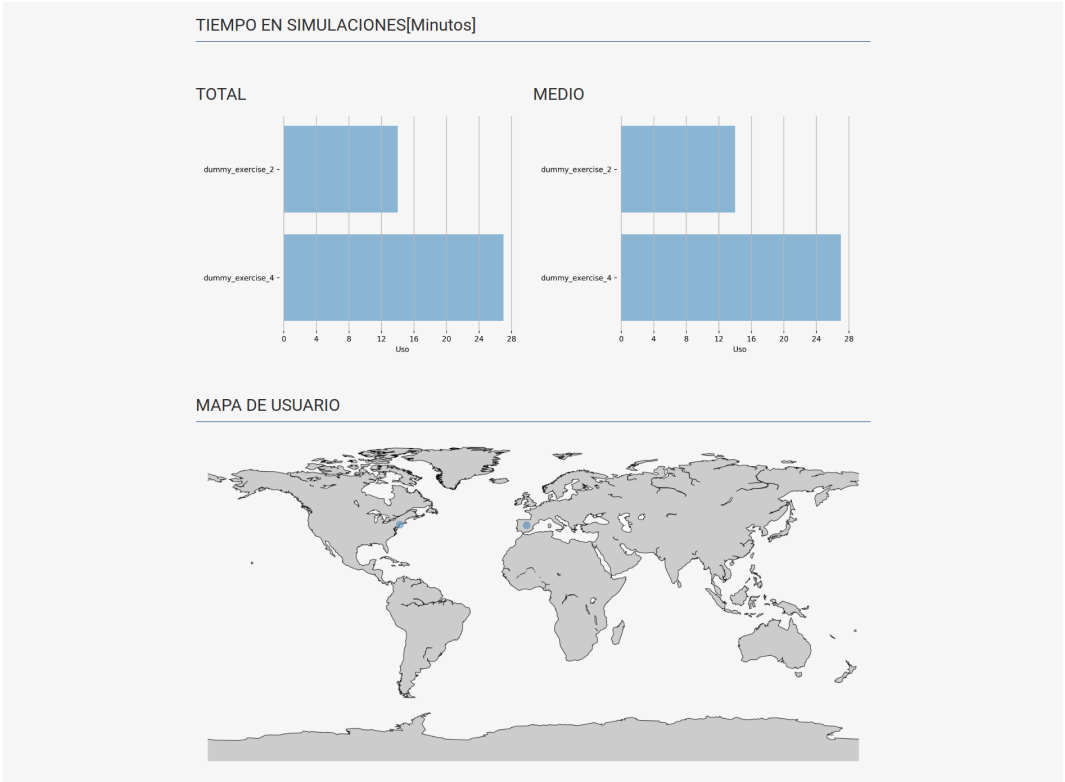


Figura 4.5: Primer prototipo parte 2.

Capítulo 5

INTEGRACIÓN DEL STACK ELK EN KIBOTICS

En este capítulo se describen las tecnologías del Stack ELK utilizadas en la versión final desarrollada del módulo de analíticas. Tanto Elasticsearch como base de datos, como Kibana de procesador de datos y generador de gráficas. Además de unos recursos de prueba generados para que futuros desarrolladores dispongan de una instalación sencilla de bases de datos y servicios necesarios.

5.1. Desarrollo local

En está sección se describe la evolución que han sufrido los logs de la aplicación. Así como la versión final de la herramienta de analíticas que hará uso del Stack ELK.

5.1.1. ElasticSearch en Kibotics Webserver

El primer paso el proceso de migración es el cambio de bases de datos. Para hacer uso del Stack ELK, es necesario sustituir MongoDB por Elastic-Search. Para esto lo primero será instalar e inciar un servicio de Elasticsearch en local.

Una vez iniciado el servicio Elasticsearch podremos acceder a los datos indexados en las tablas mediante la interfaz a la que se podrá acceder por la terminal de comandos o de manera más amigable, desde la URL y puerto local que hayamos configurado en la instalación de ElasticSearch.

Un ejemplo de llamada para un índice llamado index_name_test será la siguiente:

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

El siguiente paso en la migración es la integración de Elasticsearch en Django, se realizará haciendo uso de la librería django_elasticsearch_dsl, la cual simplificará esta tarea. La cual, se divide en dos partes principales. Creación de los índices y migración de las sondas de MongoDB a Elasticsearch.

Este primer paso de creación de índices es similar a la metodología de modelos que posee Django con las estructuras de las bases de datos. Se creará los siguientes índices:

- kibotics_session_log: En el cual se almacenarán todos los eventos referentes a las sesiones.
- kibotics_simulation_log: En el cual se almacenarán todos los eventos referentes a las simulaciones.
- kibotics_error_log: En el cual se almacenarán todos los eventos referentes a los errores.
- kibotics_visit_log: En el cual se almacenarán todos los eventos referentes a las visitas.

Aprendiendo de lo aplicado en el primer prototipo y para evitar tener que cruzar datos para el calculo de duraciones de los eventos logueados, tanto en el índice de sesiones como de simulaciones se ha eliminado el campo que registraba la fecha. Para sustituirlo, se han añadido dos nuevos campos los cuales registrarán el inicio y fin de cada evento, unificando así los dos registros de log que se tenían previamente en uno.

Por otro lado el campo USER_AGENT ha sido dividido y sustituido con la información del navegador, dispositivo y sistema operativo con el que cada usuario accederá a la web.

Para trabajar en Kibana con mapas es necesario guardar tanto la longitud como la latitud, para lo cual se creará un campo ya existente llamado Geo Point que almacenará en un diccionario ambos campos.

Para complementar las nuevas sondas, de las que se a continuación, es necesario la creación de un nuevo índice de visitas, en el que se registrarán los accesos a la pagina principal de la aplicación, estén o no registrados.

Un ejemplo de la estructura de uno de estos índices en el fichero Python 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
        }
```

Por último, ya solo será necesaria la migración de las sondas de obtención de logs. Para enriquecer las sondas ya existentes, se han añadido nuevas para registrar eventos de visitantes, así como sondas para fin de sesiones/simulaciones (cierre busco de la aplicación, temporizador por inactividad) con el fin de evitar entradas de log sin fecha de fin.

Las sondas de inicio de los eventos son similares a las que se tenían anteriormente en MongoDB, por ejemlo:

```
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, las sondas de fin de sesión/simulación cambian. Tendrán que buscar en ElasticSearch el último registro de log en el índice para el usuario del cual se quiera actualizar el registro de log y sustituir tanto los

campos end_date como duration.
Esto se realiza gracias al identificador que cada registro indexado posee al cual se le realiza una operación update con los nuevos campos:

```
latest_session = Search(index="kibotics_session_log*") \
    .query("match", username=username) \
    .query('match', duration=0) \
    .sort({"start_date": {'order': 'desc'}})[0]

# Update session with leave date and 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()
                    }
                }
            )
```

Para comprobar que todos estos campos están siendo creados y actualizados correctamente se puede realizar haciendo consultas tanto por terminal, así como a la API Rest de ElasticSearch que se levantó previamente, accediendo mediante la URL y puerto configurados durante el proceso de instalción.

En esta API, podremos realizar filtrados por campos e índices haciendo uso de expresiones regulares Regex, un ejemplo genérico para uno de los índices de kibotics será:

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

En la siguiente figura se puede observar la respuesta que se obtiene a la llamada anterior.

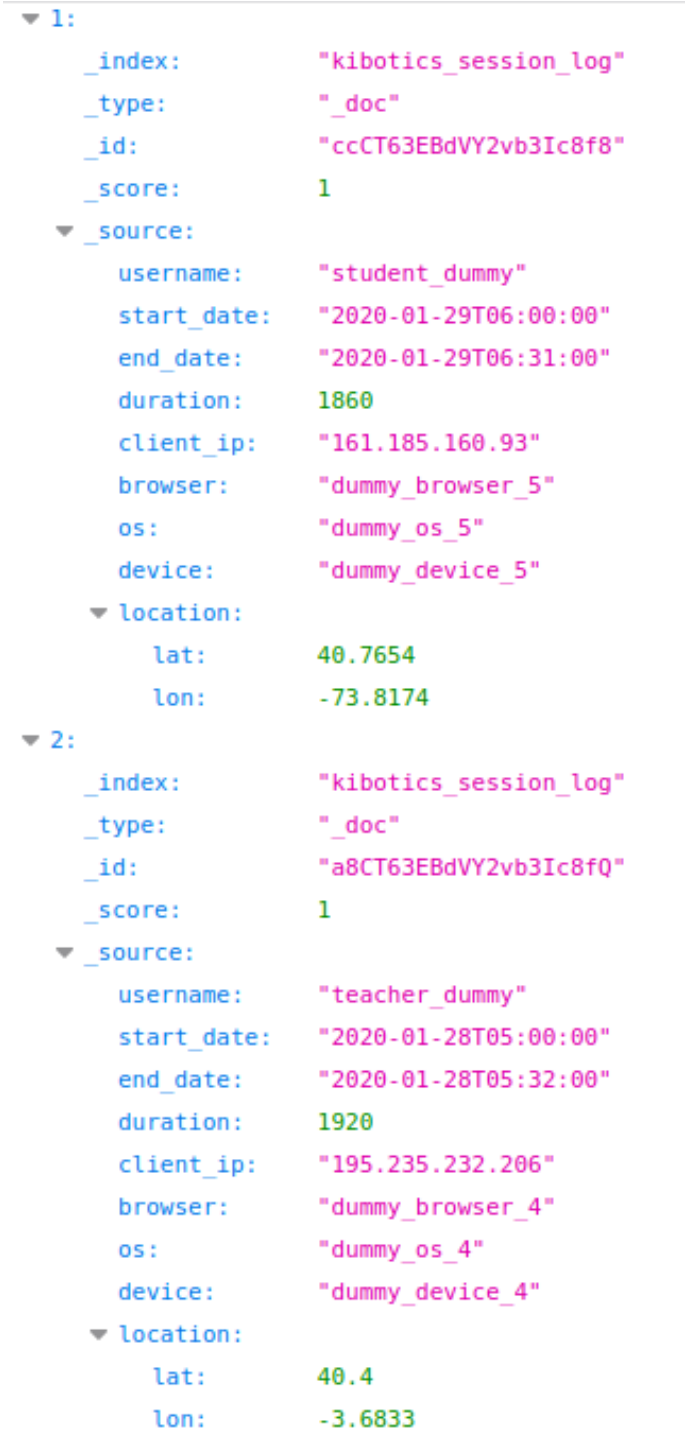


Figura 5.1: API Rest ElasticSeach.

5.1.2. Kibana en Kibotics Webserver

Para acceder a kibana, deberemos primero instalar y ejecutar el servicio. 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, Kibana nos pedirá que registremos los índices sobre los que queremos obtener soporte y seleccionemos el campo sobre el que se filtrarán temporalmente estos logs. En este caso, el campo start_date.

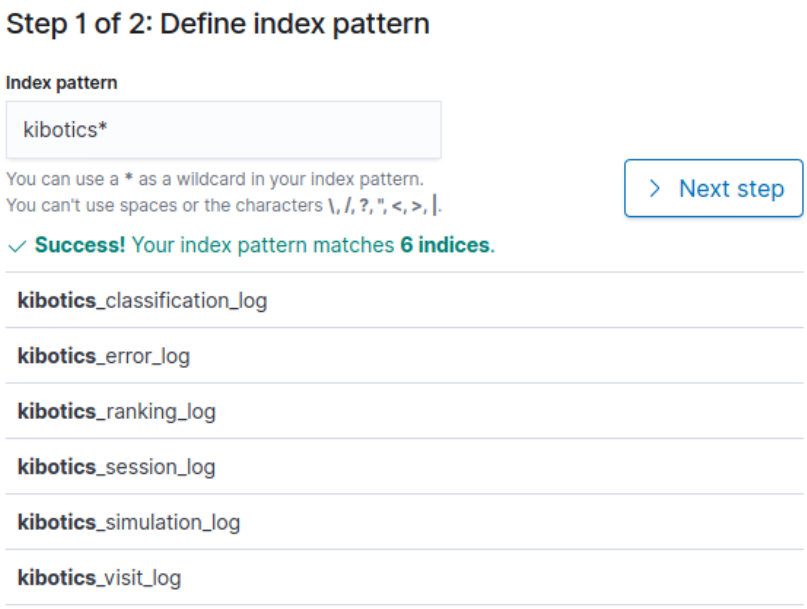


Figura 5.2: Creación de índices en Kibana.

Configurados todos los índices en Kibana, se podrá tener una primerá visualización de los datos indexados en ElasticSearch en la pestaña Discover de Kibana. En esta pestaña podremos ver todos los logs que estén guardados así como un histograma en el que se podrá ver gráficamente la evolución de los índices. 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.

Esta sección ya nos da una primera pincelada de la potencia de procesamiento de Kibana, así como la sencillez de implementación y despliegue.

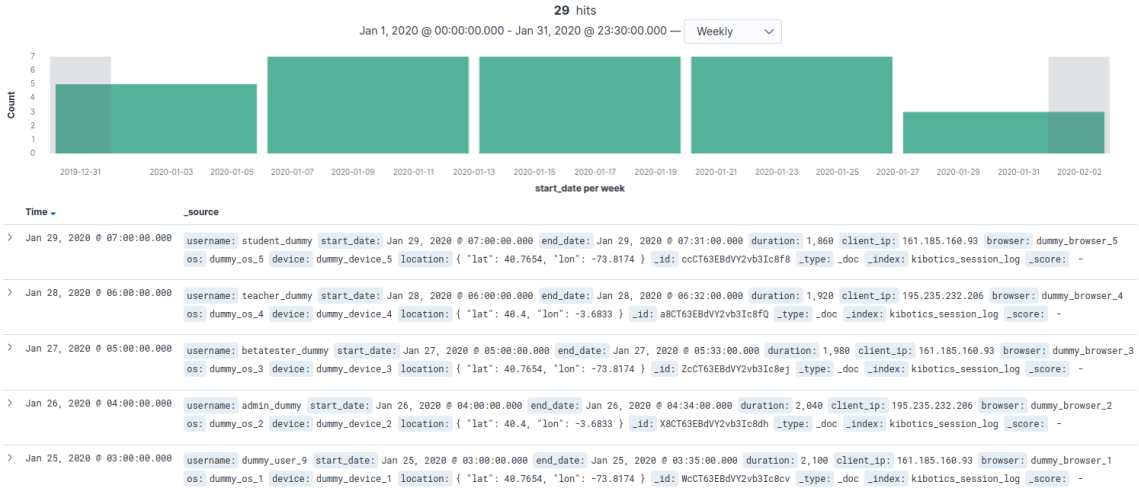


Figura 5.3: Sección Discover en Kibana.

Kibana ofrece la posibilidad de creación de scripted fields, estos son, campos cuyo valol derivará de otros campos o datos ya indexados, generados en un legunaje muy similar a C llamado painless. Para la versión final de la herramienta necesitaremos crear varios de estos campos para distintas gráficas que se explicarán más adelante como por ejemplo el mapa de calor de actividad.

Estos son el día de la semana y la hora del día en que los logs se registraron.

```
# 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 campos creados además de los contenidos en los registros de log, Kibana ya dispone de todos los datos necesarios para la creación de las visualizaciones. Para ellos en la sección Visualize, kibana tiene una colección de distintas plantillas gráficas. Las cuales, deberán ser configuradas para representar los datos e índices que sean necesarios.

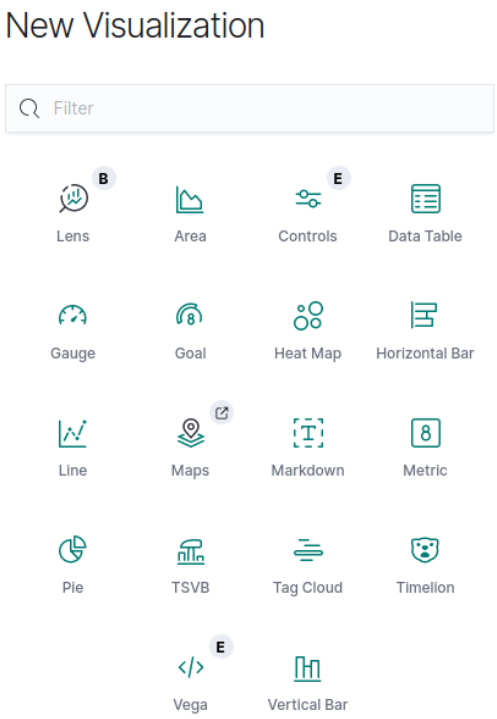


Figura 5.4: Menú creación de visualizaciones.

Se han creado una amplia selección de visualizaciones. Divididas en dos Dashboards, el primero reservado a los visitantes y el segundo para analíticas de sesiones y simulaciones de usuarios registrados. A continuación se mostrarán las gráficas creadas, así como una breve explicación de lo que representan.

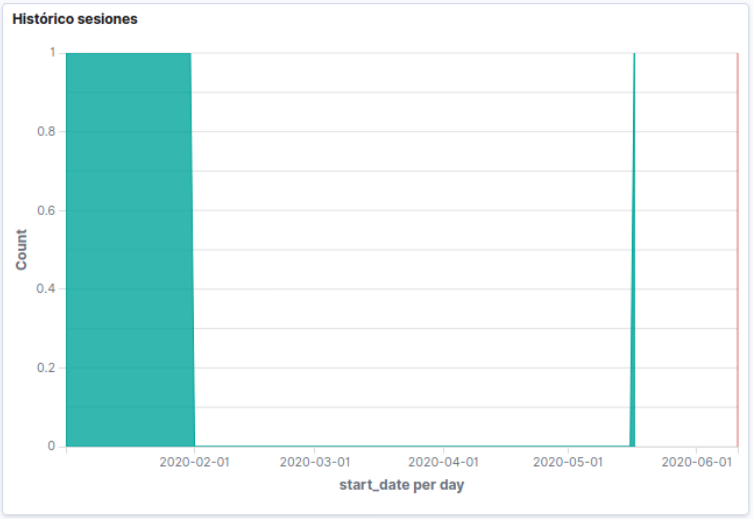


Figura 5.5: Histograma

Histograma que representará el número de logs registrados en el índice de sesiones para el rango de fechas señalado.

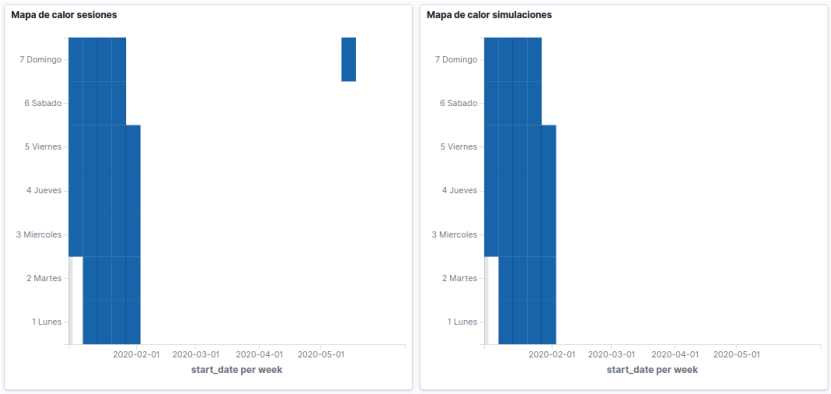


Figura 5.6: Mapa de calor sesiones y simulaciones

Mapas de calor para los índices se sesiones y simulaciones, divididos en columnas por semanas, cada una de ellas en los respectivos días de la semana. Un color más oscuro representará mayor actividad para ese día.

Figura 5.7: Gráfico de barras por día de la semana

Gráfica de barras que dividirá los datos filtrados por el día de la semana en que se registraron. Tanto para sesiones como para simulaciones.

Figura 5.8: Gráfico de barras por hora del día

Gráfica de barras que dividirá los datos filtrados por la hora del día en que se registraron. Tanto para sesiones como para simulaciones.

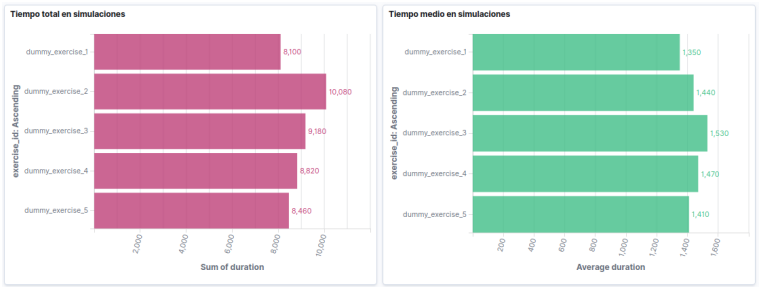


Figura 5.9: Gráfico de barras para tiempo total y medio en simulaciones

Gráfica de barras que representará el tiempo invertido por los usuarios filtrados en las respectivas simulaciones. El gráfico de la izquierda muestra el tiempo total y el de la derecha el tiempo medio invertido.

Figura 5.10: Mapa geográfico de sesiones

Representación geográfica de los eventos de sesión ocurridos para el rango de fechas filtrado.

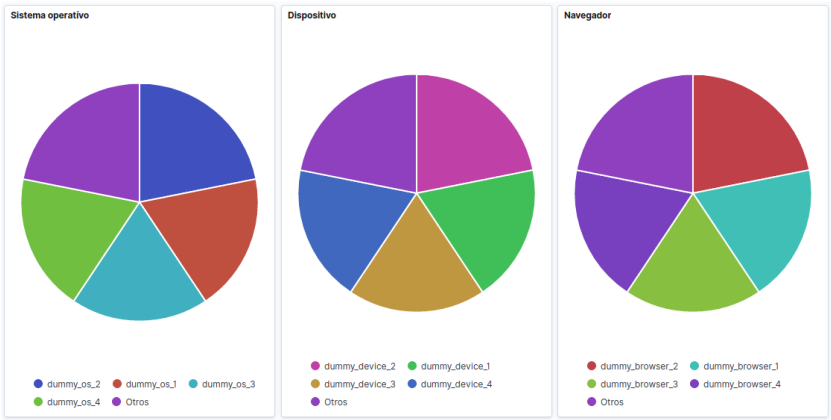
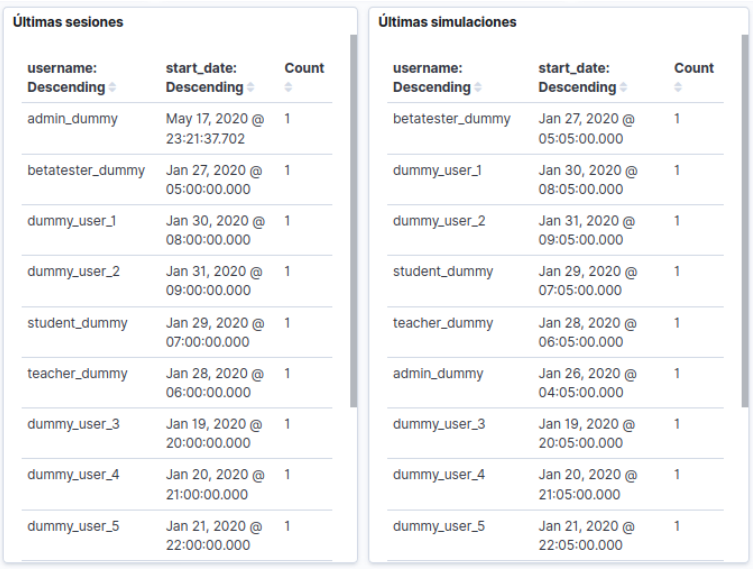


Figura 5.11: Gráficas circulares para SO, Dispositivo y Navegador

Tres gráficas circulares que representarán información acerca de los dispositivos que están siendo utilizados para acceder a la aplicación, filtra por sistema operativo, dispositivo y navegador.



Últimas sesiones

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

Últimas simulaciones

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 5.12: Últimos eventos logueados

Tabla de datos con los últimos registros de sesión y simulación para cada uno de los usuarios filtrados.

Todas estas visualizaciones son interactivas y se puede filtrar por sus campos simplemente pulsando sobre ellas. Funcionalidad muy útil que no poseían las imagenes renderizadas que se generabane n el primer prototipo.

Ya creadas las visualizaciones, solo quedará integrarlas en Kibotics al igual que se hizo con las generadas en Matplotlib. Para ello se creará una vista simple con la que se seleccionará a que tipo de analíticas se quiere acceder.

Figura 5.13: Menú de selección de analíticas en Kibotics

En esta vista se filtrará tanto por usuarios y grupos, como por fechas de las cuales queremos analíticas. Una vez filtrado, Django generará automaticamente 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 que lo insertarán en un elemento HTML iFrame.

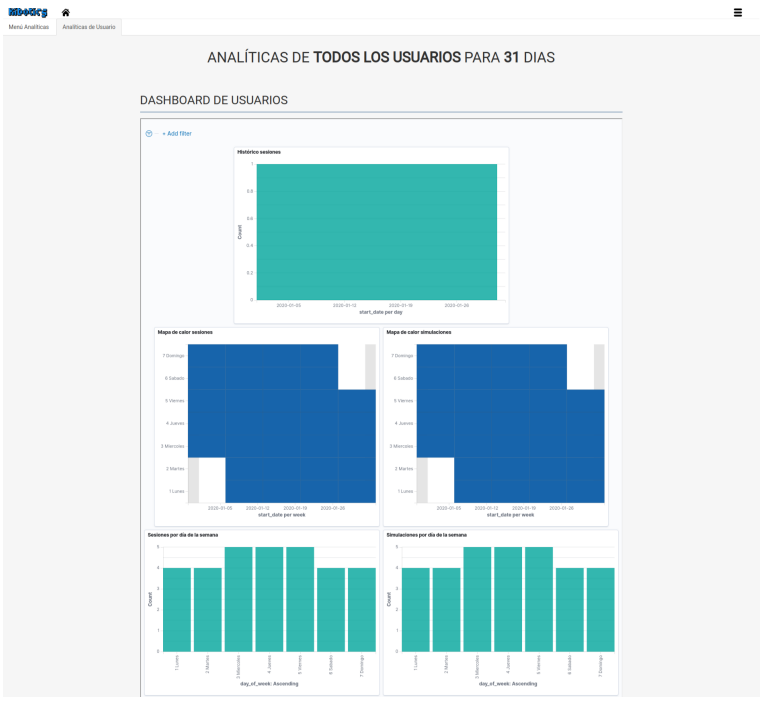


Figura 5.14: Kibana en Kibotics parte 1

Figura 5.15: Kibana en Kibotics parte 2

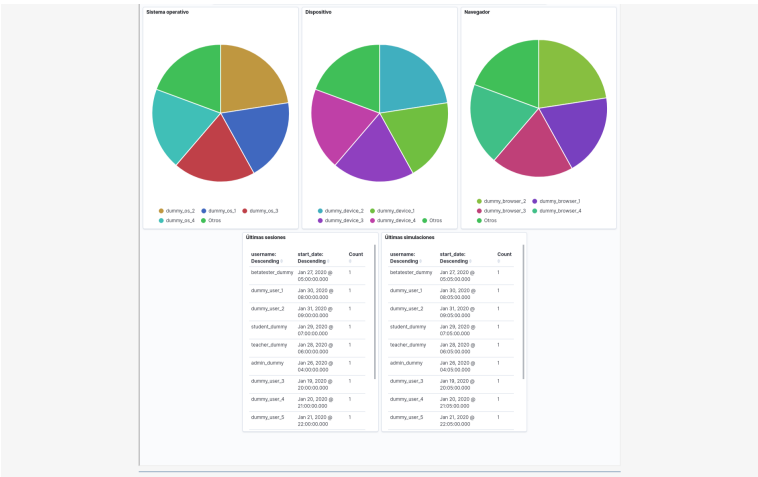


Figura 5.16: Kibana en Kibotics parte 3

5.2. Despliegue en producción

(INCOMPLETO)

5.3. Generación de recusos de prueba para el desarrollo local

Para facilitar futuros desarrollos y ampliaciones de las mejoras de analíticas implementadas se han generado una serie de recursos para proveer al repositorio Github de herramientas.

5.3.1. Receta de instalación de ElasticSearch

Se ha incluido en el repositorio GitHub la receta necesaria para instalar la versión de ElasticSearch utilizada en producción.

Con esto, se asegura que cualquier futuro desarrollador de la plataforma simplemente tenga que seguir la receta de instalación ahí explicitada para tener un entorno totalmente funcional.

Actualmente la última versión estable de ElasticSearch es la versión 7.7.2, liberada el 3 de Junio de 2020. Por cuestión de fechas, en producción, actualmente se está utilizando la versión 7.6.2 del 31 de Marzo del 2020.

5.3.2. Creación de bases de datos Dummy para Elasticsearch

Para proporcionar a los futuros desarrolladores datos realistas con los que poder empezar a trabajar sin necesidad de 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.

Para la creación de esta base de datos de pruebas, se generó mediante un script de Python el cual crea los índices y su estructura haciendo uso de la librería de 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
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 guar-

dar todos los objetos con la información de prueba que se deseé guardar.

Finalizada la ejecución del script ya tendríamos los datos en nuestro servicio local de Elasticsearch. Un problema que se encontró es que para ejecutar este script es necesaria la instalación de varias librerías. Para simplificar aún 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_elasticseach.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 tendrá que instalar elasticdump y ejecutar un script bash el cual recorrerá y cargará 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
```

5.3.3. Receta de instalación de Kibana

Para completar la instalación de recursos del Stack ELK utilizados en el proyecto se ha añadido la receta de instalación de Kibana.

Actualmente la última versión estable de Kibana es la versión 7.7.1, liberada el 3 de Junio de 2020. Por cuestión de fechas, en producción, actualmente se está utilizando la versión 7.7.0 del 13 de Mayo del 2020.

5.3.4. Creación de bases de datos Dummy para Kibana

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.

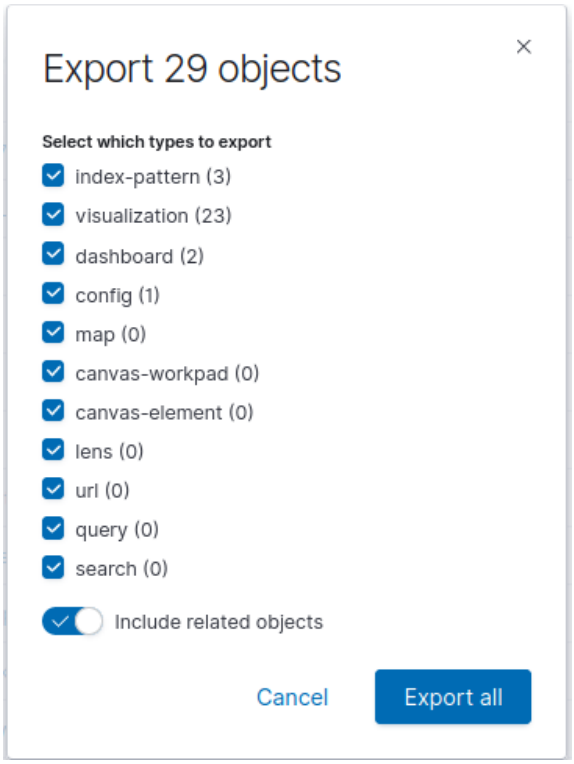


Figura 5.17: Exportación en interfaz Kibana.

Esta herramienta nos generará un fichero NDJSON similar a la estructura JSON con los patrones de índices creados, así como las visualizaciones, tablas o campos scripted 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

Capítulo 7

REFERENCIAS

Django https://www.djangoproject.com/ Django MVC https://uniwebsidad.com/libros/django-1-0/capitulo-5/el-patron-de-diseno-mtv

HTML https://uniwebsidad.com/libros/xhtml/capitulo-1/breve-historia-de-html https://www.w3schools.in/html-tutorial/history/

SQLite https://www.sqlite.org/about.html

Elasticsearch http://www.arquitectoit.com/elasticsearch/que-es-elasticsearch/ https://www.ionos.es/digitalguide/servidores/configuracion/que-es-elasticsearch/ https://www.elastic.co/guide/en/elastic-stack-get-started/7.6/get-started-elastic-stack.html#install-elasticsearch

Matplotlib https://matplotlib.org/

kibana https://www.elastic.co/es/what-is/kibana

Elasticdump https://www.npmjs.com/package/elasticdump

Capítulo 8

IMPLEMENTACIÓN DE MEJORAS PARA HERRAMIENTAS DE GESTIÓN

8.1. Asignación de permisos individuales