UNIVERSIDAD
REY JUAN CARLOS

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: (INCOMPLETO)

Secretario: (INCOMPLETO)

Vocal: (INCOMPLETO)

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2020

Dedicado a

mi familia y amigos

II

Agradecimientos

(INCOMPLETO)

Resumen

(INCOMPLETO)

Summary

(INCOMPLETO)

Índice general

Índice de figuras

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) **desarrolar la funcionalidad analíticas en una plataforma educativa** 1 expandir los datos grabados 2 grabarlos en una base de datos moderna 3 que se visualicen automaticamente

primera solucion - 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. El la figura 3.1 se muestran las 3 tecnologías web básicas utilizadas.

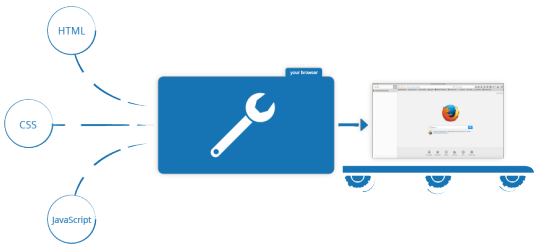

Figura 3.1: Tecnologías web básicas.

3.1.1. HTML

HTML (Hipertextual Markup Lenguaje) [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.

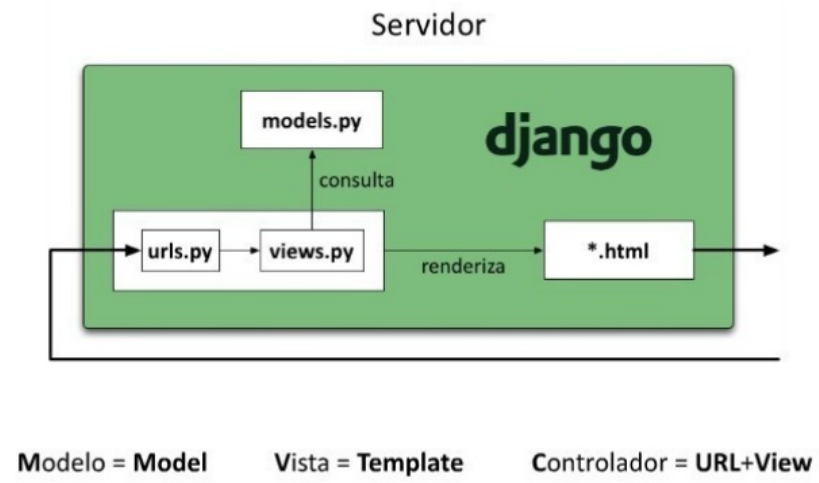

Figura 3.2: Patrón MVT Django.

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.

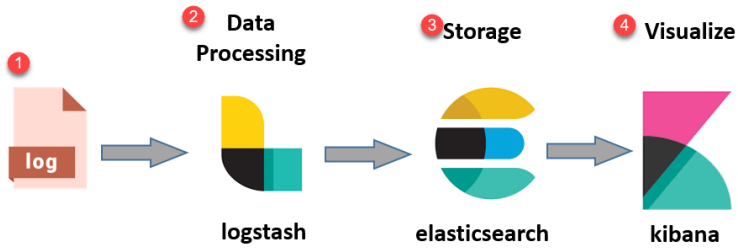

Figura 3.3: Stack ELK.

Basado en Lucene (API para recuperación de información), permite almacenar información compleja como datos de geolocalización así como realizar búsquedas de texto y autocompletado 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 tambien 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 indices 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.

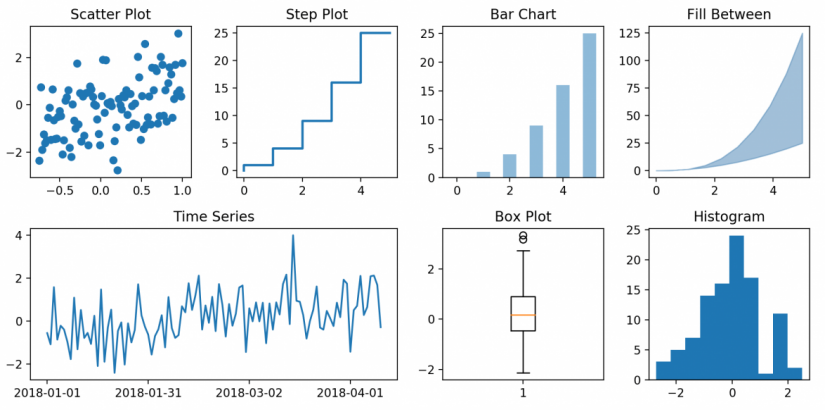

Figura 3.4: Visualizaciones en Matplotlib.

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` <img></img>

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.

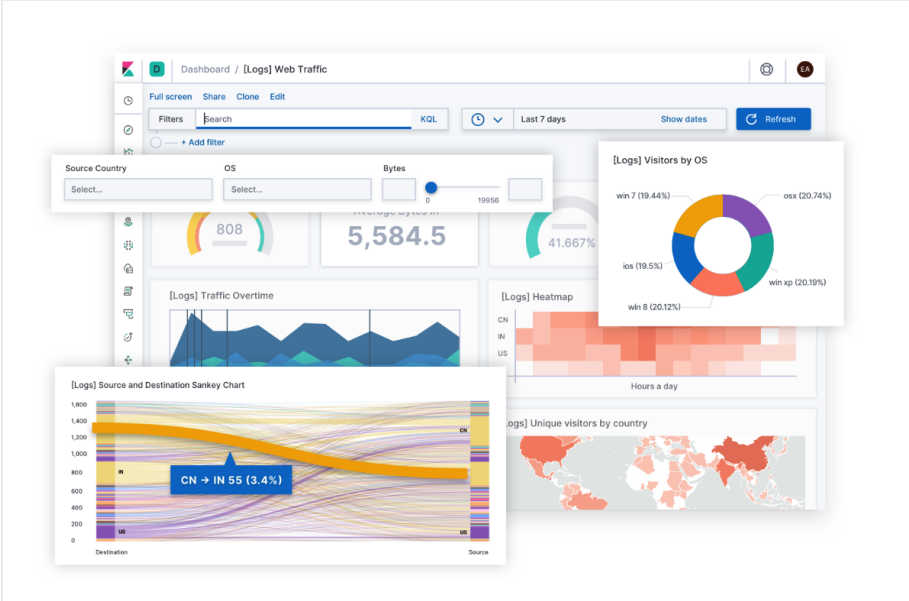

Figura 3.5: Visualizaciones en Kibana.

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.

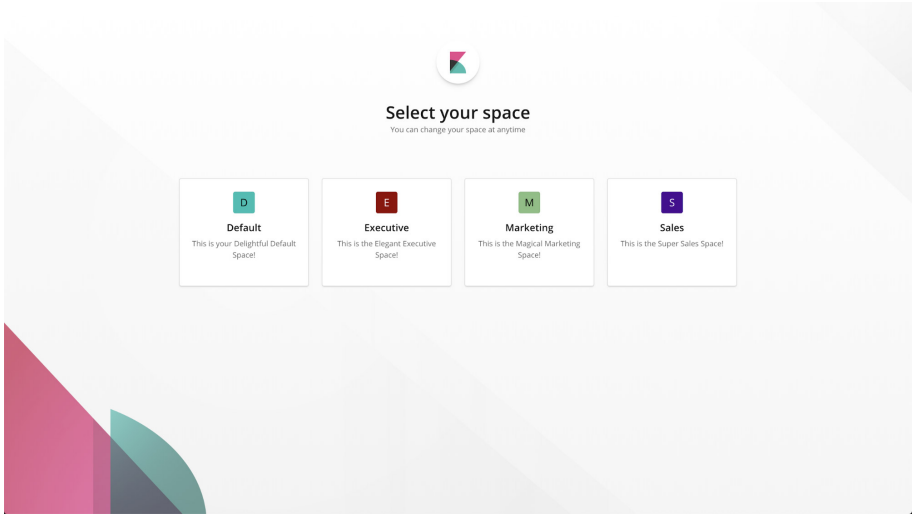

Figura 3.6: Selector de espacios de trabajo en Kibana.

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

Primer prototipo

En este capítulo se describe el estado inicial de Kibotics Webserver, tanto arquitectura de la aplicación como tecnologías en las que está desarrollado. Además, se define el diseño que será implementado para el primer prototipo de la herramienta de análisis. 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 Webserver.

4.1. Antecedentes

En esta sección se describe la arquitectura que poseía Kibotics Webserver al comienzo del desarrollo de este proyecto. Así como un caso de uso típico en la plataforma (INCOMPLETO)

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 reales. 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 y última vista será 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, este se ejecuta en el cliente del servicio.

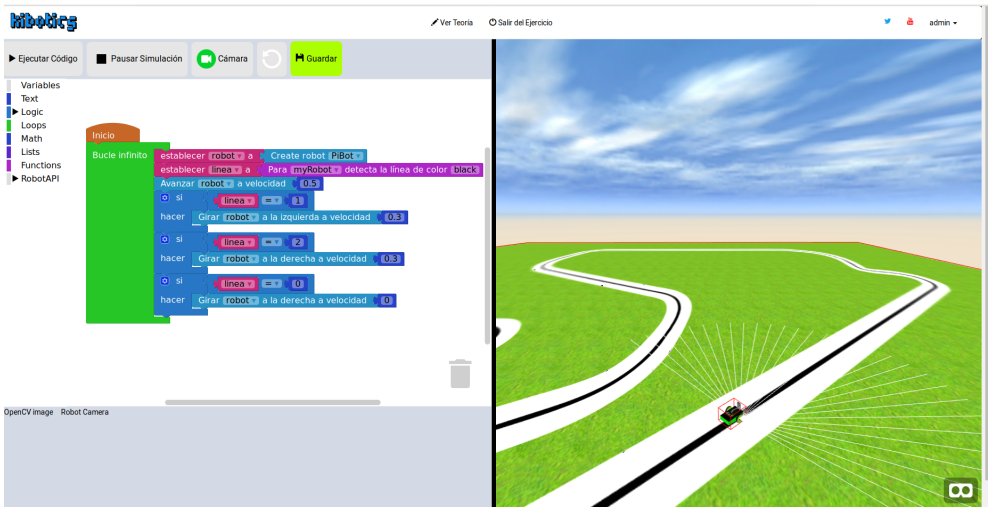

Figura 4.1: Editor y simulador en Kibotics.

Una vez resuelto el problema, Kibotics generará 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 4.3 muestra la arquitectura que poseía Kibotics inicialmente, en ella podemos observar:

- Kibotics Webserver: Servidor web desarrollado en Django, es el centro de Kibotics y donde se generan los logs sobre los que se trabajará en este proyecto fin de carrera. Se apoya en una base de datos MySQL para almacenar todos los datos de información estructural de la aplicación y en unos ficheros `*.txt` en los que se almacena información circunstancial de uso de la aplicación.

- Kibotics Websim: Simulador robótico que se ejecuta unicamente 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`

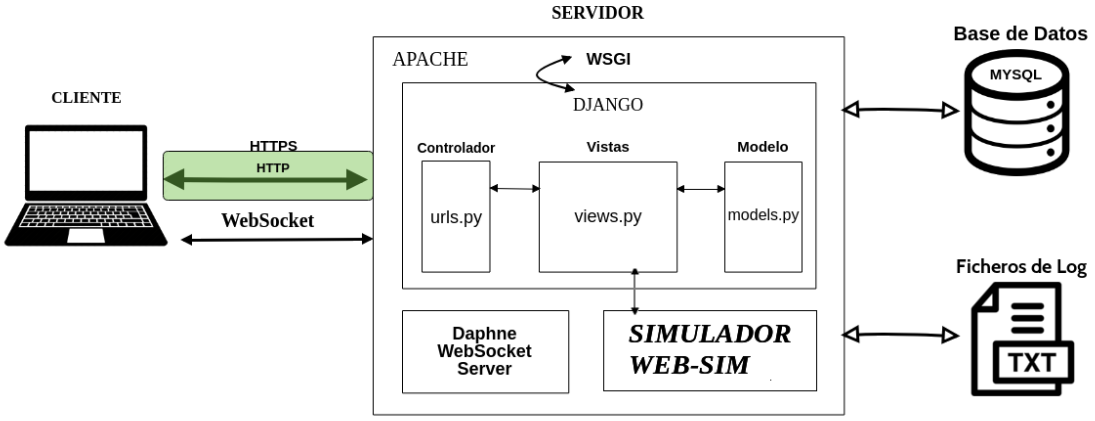

Figura 4.2: Arquitectura Kibotics.

Los 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.

Esta metodología 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 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 eventos logueados se generaban en el servidor Django haciendo uso de sondas Python, las cuales registran el evento en distintas partes del código 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 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 que pretende dar soporte 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 proce-

samiento necesaria.

Es por esto, por lo que en un primer prototipo se decidió cambiar a un motor de bases de datos. Se ha decidido usar MongoBD, es una base de datos NoSQL que permite almacenar información de forma estructurada. 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 muy 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.

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 Python llamado `analitycs.py`. 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 siguiente figura 4.3 se muestra de forma esquemática los cambios que Kibotics Webserver va a sufrir en este primer prototipo, 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.

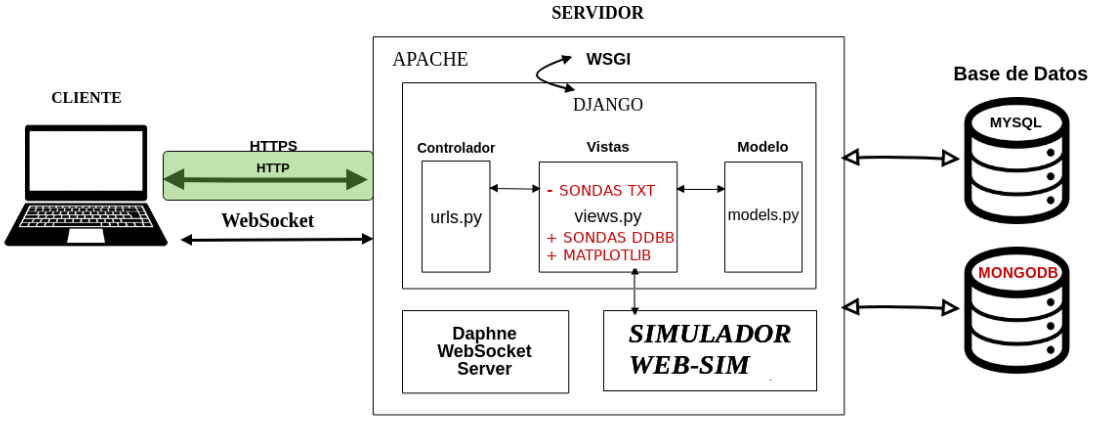

Figura 4.3: Arquitectura Kibotics.

4.3. Implementación

En esta sección se detallan los pasos que ha sido necesario realizar para el desarrollo 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 Kibotics 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, para ello se ejecutará 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 los 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 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 Kibotics Webserver

Matplotlib es una librería Python de generación de visualizaciones tanto estáticas, como animadas. 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.

Siguiendo la metodología detallada en las secciones anteriores surge 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 hacer una relación entre 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 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 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 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 se ha enriquecida la información disponible, ya se puede enviar a métodos de generación de visualizaciones. Se hace uso de la librería Matplotlib para crear diversas tipologías de visualizaciones:

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 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 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.

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 una de los motivos por el que se haga un futuro cambio de tecnologías.

Con estos objetos de imagen ya guardados, lo único que queda es devolverlos al cliente. Para ello se utilizarán las plantillas HTML enriquecidas de Django. A continuación, se muestra una sección de una de estas plantillas en las que se puede ver como se han insertado dos imágenes:

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

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

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

4.4. Validación experimental

En este capítulo se puede ver el resultado final del primer prototipo desarrollado con unos datos de prueba, no productivos. El prototipo nos muestra información de interés acerca de la actividad en la aplicación web.

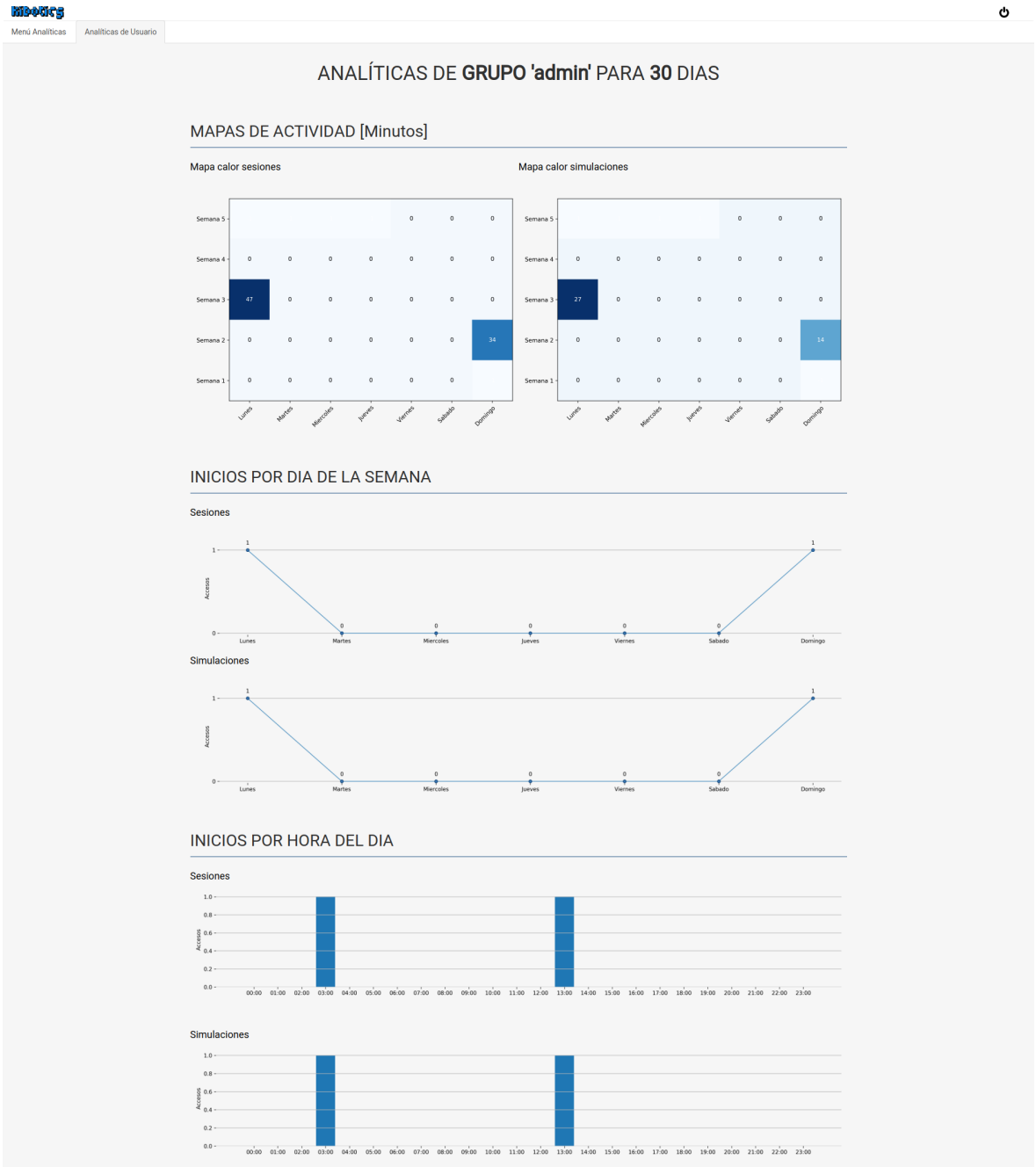

Figura 4.4: Primer prototipo parte 1.

En la figura anterior 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 para los distintos eventos de sesión y simulación. El color será más intenso a mayor 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 siguiente 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 aplicación.

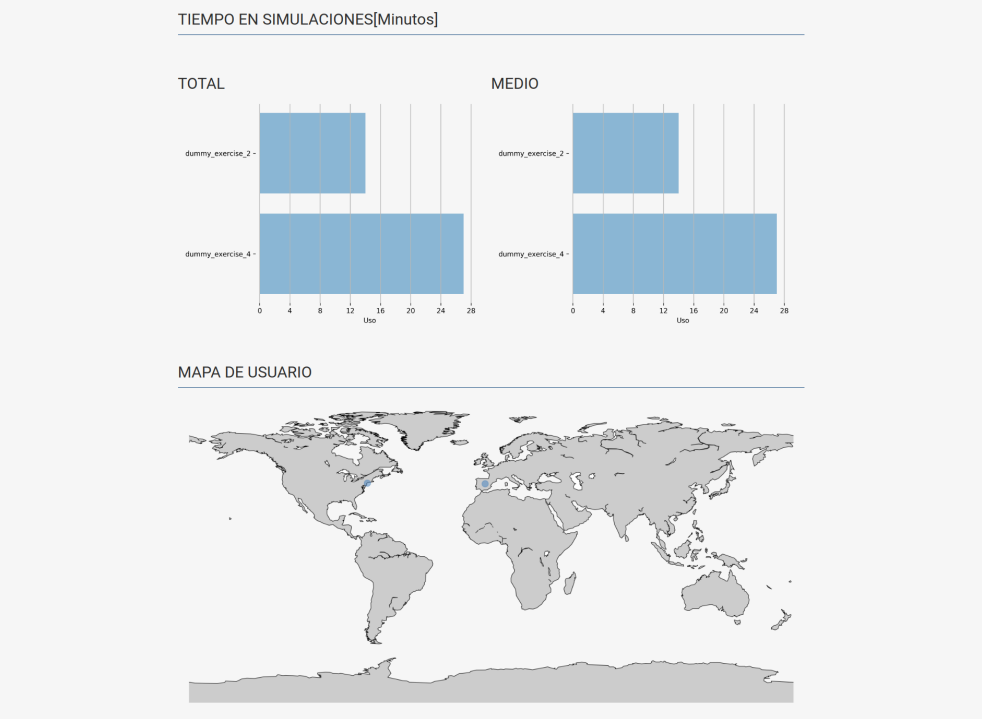

Figura 4.5: Primer prototipo parte 2.

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 o la actividad de los usuarios no registrados.

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

Además el módulo, al tener que renderizar cada una de las imágenes individualmente, tarda unos segundos en mostrar las visualizaciones.

Capítulo 5

Segundo prototipo

(INCOMPLETO)

5.1. Diseño

(INCOMPLETO) cambiar cosas sobre la figura del cap 4 ahora ddbb elasticsearch xk ahora se conecta a kibana para la visualizacion xk descripciona panoramica

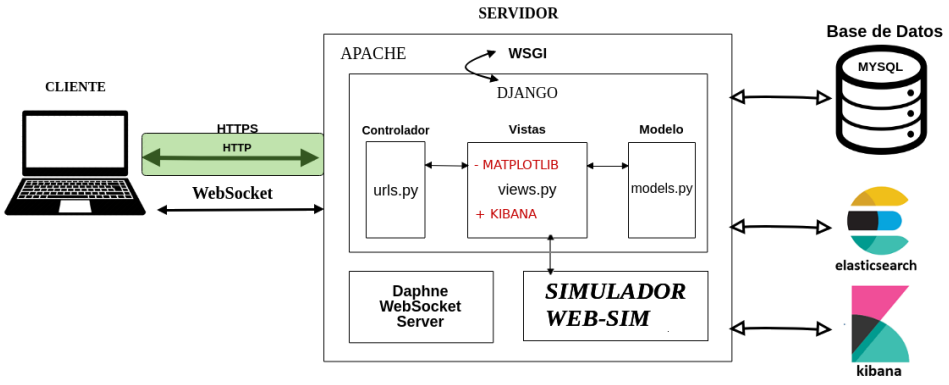

Figura 5.1: Arquitectura Kibotics segundo prototipo.

5.2. Implementacion

(INCOMPLETO) detalles de elasticsearch detalles de kibana

5.3. Validación experimental

(INCOMPLETO) video ilustrativo - pantallazos creacion de base de datos dummy ES creacion de base de datos dummy Kibana

Apéndice A

Ejemplo de apéndice

(INCOMPLETO)

Bibliografía

[1] *Estandar 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] *Quíen usa MongoDB:* https://www.mongodb.com/who-uses-mongodb

[7] *Documentación Elasticseach:* 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] *name:* `url`

[13] *name2:* `url2`

[14] *name3:* `url3`

[15] *name4:* `url4`

[16] *Documentación Matplotlib:* `https://matplotlib.org/`

[17] *Documentación Elasticdump:* `https://www.npmjs.com/package/elasticdump`