

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.

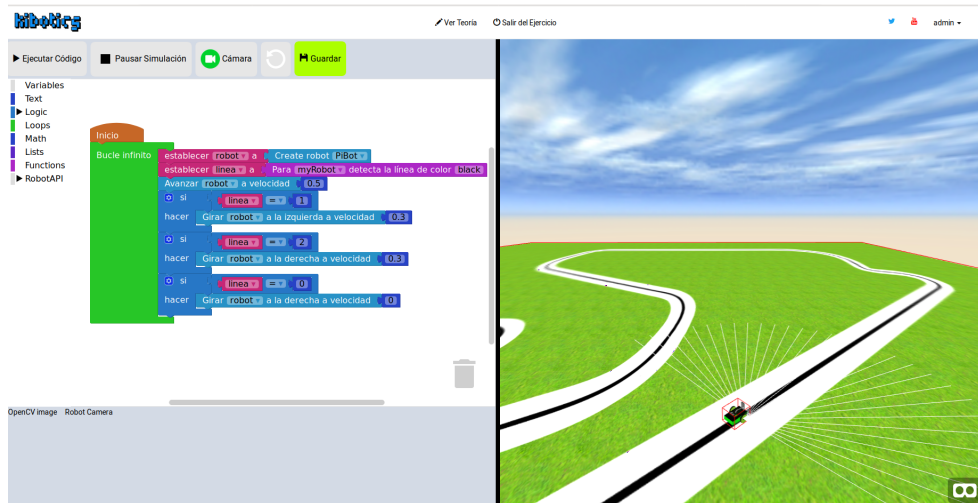


Figura 4.1: Editor y simulador en Kibotics.

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>

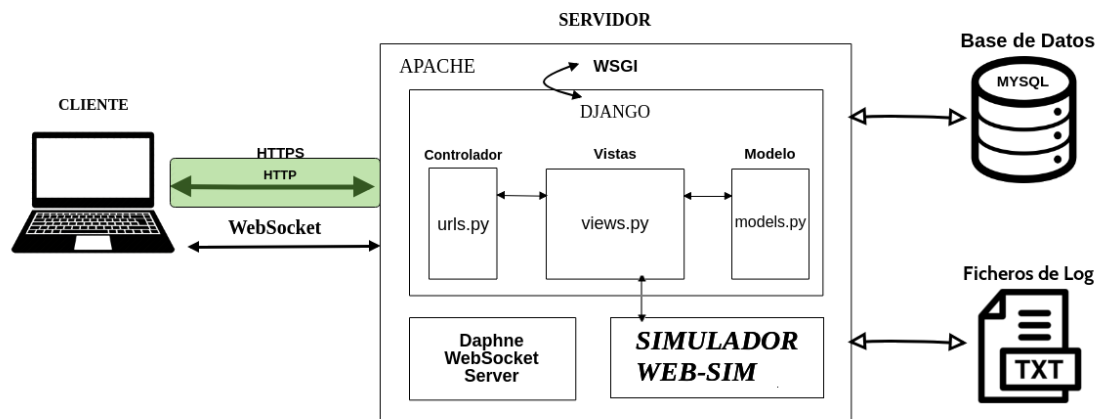


Figura 4.2: Arquitectura Kibotics.

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

Estos eventos logueados se generaban en el servidor Django haciendo uso de sondas 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 MongoDB, 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 `analytics.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.

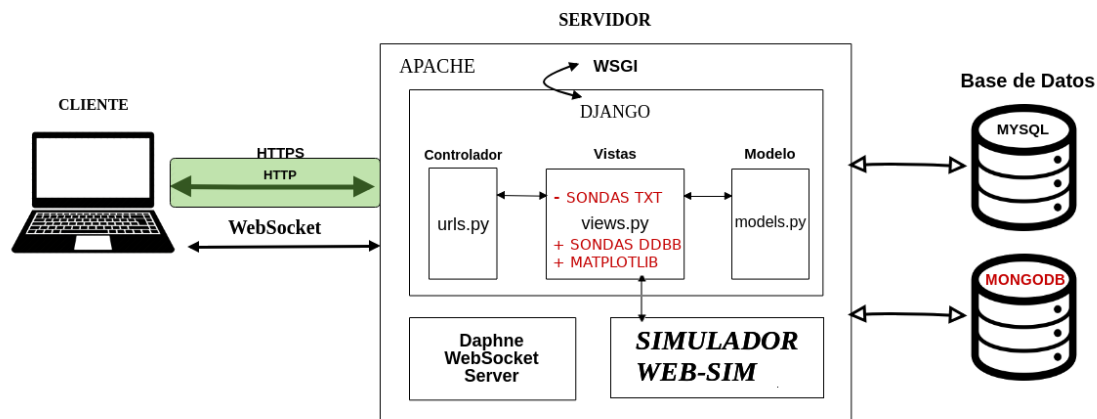


Figura 4.3: Arquitectura Kibotics.

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>
        
    </div>

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

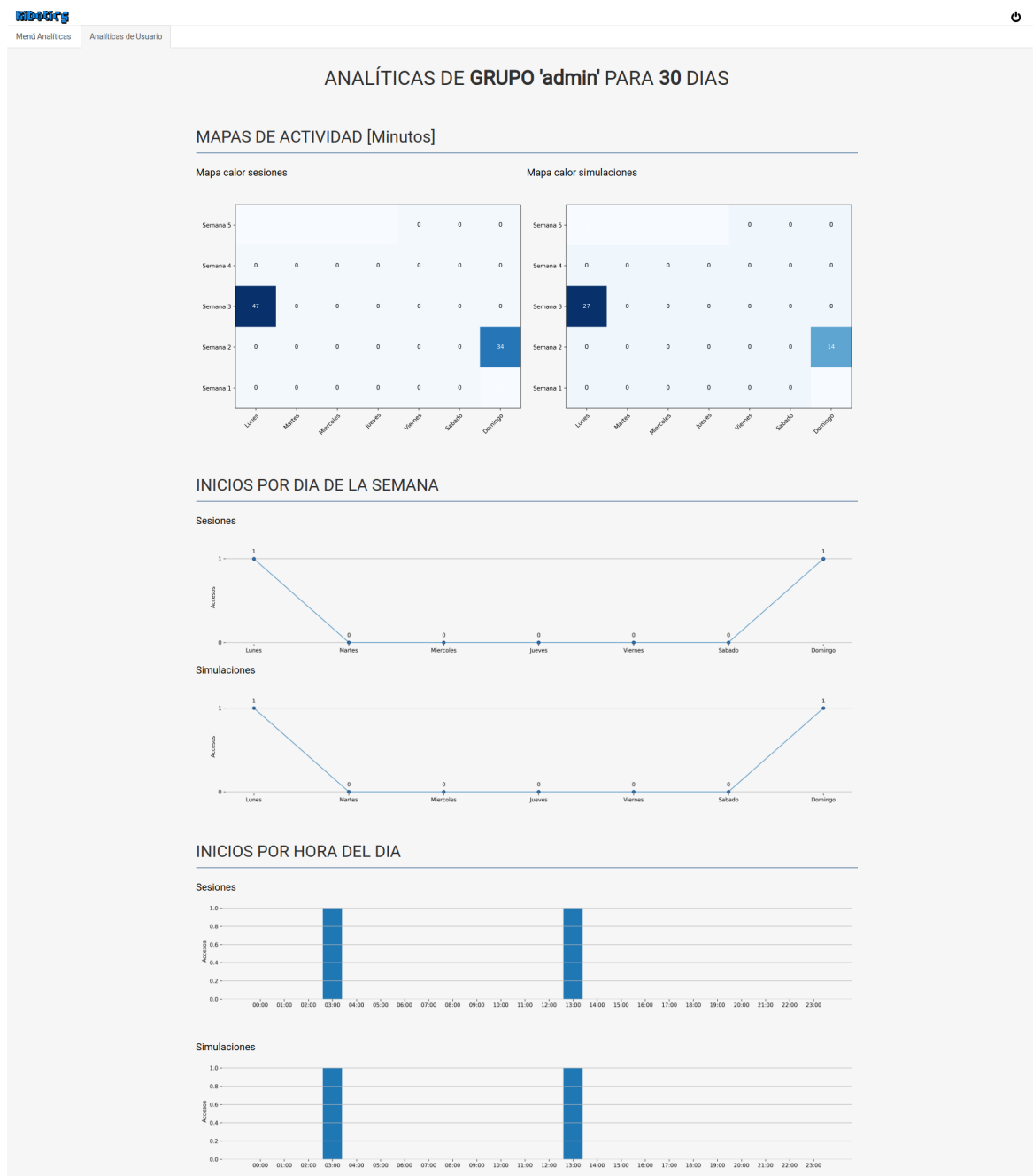


Figura 4.4: Primer prototipo parte 1.

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.

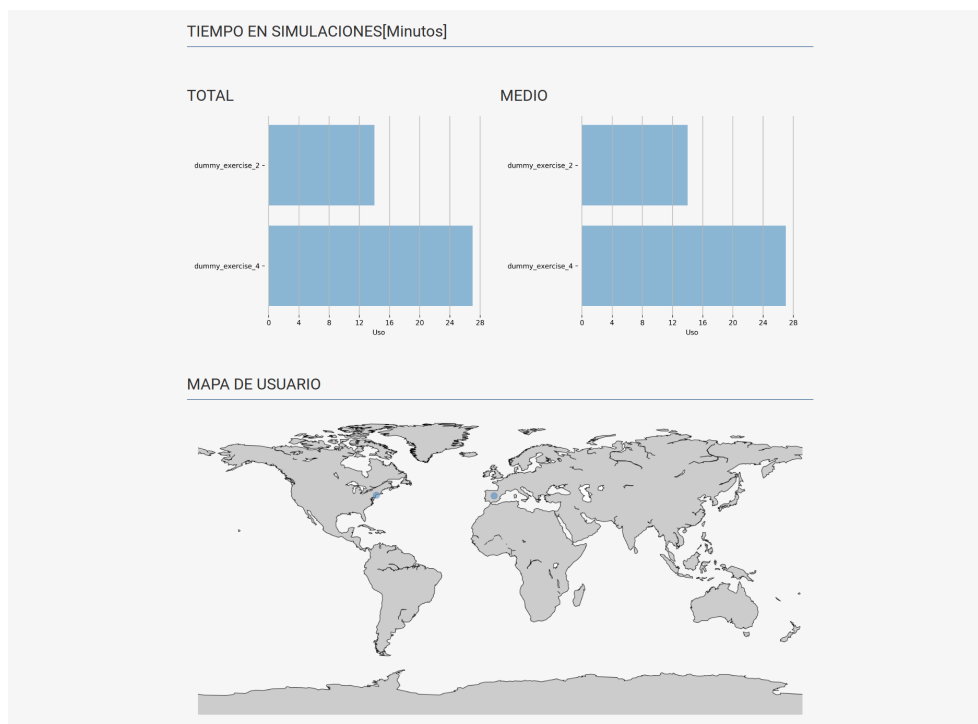


Figura 4.5: Primer prototipo parte 2.

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.