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 como tecnologías ya utilizadas. Además, se exponen los pasos realizados para la integración de MongoDB como base de datos, y de Matplotlib como generador de visualizaciones para Kibotics Webserver.

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 es un servicio web para docencia en robótica y programación, desarrollado en Django. El servidor está compuesto por varias partes:

- Kibotics Webserver: Servicio 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.
- Kibotics Websim: Simulador robótico que se ejecuta unicamente en el lado del cliente. Desarrollado en A-Frame, HTML5, JavaScript, entre otras. Permite a los usuarios aprender los fundamentos de la programación robótica y visión artificial.

(INCOMPLETO - arquitectura antigua)

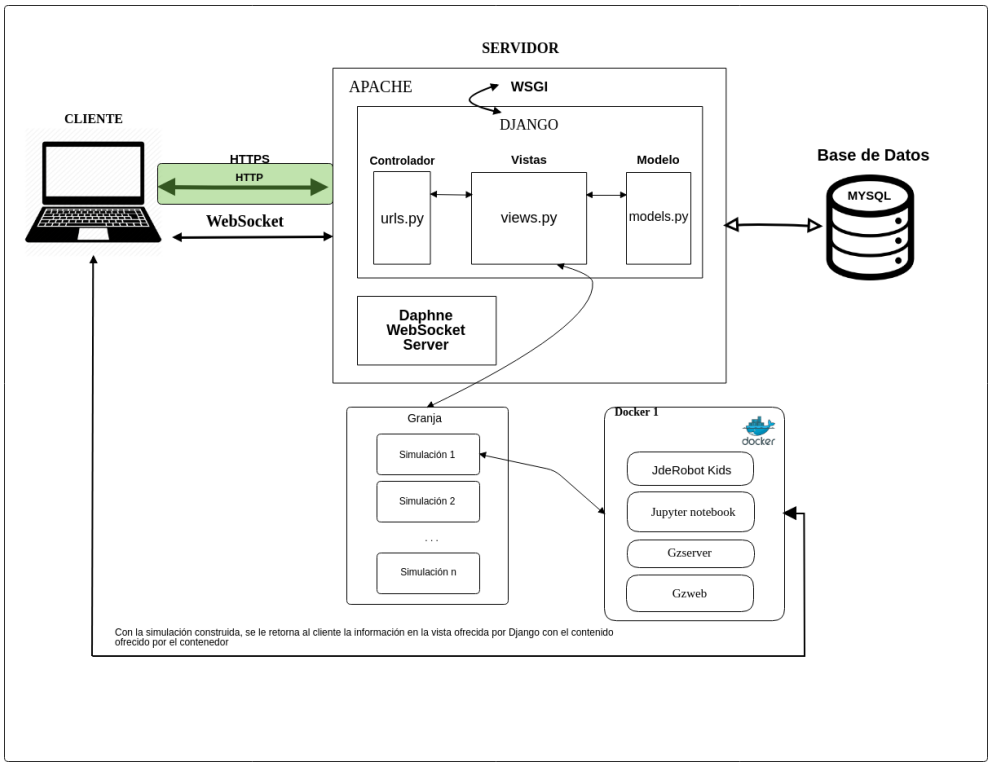

Figura 4.1: Arquitectura Kibotics.

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.

(INCOMPLETO - arquitectura antigua)

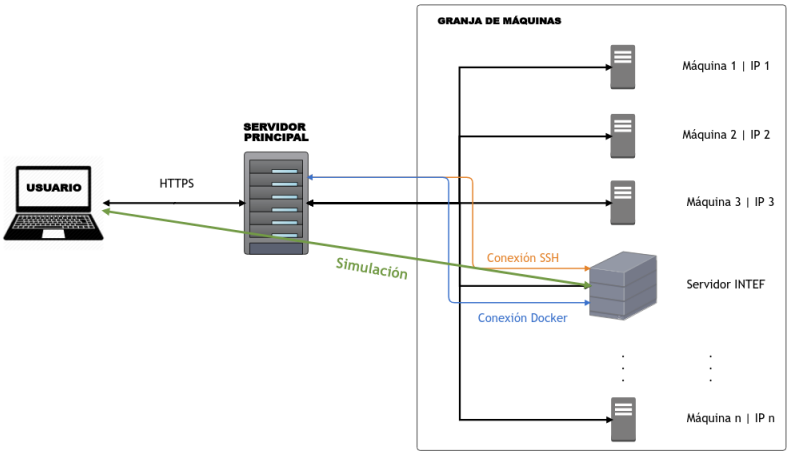

Figura 4.2: Infraestructura Kibotics.

El aspecto clave de Kibotics Webserver 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.

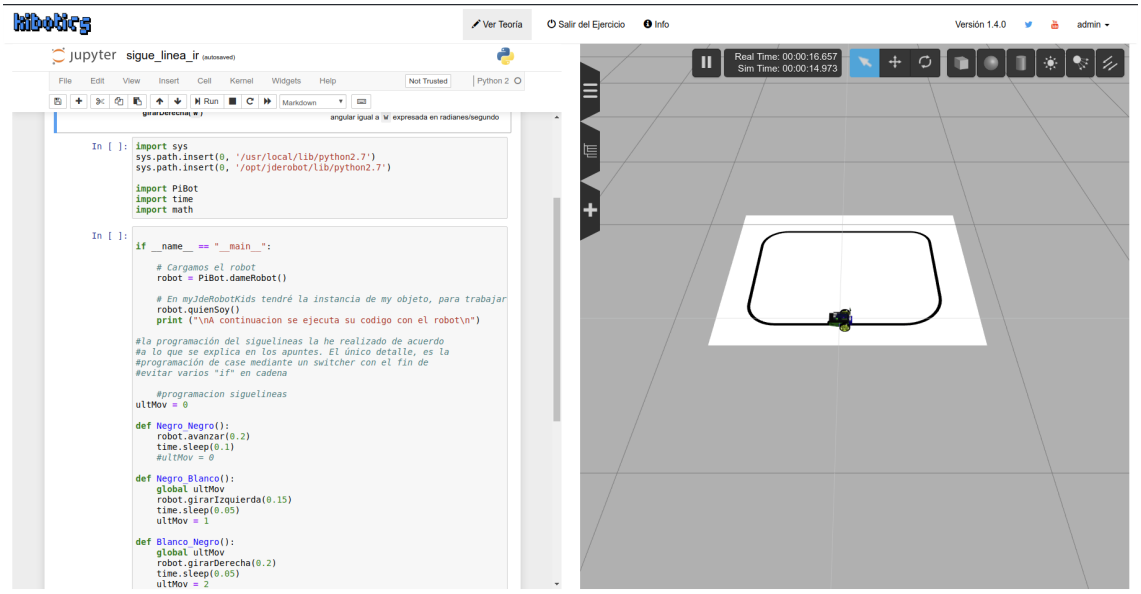

Figura 4.3: Simulador Kibotics.

4.1.2. Logs

Los logs generados por la aplicación Django se guardan en una serie de archivos indexados en el servidor con 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 log, se generaban en el servidor Django haciendo uso de sondas Python, las cuales registran el evento en distintas partes del código del servidor 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.

4.2. Desarrollo local

En está sección se describe la evolución del método de recogida y análisis de los logs. 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 primitiva de trazabilidad de eventos basada en ficheros TXT y con limitados eventos. Con un gran coste computacional derivado de la apertura y cierre constante de ficheros, limitaba 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 procesamiento necesaria.

Es por esto, por lo que en un primer prototipo se decidió 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, iniciar el proceso para levantar el servicio MongoDB se puede realizar con la siguiente sentencia:

```
$ 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 el servicio MongoDB ya instalado lo único necesario para completar la migración de registro de logs, es modificar las sondas previamente definidas. Estas sondas, pasarán de escribir en ficheros a registrar los eventos en MongoDB.

Haciendo uso de la librería pymongo 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, 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 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 logueados, 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.

4.2.2. 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 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):
                    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
```

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:

- Primero, 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.

- Segundo, 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.

- Tercero, 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.

- 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>
...
```

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

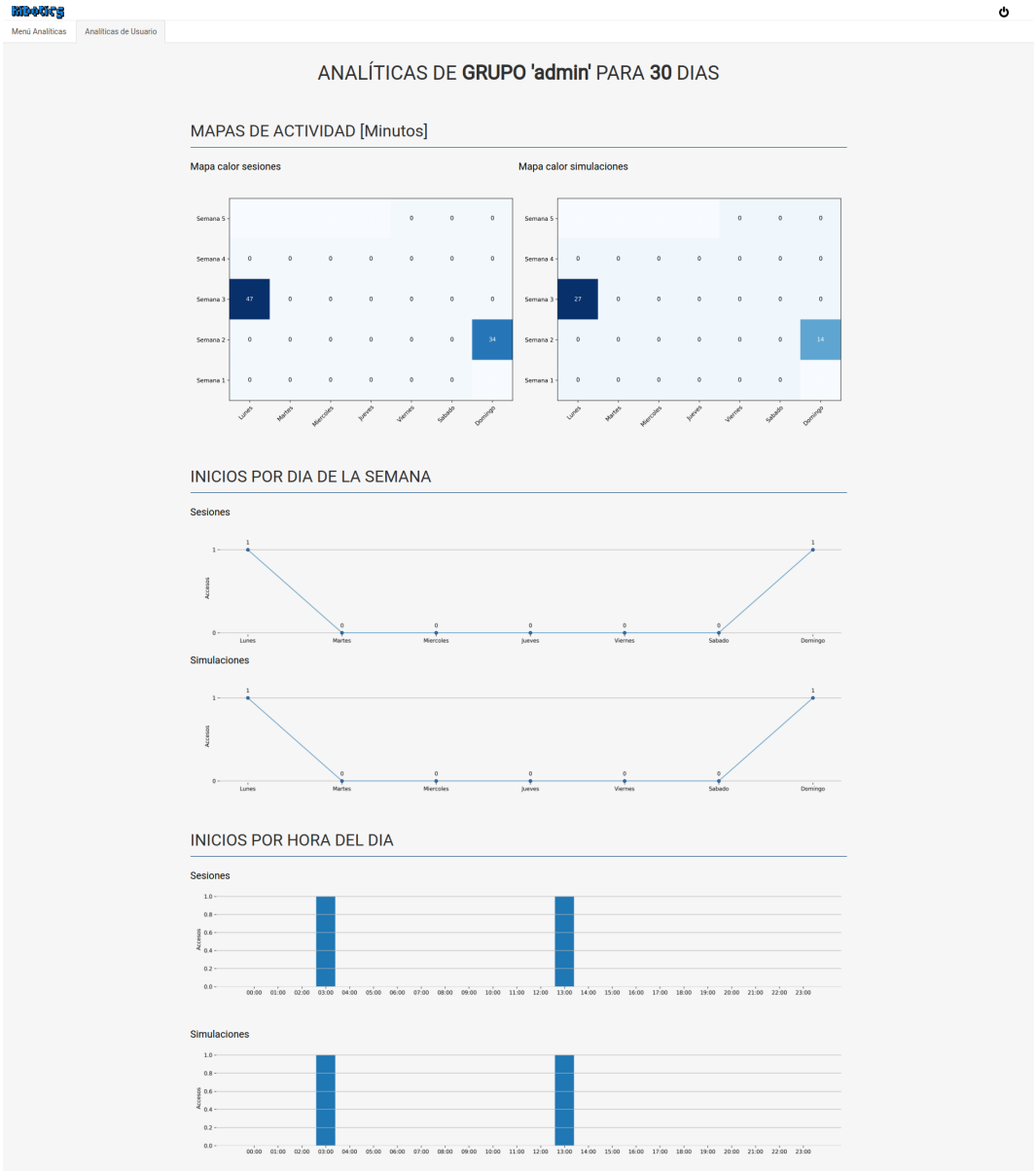

Figura 4.4: Primer prototipo parte 1.

En la primera figura, 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 (celdas).

En una segunda parte, 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 muestran, 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.

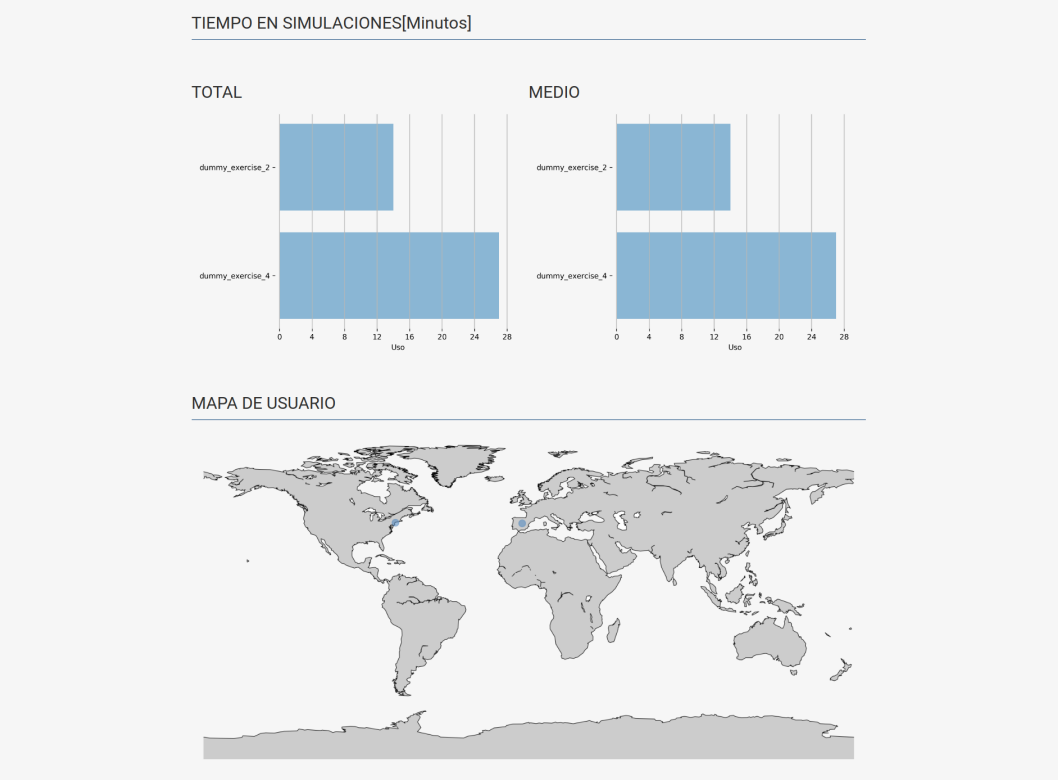

Figura 4.5: Primer prototipo parte 2.

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.

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.