



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES
Y MULTIMEDIA

TRABAJO FIN DE GRADO

**Analíticas automáticas en una plataforma web de
robótica educativa**

Autora: Claudia Álvarez Bravo

Tutor: Dr. José María Cañas Plaza

Co-tutor: Dr. David Roldán Álvarez

Curso Académico 2021/2022

Agradecimientos

Este trabajo va dedicado principalmente a toda mi familia, la cual me ha apoyado durante todo este proceso, en especial a mi hermano César, que siempre me ha animado a continuar con mis sueños y a sobrepasar mis límites.

También agradezco a mis tutores del TFG, José María Cañas y David Roldán Álvarez, por el esfuerzo y el seguimiento de mi TFG.

Y por último agradecer a todos los profesores y compañeros que he tenido durante la carrera los cuales me han enseñado lo maravilloso y apasionante que es la ingeniería.

Resumen

Conocer las interacciones de los usuarios en una plataforma Web se ha convertido en un aspecto fundamental para poder analizar el comportamiento de éstos con idea de mejorar los servicios ofrecidos por la plataforma. La monitorización y posterior análisis de esta información, mejorar la experiencia del usuario e incluso hacer una optimización de la plataforma web. Para realizar estas tareas es necesario recoger el máximo de datos posibles lo cual permite hacer un mejor análisis.

Este Trabajo de Fin de Grado (TFG) muestra el proceso seguido para la *recogida de la información* de los usuarios en la plataforma web educativa Unibotics (realizando una monitorización automático) y cómo se han *visualizado estos datos masivos* para poder analizarlos. Unibotics está dirigido a estudiantes universitarios donde aprenden sobre robótica y a programar en Python a través de la realización de varios ejercicios. Estos datos recogidos permitirán a los administradores conocer mejor el comportamiento de los estudiantes y a la vez, los estudiantes podrán ver sus progresos en cada ejercicio.

Para la realización de este TFG se han combinado diferentes tecnologías que han permitido tanto el almacenamiento de la información como su posterior visualización automática. La base de datos utilizada ha sido Elasticsearch y se ha elegido Dash como entorno para la creación de las visualizaciones automáticas. Elasticsearch forma parte del ELK Stack, que tiene su propio visualizador, Kibana, pero se decidió utilizar Dash debido a su facilidad de uso, configuración y administración.

Índice general

1. Introducción	1
1.1. Tecnologías web	1
1.1.1. Tecnologías web del lado del cliente	3
1.1.2. Tecnologías web del lado del servidor	3
1.2. Plataforma web de programación	4
1.3. Robótica y software de robots	7
1.4. Plataformas de programación de robótica	10
1.4.1. Unibotics	11
1.5. Estructura del documento	14
2. Objetivos y Metodología del Trabajo	15
2.1. Objetivos	16
2.2. Metodología	16
2.3. Plan de Trabajo	17
3. Herramientas utilizadas	19
3.1. HTML	19
3.2. CSS	22
3.3. JavaScript	23
3.4. Django	24
3.5. Lenguaje SQL	26
3.6. Elasticsearch	28
3.7. DASH	30

4. Analíticas automáticas en Unibotics	31
4.1. Diseño	31
4.2. Recogida de sondas	33
4.2.1. Tipos de sondas	33
4.2.2. Base de datos de eventos interesantes	34
4.2.3. Detección de las sondas	35
4.2.4. Procesamiento y grabación de las sondas	37
4.2.5. Despliegue	38
4.3. Visualización de la información	40
4.3.1. Filtros habituales	43
4.3.2. Visualización del número de registros y usuarios de la plataforma	44
4.3.3. Visualización de actividad en la plataforma Unibotics	46
4.3.4. Visualización de los metadatos de los usuarios	52
4.3.5. Visualizaciones de las puntuaciones automáticas	54
4.3.6. Despliegue	56
5. Conclusiones y trabajos futuros	57
5.1. Conclusiones finales	57
5.2. Competencias adquiridas	58
5.3. Trabajos futuros	59

Índice de figuras

1.1. Primera página web	2
1.2. Arquitectura Cliente-Servidor	3
1.3. Plataforma Scratch	5
1.4. Snap!	6
1.5. Robocode	6
1.6. CodeCombat	7
1.7. Esquema ROS	8
1.8. Curso de robótica en TheConstruct	10
1.9. Curso de robótica en Riders.ai	11
1.10. Estructura de Unibotics	12
1.11. Unibotics en la actualidad	13
3.1. Estructura HTML	20
3.2. Estructura de un elemento HTML	21
3.3. Sintaxis CSS	22
3.4. Estructura de Django	25
3.5. Esquema funcionamiento SQL	26
3.6. Sentencia SQL	27
3.7. Arquitectura Elasticsearch	29
4.1. Antigua arquitectura de Unibotics	31
4.2. Arquitectura Unibotics con las nuevas tecnologías	32
4.3. Elasticsearch dummy	40
4.4. Esquema de Dash	41
4.5. Menú de un administrador en Dash	42

4.6. Filtros utilizados en Dash	43
4.7. Gráficas relativas a los usuarios	45
4.8. Gráfica registro de usuarios acumulado	45
4.9. Gráfica de evolución de usuarios activos cada día	46
4.10. Gráfico sesiones totales por día	47
4.11. Gráfico sesiones únicas por día	47
4.12. Gráfico de tiempo de uso en Unibotics	48
4.13. Gráfico de tiempo en Unibotics de un usuario concreto	49
4.14. Histograma de las duraciones de sesiones	49
4.15. Histograma del ejercicio <code>follow_line</code> de un grupo de usuarios	50
4.16. Mapas de calor sesiones	51
4.17. Mapa geográfico de sesiones	52
4.18. Mapa geográfico de sesiones de España	53
4.19. Gráfico de navegadores usados por los estudiantes	53
4.20. Gráfico de sistemas operativos	54
4.21. Gráfica de puntuación de estilo	55
4.22. Gráfica de puntuación de eficacia	55

Capítulo 1

Introducción

La forma de impartir conocimientos ha ido cambiando y evolucionando a lo largo de los años. Actualmente es difícil no encontrar un aula donde las tecnologías web están muy presentes y más en el ambiente universitario. Para poder implementar una plataforma web competitiva en el mercado y además atrayente al usuario es imprescindible no sólo proporcionar la funcionalidad apropiada cumpliendo siempre con los requisitos de usabilidad correspondientes, sino que también es necesario conocer cómo los usuarios utilizan la plataforma. Para ello, es imprescindible recoger la interactividad que el usuario tiene con la plataforma web, guardar esos datos y después visualizarlos.

En este capítulo se va a hablar de los elementos clave en los que se basa este trabajo que son las tecnologías web, el *software* de robots y la robótica educativa.

1.1. Tecnologías web

Las tecnologías de internet se remontan a 1970, cuando el departamento de defensa de Estados Unidos creó una red descentralizada que pudiera aguantar ataques nucleares. En 1990, Tim Berners-Lee trabajador del CERN¹ originó un protocolo de comunicación basado en hipertextos (HTTP) donde científicos compartían documentos, gracias a eso, Berners-Lee junto a su equipo desarrolló el lenguaje de marcado HTML² y el sistema de direcciones de web URL³,

¹Organización Europea para la Investigación Nuclear

²HyperText Markup Language

³Uniform Resource Locator

ese mismo año creó la primera página web (Figura 1.1) dando inicio a la WWW⁴. Al cabo de los años se fueron desarrollando los diferentes navegadores hasta los que se conocen hoy en día.

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#) , [Policy](#) , November's [W3 news](#) , [Frequently Asked Questions](#) .

[What's out there?](#)

Pointers to the world's online information, [subjects](#) , [W3 servers](#), etc.

[Help](#)

on the browser you are using

[Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#) , [X11 Viola](#) , [NeXTStep](#) , [Servers](#) , [Tools](#) , [Mail robot](#) , [Library](#))

[Technical](#)

Details of protocols, formats, program internals etc

[Bibliography](#)

Paper documentation on W3 and references.

[People](#)

A list of some people involved in the project.

[History](#)

A summary of the history of the project.

[How can I help?](#)

If you would like to support the web..

[Getting code](#)

Getting the code by [anonymous FTP](#) , etc.

Figura 1.1: Primera página web

En la actualidad las principales ventajas de las tecnologías web son que se pueden utilizar en cualquier dispositivo, el usuario no necesita instalar nada más allá de un navegador web y su usabilidad es sencilla, si bien es necesaria una conexión a Internet para poder acceder a ellas [1].

Las páginas Web modernas utilizan el modelo Cliente-Servidor, en el que el cliente hace peticiones al servidor esperando una respuesta de éste. Un mismo cliente puede estar conectado con varios servidores y a la vez interactúa con el usuario final normalmente a través de una interfaz gráfica. La parte del servidor se encarga de recibir las peticiones, analizarlas y procesarlas para enviar una respuesta. En las siguientes subsecciones, describimos algunas de las tecnologías más populares utilizadas en el desarrollo de páginas Web.

⁴World Wide Web

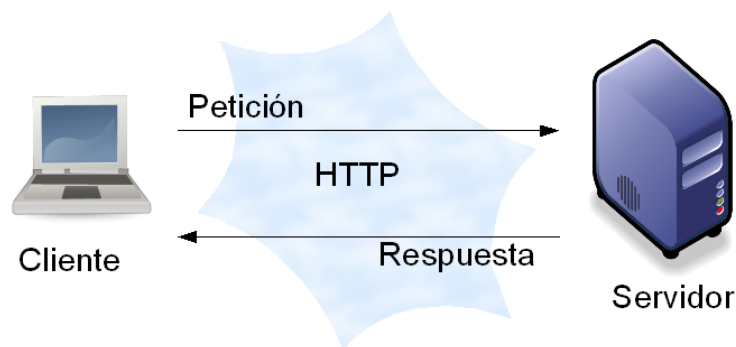


Figura 1.2: Arquitectura Cliente-Servidor

1.1.1. Tecnologías web del lado del cliente

Las tecnologías del lado del cliente suelen recibir el nombre de *frontend*, son las que interactúan con el usuario directamente, las más importantes son:

- **HTML**: Lenguaje de marcado que se encarga de dar la estructura a la página web. HTML utiliza etiquetas para mostrar los diferentes contenidos. La última versión es HTML5 donde se introducen las etiquetas de audio y vídeo.
- **CSS⁵**: Lenguaje de estilo que se encarga del diseño gráfico, de la apariencia de las páginas web. La última versión es CSS3.
- **JavaScript**: Lenguaje de programación interpretado que permite la ejecución de código orientado a eventos (por ejemplo, pulsar un botón). Puede actuar sobre el navegador a través de objetos integrados, es decir puede manipular el HTML cargado.

1.1.2. Tecnologías web del lado del servidor

Las tecnologías web del lado del servidor, también llamadas *backend*, son las encargadas de procesar las peticiones del cliente y enviar una respuesta en forma de páginas HTML. Normalmente se utilizan *frameworks* que son herramientas que te permiten desarrollar una aplicación web de forma ágil y sencilla a partir de librerías o funciones ya creadas. Algunos de estos *frameworks* son:

⁵Cascading Style Sheets

- **Django:** Basado en Python. Django hace uso de plantillas permitiendo hacer páginas web más rápidas y sencillas. Gracias a su ORM⁶ no es necesario saberse el manejo de base de datos ya que este traduce directamente el código de Python al lenguaje estándar de las peticiones a bases de datos, SQL.
- **Node.js:** Permite ejecutar código JavaScript fuera del navegador y está construido con el motor de JavaScript V8 de Chrome. Node.js utiliza un modelo de entrada y salida no bloqueante, no espera a que los procesos finalicen.
- **Symfony:** Utiliza de lenguaje de programación PHP⁷. Está basado en el patrón Modelo Vista Controlador (MVC). Symfony es muy flexible permitiéndote instalar únicamente lo que se necesite en vez del *framework* completo.
- **Spring Boot:** Desarrollado para aplicaciones programadas en Java, surgió debido a que el Spring Framework era difícil de configurar.
- **Laravel:** *Framework* de PHP. En comparación con otros *frameworks* de PHP su curva de aprendizaje es más baja. Es flexible y adaptable, no solamente debido al MVC, sino que también propone utilizar *routes with closures*, a causa de lo cual reduce código [2].

1.2. Plataforma web de programación

En la actualidad, gracias a Internet se tiene al alcance plataformas web donde se enseña a programar de una forma interactiva. Una de las ventajas de aprender programación en una plataforma web es que puedes hacerlo desde la comodidad de tu casa y una gran parte son gratuitas. Pueden ser utilizadas por usuarios de todos los niveles. A continuación, se nombra algunos ejemplos de estas plataformas:

⁶Object Relational Mapping

⁷Hypertext Preprocessor

- **Scratch**⁸: Es un entorno de programación desarrollado por el MIT⁹. Se puede descargar en local o se puede utilizar la plataforma de forma *online*. Utiliza un lenguaje de programación visual llamado igual que la plataforma, Scratch, el cual está basado en bloques permitiendo crear programas arrastrando y soltando para agrupar dichos bloques como un puzle. Cada bloque puede significar un evento, movimiento, un operador o una variable, entre otras. Con esta plataforma, por ejemplo, se pueden crear juegos, historias interactivas, música de manera sencilla y dinámica para el usuario. Las edades que tienen los usuarios que utilizan la plataforma está entre los 8 y 16 años, aunque hay adultos que también la utilizan [3].

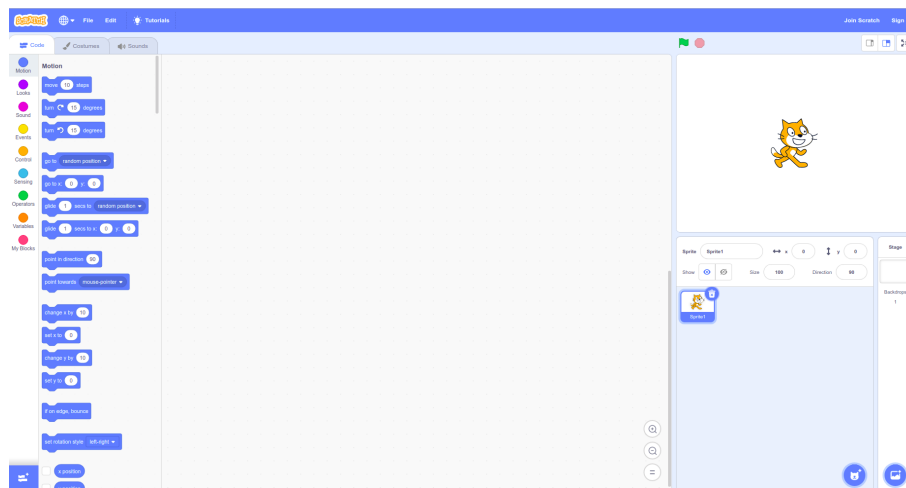


Figura 1.3: Plataforma Scratch

- **Snap!**¹⁰: Está basada en el código de programación de Scratch, creada en la Universidad de Berkeley. Esta aplicación la puedes utilizar tanto *online* como descargándola y pudiendo utilizarla sin necesidad de Internet. La principal diferencia con Scratch es que te permite crear tus propios bloques utilizando JavaScript y así poder formar tu librería propia, también se puede crear listas avanzadas donde se puede almacenar cualquier tipo de dato, incluso otras listas. Esta aplicación web permite aprender a programar de una manera más visual y evitando los errores de sintaxis que se podrían cometer. Snap! tiene una gran comunidad de usuarios que suben sus proyectos, donde se puede aprender de ellos [4].

⁸<https://scratch.mit.edu/>

⁹Massachusetts Institute of Technology

¹⁰<https://snap.berkeley.edu/>

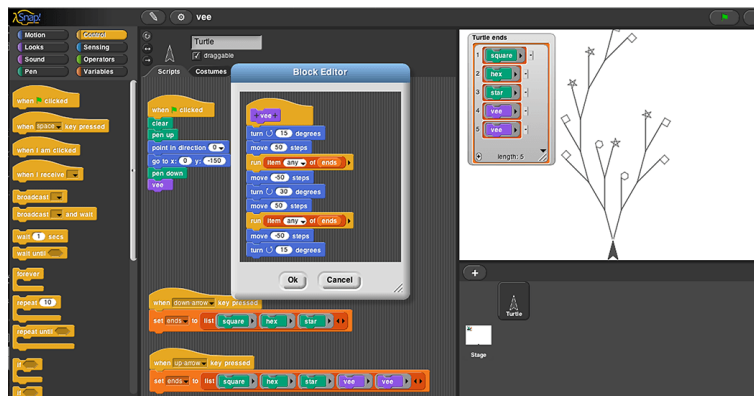


Figura 1.4: Snap!

- **Robocode**¹¹: Es un videojuego multiplataforma que consiste en la creación de un tanque robot el cual tendrá que atacar y esquivar otros tanques para no ser destruido. No es necesario descargar ningún *software* adicional aparte del propio Robocode. Está dirigido para aprender Java y .NET. Nos podemos encontrar ligas donde las batallas de los robots transcurren en tiempo real, éstas están divididas en diferentes categorías dependiendo del tamaño de código efectivo en bytes, para que sean competiciones más justas. Se puede programar las tres partes del tanque que son: el cuerpo que se encarga de mover el tanque, el radar que detecta a los adversarios y el cañón que se utiliza para apuntar y disparar. La aplicación dispone de un editor de texto, un *debugger* y un compilador, todos los robots que se vayan desarrollando se podrán guardar para futuras batallas [4].

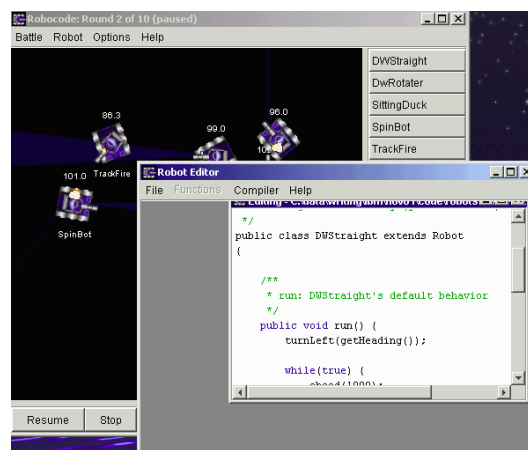


Figura 1.5: Robocode

¹¹<https://robocode.sourceforge.io/>

- **CodeCombat**¹²: Es una página web donde a través de un personaje se enseña a programar en diferentes lenguajes como Python, JavaScript, C++, entre otros. El objetivo del juego es ir superando los diferentes niveles, los cuales van aumentando de dificultad. Está compuesto por 110 niveles gratuitos con opción de jugar más niveles si se paga, además pagando se pueden desbloquear diferentes héroes, aprender a programar juegos y páginas web [4].

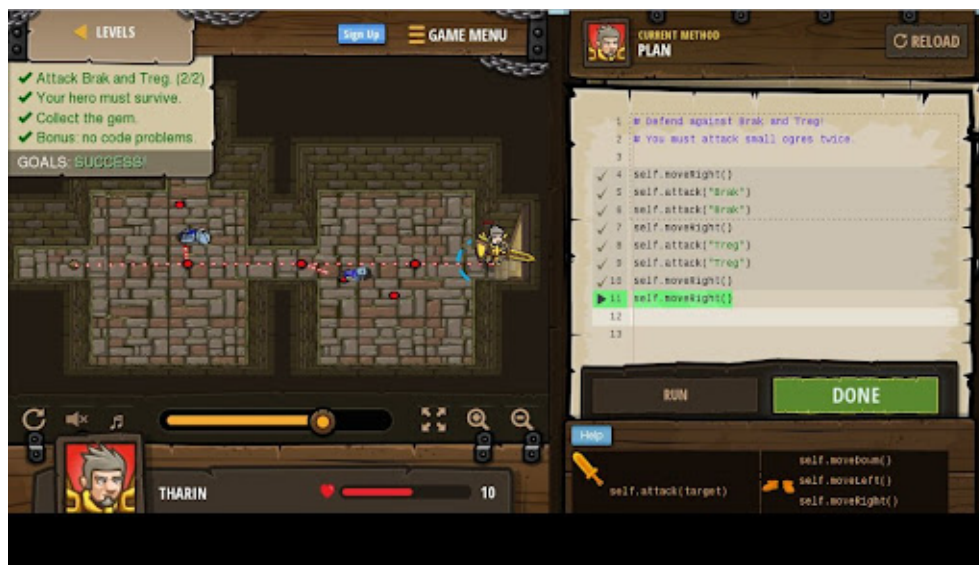


Figura 1.6: CodeCombat

1.3. Robótica y software de robots

La robótica es la ciencia o rama tecnológica encargada del estudio, diseño, creación y aplicación de los robots. En la actualidad existen muchas formas de estudiar robótica, puedes estudiarla tanto en universidades como en cursos. Existen diferentes herramientas para trabajar con robots, como los entornos y las plataformas robóticas. Se encuentran varios ejemplos de entornos como RoboComp¹³ o Player¹⁴, pero el más conocido y usado es ROS¹⁵ (*Robot Operating System*).

¹²<https://codecombat.com/>

¹³<https://robocomp.github.io/web/>

¹⁴<http://playerstage.sourceforge.net/>

¹⁵<https://www.ros.org/>

ROS es un *framework* flexible para escribir *software* de robots. Las principales ventajas son que es de código abierto, contando con una comunidad global para mejorar el *software*. Además, aunque empezó en Linux, hoy es una herramienta multiplataforma e independiente de las interfaces de usuario [5].

El funcionamiento de ROS está basado en nodos, éstos son procesos ejecutables que realizan una tarea simple. Las tareas de los nodos pueden ser controlar la velocidad, el motor de las ruedas o la localización del robot. Los nodos pueden enviar mensajes o recibir mensajes de otros nodos a través de los *tópicos*.

Los tópicos se encargan de transmitir el mensaje a todos los nodos que se han suscrito y reciben los mensajes que publican otros nodos. Los nodos no saben qué nodo ha publicado ni qué nodo se ha suscrito a un tópico. Los nodos también pueden ofrecer servicios que serán acciones que otro nodo puede pedir. El servidor que ofrece el *servicio* solo manda la respuesta cuando recibe una solicitud de otro nodo, haciendo la comunicación síncrona. En la Figura 1.7 se muestra un esquema del funcionamiento de ROS [6].

Otro nodo importante es el nodo maestro que se encarga de registrar todas las suscripciones y publicaciones a los tópicos.

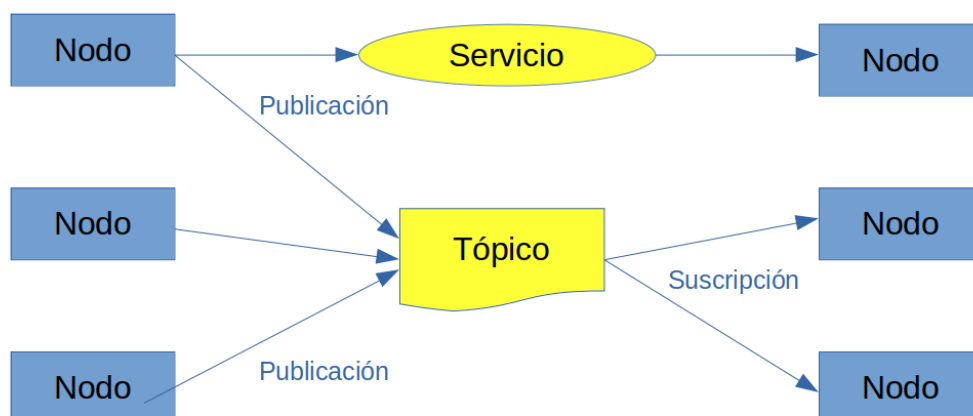


Figura 1.7: Esquema ROS

Para programar robots, otra herramienta potente son los simuladores de robots. Permiten emular a robots reales de forma virtual para poder trabajar con ellos. La simulación es importante para saber cómo se comportaría un robot en la realidad y saber si su *software* necesita cambios. Además, una ventaja es que económicamente la construcción de un robot puede ser costosa, con un simulador se evitan estos gastos.

Existen varios simuladores, como CoppeliaSim¹⁶, antiguamente llamado V-REP, que cuenta con un entorno de desarrollo integrado. Cada objeto o modelo se puede controlar vía ROS, un *plugin*, un *script* integrado, un cliente API remoto o una solución personalizada [7]. Webots¹⁷ es otro simulador de código abierto y multiplataforma, ofrece bibliotecas con ejemplos de mundos, sensores o robots, entre otras características [8].

Entre todos los simuladores cabe destacar Gazebo¹⁸. Para crear su simulación de las físicas puede utilizar cuatro motores diferentes: ODE¹⁹ (simulación de cuerpos rígidos, que permite detectar y simular colisiones), Bullet (simulación de cuerpos rígidos y blandos, también detecta colisiones), DART²⁰ (simulación de la dinámica y cinemática de un robot) y Simbody (herramienta útil para simular articulaciones con restricciones y resuelve la segunda ley de Newton) [9].

Gazebo representa gráficos 3D avanzados y realistas utilizando OGRE²¹, que facilita la *renderización*. Dispone de diferentes sensores con la opción de añadir ruido. Además, se pueden desarrollar complementos para los robots, los sensores o el escenario. Gazebo ofrece varios robots o la posibilidad de crear uno propio [10].

¹⁶<https://www.coppeliarobotics.com/>

¹⁷<https://cyberbotics.com/>

¹⁸<http://gazebosim.org/>

¹⁹*open dynamics engine*

²⁰*Dynamic Animation and Robotics Toolkit*

²¹*Object-Oriented Graphics Rendering Engine*

1.4. Plataformas de programación de robótica

Una parte de la robótica es la programación de estos robots, por lo que se han ido creando herramientas de aprendizaje sencillas y atrayentes para el usuario. En Internet se pueden encontrar diversas plataformas web las cuales te enseñan sobre programación y robótica. Entre ellas se encuentran:

- **TheConstruct**²²: Plataforma web que enseña sobre robótica, ROS e inteligencia artificial. Tiene una versión gratuita en la que se ofrecen tres cursos: Linux para robótica, Python3 para robótica y C++ para robótica, si se quiere puedes acceder a todos los cursos con su versión de pago. Está desarrollado para que puedan utilizarlo tanto principiantes como profesionales. No requiere de la instalación de ROS. Además, una de sus ventajas es que tiene una gran comunidad donde se puede establecer contactos y aprender nuevas formas de programar robots. TheConstruct cuenta con robots reales que se pueden alquilar por un determinado tiempo, dando la posibilidad de poder conectarse a ellos y programarlos desde cualquier lugar. Una vez se selecciona un curso, se tiene en la parte izquierda la teoría para realizar el curso. También tiene un interfaz de usuario dónde se escribe el código, un terminal para escribir comandos y una simulación del robot que se va a programar.

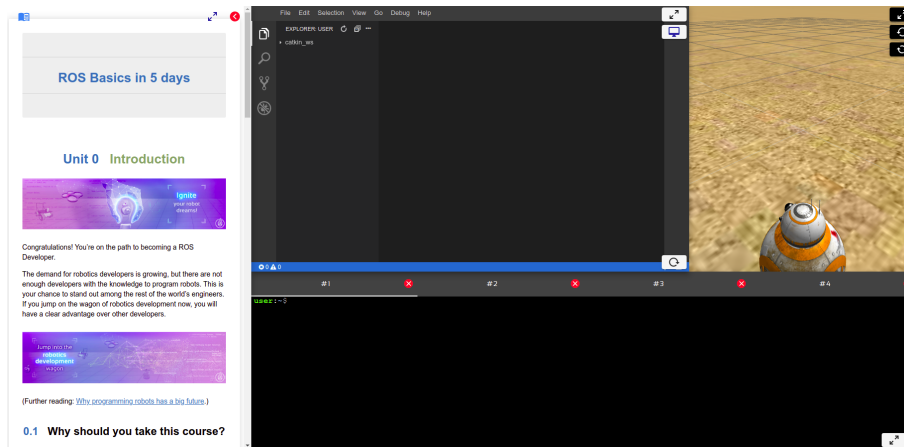


Figura 1.8: Curso de robótica en TheConstruct

²²<https://www.theconstructsim.com/>

- **Riders.ai**²³: Plataforma robótica de simulación, educación y competiciones basada en la nube, desarrollada por Acrome Robotic Systems [11]. Es una plataforma de pago, solamente es gratis la primera lección del curso Introducción a la robótica: parte 1. Consta de otros dos cursos, los cuales son la continuación del curso mencionado anteriormente. Además, una vez finalizados los cursos se obtiene un certificado. Se enseña a programar en Python o C++ los robots. Las lecciones están formadas por la teoría, el interfaz de usuario y el simulador Gazebo, todo ello en la misma pestaña, como se muestra en la Figura 1.9. Riders dispone de dos ligas para competir con otros programadores. Una liga consiste en programar drones voladores y otra en programar robots móviles.

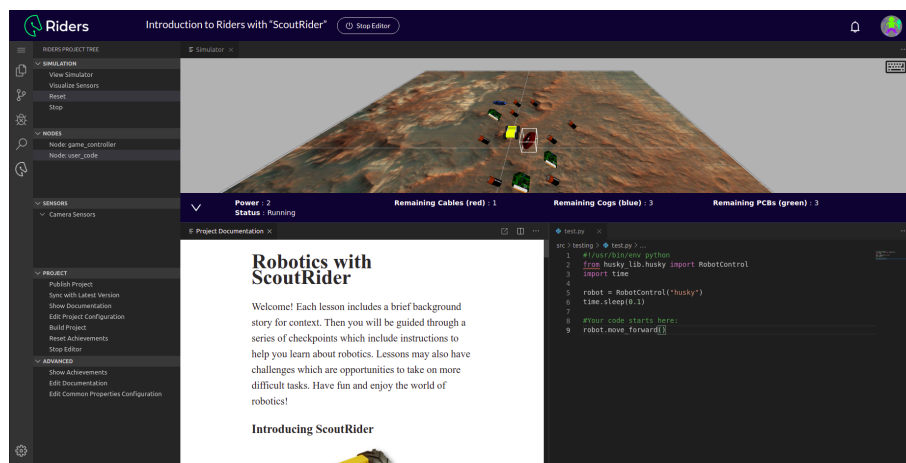


Figura 1.9: Curso de robótica en Riders.ai

1.4.1. Unibotics

Unibotics²⁴ es una plataforma en línea de robótica educativa, en la cual se enseña a programar de una manera llamativa y divertida para estudiantes universitarios. Es justo en la que se desarrolla este Trabajo de Fin de Grado. Unibotics proviene de otra plataforma que no está en línea, llamada Robotics Academy²⁵. Ambas plataformas han sido creadas por la Asociación de robótica e inteligencia artificial JdeRobot.

²³<https://riders.ai/>

²⁴<https://unibotics.org/>

²⁵<http://jderobot.github.io/RoboticsAcademy/>

1.4. PLATAFORMAS DE PROGRAMACIÓN DE ROBÓTICA

La plataforma consta de varios ejercicios que se pueden dividir en ejercicios de *conducción autónoma* (Follow Line, Obstacle avoidance, Global Navigation, Car Junction y Autoparking), *robots de servicios* (Vacuum Cleaner, Localized Vacuum Cleaner y Laser Mapping), *drones* (Drone Cat and Mouse y Rescue People) y *visión artificial* (3D Reconstruction, Color Filter, Optical Flow Teleop y Montecarlo Visual Loc).

Se pueden programar los ejercicios directamente desde la web sin la necesidad de instalar *software* adicional. En cada ejercicio, el usuario puede introducir código fuente en el navegador, cargarlo en el cerebro de un robot simulado, ejecutarlo en simulación y visualizar el interfaz gráfico durante su ejecución. Además, se puede subir o guardar el código realizado y comprobar la eficacia y el estilo del código. Se utiliza el lenguaje Python para realizar los ejercicios. Unibotics utiliza una imagen de Docker llamada RADI ²⁶(*Robots Academy Docker Image*) donde están preinstaladas todas las dependencias y así no se tiene que descargar nada localmente. Está basada en ROS y Gazebo [12].

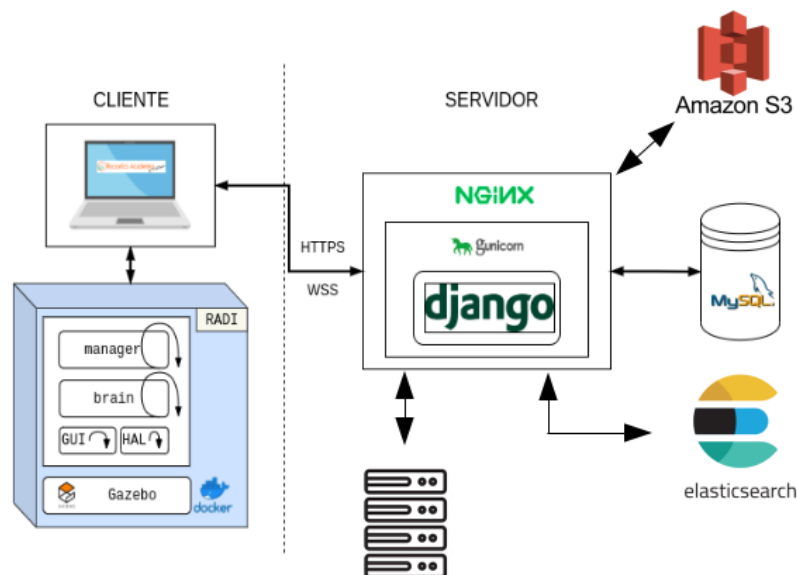


Figura 1.10: Estructura de Unibotics

²⁶<https://hub.docker.com/r/jderobot/robotics-academy/>

1.4. PLATAFORMAS DE PROGRAMACIÓN DE ROBÓTICA

En la Figura 1.10 se muestra la estructura de la plataforma actualmente. En la parte del servidor se encuentra un *webserver*, el cual se conecta a la base de datos, como Elasticsearch y MySQL. También, se conecta al almacenamiento de la nube de Amazon S3 y a una granja de 80 puestos. En la parte del cliente, aparte de la plataforma web se conecta a través de tres websockets al contenedor Docker, RADI. Esto hace que cuando se tiene muchos usuarios, el coste computacional del simulador esté distribuido.

En los últimos 5 años se ha recibido ayuda de *Google Summer of Code*²⁷, gracias a la cual se han ido añadiendo contenidos a la plataforma y también gracias a diferentes Trabajos de Fin de Grado.

En la actualidad esta plataforma es utilizada en las asignaturas de Robótica móvil y Robótica de servicios.^{en} el grado de Ingeniería de Robótica Software de la universidad Rey Juan Carlos y en el máster de visión artificial de la misma universidad.

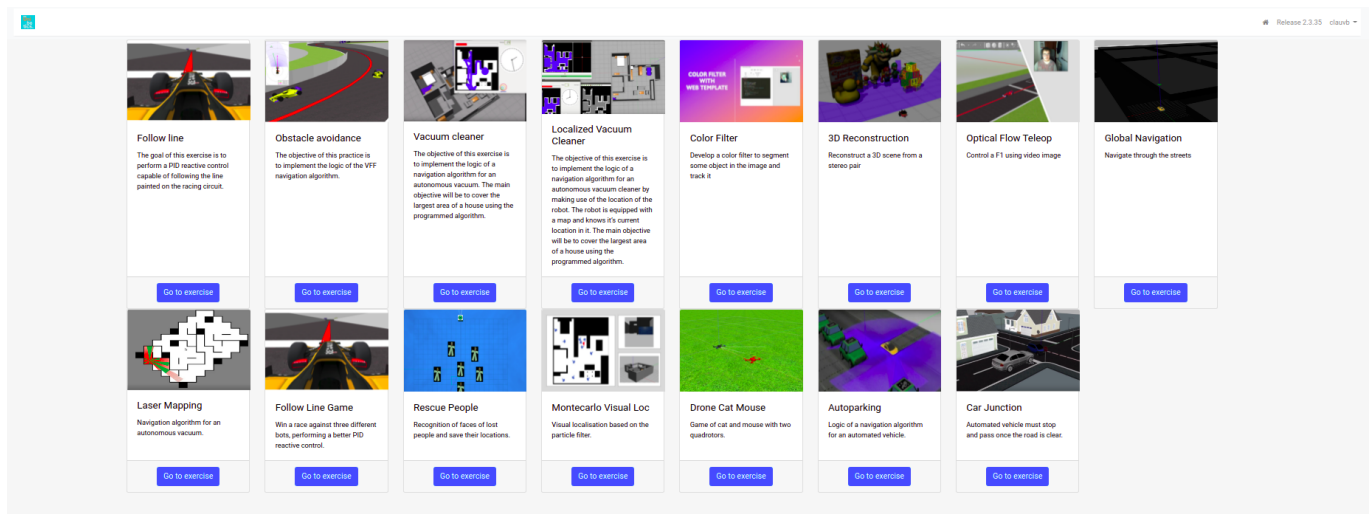


Figura 1.11: Unibotics en la actualidad

²⁷<https://summerofcode.withgoogle.com/archive/>

1.5. Estructura del documento

Este Trabajo de Fin de Grado consta de los siguientes capítulos:

- *Capítulo 1 Introducción:* introducción a las tecnologías web, a páginas web educativas sobre robótica y Unibotics antes de la realización de este TFG.
- *Capítulo 2 Objetivos y Metodología:* Descripción de los diferentes objetivos a cumplir en el TFG y la metodología seguida para la realización de estos.
- *Capítulo 3 Herramientas utilizadas:* se describen las diferentes tecnologías web utilizadas en este trabajo y las herramientas usadas para la recogida y grabación de datos y la visualización de estadísticas automáticas.
- *Capítulo 4 Analíticas automáticas en Unibotics:* en este capítulo se expone el diseño y la implementación de la recogida de sondas y su posterior visualización en la plataforma de Unibotics.
- *Capítulo 5 Conclusiones:* se recopilan las principales conclusiones y los resultados obtenidos en este TFG, además de las competencias que se ha adquirido y futuros trabajos que se podrían realizar a partir de este.

Capítulo 2

Objetivos y Metodología del Trabajo

En los últimos años el uso de plataformas web educativas ha ido incrementándose. La pandemia ha sido otro factor por el cual algunas clases anteriormente presenciales ahora son *online*. Por consiguiente el uso de estas plataformas es más demandado. En el área robótica esto no es una excepción, y se está haciendo especial hincapié en el desarrollo de plataformas *online* que permitan programar robots.

Para que la plataforma pueda mejorarse y ser más atrayente a los alumnos es necesario monitorizar su uso y un posterior análisis, donde los administradores podrán ver cómo los usuarios interactúan con la plataforma y tomar decisiones a partir de los resultados. Los administradores, por lo tanto, podrán ver por ejemplo cuánto tiempo los alumnos tardan en resolver un ejercicio y qué nota obtienen, pudiendo hacer correlaciones entre los datos obtenidos.

Para conseguir que Unibotics cuente con las mejoras mencionadas, se han establecido los siguientes objetivos, metodología y plan de trabajo.

2.1. Objetivos

El objetivo principal que sigue este proyecto es la integración de un sistema de monitorización automática en una aplicación web, en concreto en la plataforma web de robótica educativa Unibotics. Para cumplir el objetivo principal se ha marcado diferentes subjetivos:

1. Capturar las diferentes interacciones de los usuarios, como puede ser el tiempo que un usuario está en la plataforma Unibotics. Esos datos recogidos son llamados sondas. Estas sondas serán capturadas en la parte cliente de Unibotics, utilizando JavaScript.
2. La grabación de las sondas en una base de datos.
3. Creación de gráficas dinámicas automáticas desde de las sondas recogidas. Se mostrarán en varias páginas web dentro de la propia plataforma. Por un lado, en una página de acceso exclusivo a los administradores. Por otro lado, los usuarios podrán acceder a otra página web donde ver un histórico de sus correspondientes puntuaciones en cada ejercicio.

2.2. Metodología

La metodología seguida en este proyecto ha comenzado con la planificación de reuniones semanales con los tutores. En estas reuniones semanales se hace un repaso de lo avanzado durante toda la semana y se fijan los nuevos objetivos para la semana siguiente. Estas reuniones permitían resolver las dudas más inmediatas. Gracias a la plataforma Slack¹ se ha podido mantener la comunicación a lo largo de la semana y poder resolver dudas de una manera más dinámica. En el grupo creado en esta plataforma también se encontraban los administradores y otros desarrolladores de Unibotics.

Para hacer el seguimiento del trabajo se ha creado un blog², donde periódicamente se han ido escribiendo los avances realizados. El blog ha sido creado con *Github-Pages*³

¹<https://slack.com/>

²<https://roboticslaburjc.github.io/2021-tfg-claudia-alvarez/>

³<https://pages.github.com/>

Para comenzar el TFG se ha realizado un primer estudio del código fuente de Unibotics para entender su funcionamiento y poder comprender dónde habría que añadir el código para la recogida de las sondas. Cuando se ha fijado un objetivo semanal en el que hubiera que añadir código nuevo, lo primero se creaba una incidencia o *issue* en el repositorio de Github correspondiente de Unibotics describiendo el problema, después se creaba una nueva rama o *branch* en la que se trabajaba en el código que solucionaba la incidencia para no modificar directamente el código fuente original de Unibotics. Una vez conseguido solucionar la incidencia se crea un parche o *pull request* con la solución donde los administradores de la plataforma lo revisan y si está todo bien fusionan la rama creada con la original.

Unibotics cuenta con tres despliegues: D1 se despliega en local para los desarrolladores, D2 en test para probar la plataforma antes de producción y D3 en producción, en la nube de Amazon. En este proyecto se ha trabajado principalmente en el despliegue D1.

Se ha seguido el modelo de desarrollo software en espiral basado en iteraciones. En cada iteración primero se establece un objetivo, la siguiente etapa es el diseño, luego la implementación y por último, la realización de pruebas.

2.3. Plan de Trabajo

El plan de trabajo que se ha diseñado consta de varias etapas:

1. **Estudio de Unibotics:** Estudio del código fuente de Unibotics.
2. **Estudio de los componentes de terceros y herramientas utilizadas:** Estudio de los diferentes componentes utilizados empezando por Django. Después se estudió la base de datos Elasticsearch y el visualizador Dash y por último la herramienta de contenedores *docker*.

3. **Recogida y captura de sondas:** Con Elasticsearch se graban-almacenan las diferentes sondas para la monitorización de la plataforma. Además, crear una base de datos de prueba para poder trabajar en local. Incorporar las sondas que permiten recoger las principales interacciones de los usuarios con la plataforma web. Primero con un prototipo y luego sobre Unibotics.
4. **Visualización automática de las sondas:** Una vez definido qué sondas se desean capturar y añadido el código para su recogida, se crea la aplicación web para visualizarlas y poder hacer el análisis masivo.
5. **Redacción de la memoria:** Por último, se ha escrito esta memoria utilizando Latex⁴.

⁴<https://www.latex-project.org/>

Capítulo 3

Herramientas utilizadas

En este capítulo se describen las tecnologías web que se han utilizado en este TFG. En Unibotics se ha utilizado como tecnologías del lado del cliente HTML para dar estructura, CSS para el diseño y JavaScript para definir las acciones. Las tecnologías del lado del servidor han sido el entorno web de Django y la base de datos MySQL¹. También se ha integrado una base de datos llamada Elasticsearch para la recogida de la información y para visualización de estadísticas automáticas se ha utilizado Dash.

3.1. HTML

HTML son las siglas de *HyperText Markup Language* donde "HiperTexto" se refiere a un texto donde hay enlaces a otra página web o a la misma página permitiendo que los documentos estén interconectados entre sí. Con marcado se hace referencia a que HTML define la estructura del documento con marcas, por ejemplo, que parte del documento va a ser un título y dónde se va a encontrar. HTML es un lenguaje, pero no de programación sino de marcado.

La estructura de un documento HTML está compuesta por la definición del tipo de documento con `<!DOCTYPE html>`, el elemento `<html>` y `</html>` para dar comienzo y final al documento HTML, el elemento `<head>` y `</head>` donde se introducen los metadatos como es el idioma del documento o el título que aparece en la pestaña de la página, y el elemento `<body>` y `</body>` donde se escribe todo el contenido que se quiere mostrar a los usuarios

¹My Structured Query Language

[13].

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <!-- cabecera del documento web -->
5   <title>Mi primer documento web</title>
6   <meta charset="utf-8"/>
7   <meta name="author" content="Francesc Ricart"/>
8 </head>
9 <body>
10
11   <!-- cuerpo del documento web -->
12   <p>El lenguaje HTML sirve para aportar contenido
13     y estructura a un documento web</p>
14 </body>
15 </html>
```

Figura 3.1: Estructura HTML

Un elemento de HTML está compuesto por etiquetas que son palabras que marcan el inicio y final de una sección. La etiqueta de apertura está formada por una palabra o letra rodeada por '`<`' y '`>`' dando comienzo al elemento y normalmente con una etiqueta de cierre al igual que la de la apertura pero rodeada por '`<`' y '`>`'. La etiqueta no distingue entre mayúsculas y minúsculas. Aunque haya una gran cantidad de etiquetas a veces se necesita información adicional para completar los elementos, esto se consigue gracias a los atributos.

El atributo se encuentra dentro de la etiqueta de apertura con un espacio en blanco del nombre de la etiqueta o de otro atributo, el atributo está compuesto por un nombre seguido del signo igual (=) y el valor del atributo entre comillas.

Cada etiqueta tiene unos atributos asociados y éstos a la vez unos valores predefinidos. Si se da un valor erróneo a un atributo al renderizar la página esta lo ignorará. Algunos atributos son obligatorios como en el caso de las imágenes, vídeos o enlaces, como se muestra en la Figura 3.2, la etiqueta para los enlaces es *a* y es obligatorio que le siga el atributo *href* para poder añadir la dirección a la que va a dirigir dicho enlace. Entre las etiquetas de apertura y cierre nos encontramos con el texto que será el contenido de la sección [14].

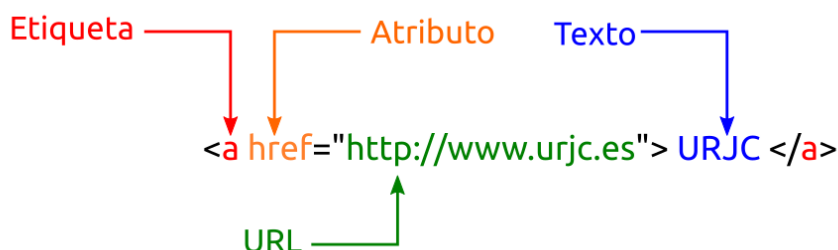


Figura 3.2: Estructura de un elemento HTML

Hay dos tipos de elementos, los bloques, que son los elementos que ocupan toda una línea en el documento estos son los encabezados, listas o párrafos; y los elementos en línea que solo ocupan el espacio de su contenido como los botones, enlaces o imágenes. Esto se puede cambiar gracias a los atributos. Para estructurar el documento disponemos de dos etiquetas generales, `<div>` la cual crea una sección de tipo bloque y `` para crear una sección en línea [15].

La última versión del lenguaje HTML es HTML5, que ha sido la utilizada en este proyecto. Como mejoras respecto a anteriores versiones está la introducción de las etiquetas de audio `<audio>` y vídeo `<video>`. Anteriormente la web no estaba pensada para multimedia y había que meter parches como Flash de Adobe. En HTML5 se introduce SVG² para hacer gráficos vectoriales y así no se verán pixeladas las imágenes. También como novedad tiene Canvas, el cual es un elemento de diseño gráfico y composición de imágenes. Y WebGL, que es una especie de Canvas pero en 3D.

En HTML5 si el usuario del navegador da permiso, incorpora una API³ de geolocalización donde se puede ver la ubicación. Por último, también incorpora lo denominado *Drag and Drop* que consiste en arrastrar y soltar para facilitar la interacción con el usuario.

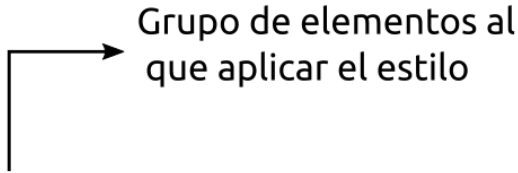
²Scalable Vector Graphics

³Application Programming Interfaces

3.2. CSS

CSS es el lenguaje de estilo que se encarga del diseño y la presentación de los documentos HTML. Para llamar la atención de los usuarios en páginas web es importante añadir estilo a los documentos, por eso se utiliza CSS, el cual puede definirse como un atributo de HTML llamado *style*. Otra forma de meter estilo es utilizando la etiqueta `<style>` en la cabeza del documento HTML. La mejor opción para añadir estilo es separándolo de la estructura, es decir, del documento HTML, creando una hoja de estilo *.css*. De esta forma es más sencillo realizar cambios y se podrá diversificar el trabajo en estructura y estilo, siendo más productivo. Para vincular la hoja de estilo con el HTML se utiliza la etiqueta `<link>` en la cabecera.

La estructura de una regla CSS se divide en selectores que contienen pares de propiedad-valor, como se muestra en la Figura 3.3. En los selectores se pone el nombre de las etiquetas las cuales quieres cambiar su estilo, como puede ser la etiqueta *body*. Además, estos selectores pueden ser el valor del atributo *id*, que representa el identificador único. En este caso se pone el símbolo *#* antes del valor de su *id*. Aparte de identificar un elemento con un *id* se puede utilizar el atributo *class*, que es un identificador para varios elementos. En este caso para añadirle estilo a los elementos pertenecientes a la misma clase al selector se le añade un punto delante.



```
Selector {  
    propiedad1: valor;  
    propiedad2: valor;  
}
```

Figura 3.3: Sintaxis CSS

Las propiedades indican cuál es el estilo que se quiere cambiar en un elemento, como puede ser el color o el tamaño. Cada propiedad tiene unos valores asociados que en algunas ocasiones se pueden escribir de diferente manera, como en el caso de los colores, se puede escribir directamente como *red* o ponerlo en su valor hexadecimal o en valores RGB⁴.

A la hora de utilizar estilos se pueden poner estilos contradictorios, en este caso el último estilo definido será el que se acabe aplicando. Esta es la parte de cascada que indica las siglas CSS. Si se ha utilizado una hoja de estilo, pero además, se ha definido en una etiqueta HTML, el definido en la etiqueta será el utilizado al *renderizar* el documento. La herencia también es un concepto importante en CSS ya que si un elemento no tiene estilo, pero está contenido en otro elemento que sí tiene, éste heredará su estilo. Los identificadores únicos tendrán preferencia a añadir el estilo sobre los identificadores de clase, el nombre de la etiqueta como selector es el de menor preferencia entre los selectores [16].

3.3. JavaScript

JavaScript es un lenguaje de programación interpretado que permite la ejecución de código orientado a eventos. Pueden actuar sobre el navegador a través de objetos integrados como un botón. El DOM⁵, es el modelo de objetos que representa al documento HTML y define la manera de interactuar con él, puede ser modificado dinámicamente gracias a JavaScript. Es importante la colocación del código JavaScript ya que se ejecuta ordenadamente de arriba a abajo.

JavaScript suele estar en el lado del cliente haciendo que su código se ejecute en el navegador web, donde podrá interactuar con el navegador, además, con el documento HTML o dibujar en la página. También existen entornos para el desarrollo de la parte servidor de una plataforma Web que están basados en JavaScript, como Node.js⁶ [17].

⁴Red, Green, Blue,

⁵Document Object Model

⁶<https://nodejs.org/>

Hay diferentes formas de agregar código JavaScript a un documento HTML. Como ocurría con CSS la mejor opción es tener un fichero .js separado de la estructura (HTML) y del estilo (CSS) para poder trabajar de una mejor manera. Para añadirlo en la cabecera del HTML se le inserta la etiqueta `<script>` con el atributo *src* para indicar la ubicación del fichero. Para que no haya ningún problema conviene ejecutar el código cuando se haya cargado la página, para ello se utiliza el atributo *onload* que indica que se ha cargado la página y se llama a una función principal del fichero JavaScript. Otra forma de introducir JavaScript en el documento HTML es directamente en sus etiquetas, por ejemplo, cuando ocurre un evento o mediante la etiqueta `<script>` que ofrece HTML [18].

En este TFG se ha utilizado JavaScript para detectar las diferentes interacciones de los usuarios en la plataforma web de Unibotics a través de eventos. También se ha hecho uso de JQuery⁷, una librería de JavaScript que permite una manipulación más sencilla del DOM y de los eventos que se generan en éste.

3.4. Django

Django es un entorno de desarrollo web de código abierto escrito en Python. Django actualmente dispone de mucha funcionalidad, es decir, viene con extras para ayudar al desarrollo de una web, además es versátil, escalable, rápido y seguro. Una de sus principales ventajas es que levanta automáticamente una página web de administración donde se pueden hacer acciones como retocar la base de datos. Django sigue el Modelo Vista Controlador (MVC), y además incorpora plantillas como se muestra en la Figura 3.4.

⁷<https://jquery.com/>

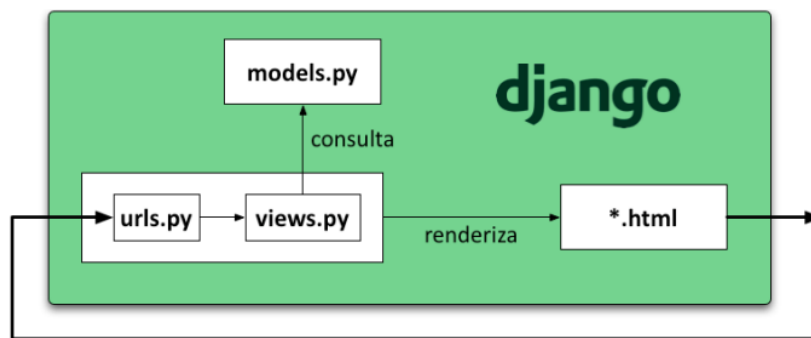


Figura 3.4: Estructura de Django

Los modelos en Django son objetos en Python que son guardados en una base de datos. Los modelos son independientes al gestor de base de datos concreta que se vaya a utilizar, se definen las estructuras de información de una manera genérica y además se puede añadir restricciones. Django puede traducir automáticamente filtros de Python en SQL gracias a su ORM, por lo tanto, no es necesario saber el lenguaje de las bases de datos para hacer consultas, basta con programarlas como filtros de Python. Los modelos están formados por una lista de campos donde se define el dominio de cada campo, si no hay ningún campo que tenga una clave primaria, Django generará una columna llamada *id* que se incrementará automáticamente. Los modelos se escriben en el fichero *models.py* [19].

Las vistas en Django están formadas por el fichero *urls.py* y *views.py*. En el fichero *urls.py* se definen a partir de expresiones regulares las urls que redirigen a funciones de *views.py*. La función *views.py* recibe un objeto *HTTP Request* y todos los parámetros de la URL capturados teniendo que devolver un objeto *HTTP Response*. Lo habitual en web dinámicas es hacer una consulta a la base de datos, generar un contexto que empotra en una plantilla y a través de la cual se *renderiza* devolviendo un *HTTP Response*.

Las plantillas son páginas dinámicas, es decir, son protopáginas las cuáles solo se pueden *renderizar* juntando el contexto (diccionario con los valores que se dan a variables de la plantilla) pasado por las vistas y dando como resultado normalmente un documento HTML. Esto es útil al programar ya que permite hacer páginas dinámicas en pocas líneas

En resumen, para implementar un servidor en Django primero hay que diseñar el modelo de datos, luego se diseñan las urls que enrutarán a las vistas, las cuáles preparan un contexto que juntándolo con la plantilla se genera el *HTTP Response* correspondiente.

La versión actual de Unibotics está basada en Django 2.2.

3.5. Lenguaje SQL

SQL⁸ es un lenguaje de consulta estructurado, se utiliza para definir, manipular y gestionar los datos almacenados en una base de datos relacional. SQL es un estándar reconocido en 1986 por ANSI⁹ y en 1987 por ISO¹⁰

Se necesita un gestor de base de datos RDBMS¹¹ que se encarga de interactuar con la base de datos, por ejemplo MySQL o Access SQL. Algunos de estos gestores trabajan en local y otros en un servidor remoto [20].

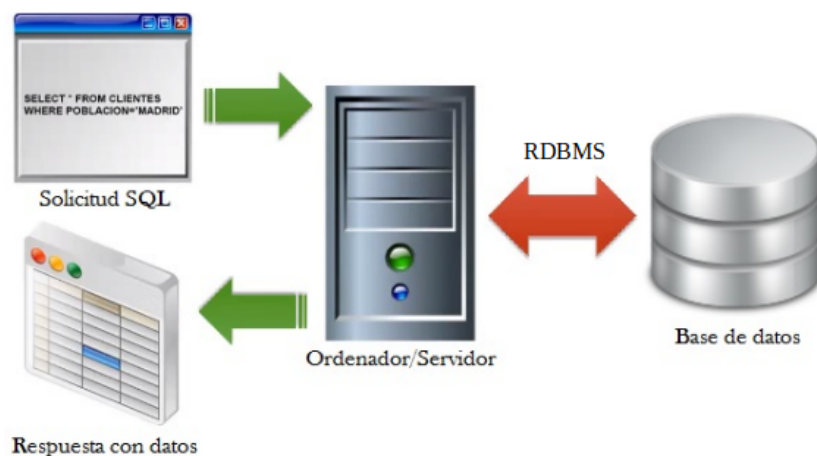


Figura 3.5: Esquema funcionamiento SQL

⁸Structured Query Language

⁹American National Standards Institute

¹⁰International Organization for Standardization

¹¹Relational Database Management System

3.5. LENGUAJE SQL

Una instrucción SQL está formada por comandos, cláusulas, operadores y funciones. Los comandos de una sentencia SQL pueden servir para crear, modificar o hacer consultas a la base de datos, entre otras funciones.

Las cláusulas son condiciones de modificación para poder definir los datos que se quieren seleccionar o manipular. Entre ellas se encuentran *FROM* para especificar la tabla o *WHERE* para definir las condiciones de los registros que se desean.

Los operadores pueden ser lógicos (*AND*, *OR* o *NOT*) o de comparación, que serían las operaciones del estilo mayor que, menor que o igual que. Las funciones se utilizan con el comando *SELECT* para devolver un único valor de un grupo de registros, como puede ser *AVG* que te devuelve la media. En la Figura 3.6 se muestra un ejemplo de una sentencia SQL con todas sus partes [21].

```
SELECT Avg(Gastos) AS Promedio  
FROM Pedidos|  
WHERE Gastos > 100;
```

Figura 3.6: Sentencia SQL

En este TFG se ha utilizado SQL ya que Unibotics utiliza una base de datos MySQL donde se almacena la información estructural de la plataforma como los usuarios o los ejercicios existentes, a la cual se accede a través de Django.

3.6. Elasticsearch

Para la recogida y grabación de las sondas es necesario tener una base de datos, en este caso para este proyecto se ha decidido utilizar una base de datos no relacional (NoSQL¹²). Una base de datos no relacional se caracteriza por almacenar datos no estructurados o semi-estructurados, además, sus datos no están organizados en tablas, como es el caso de las bases de datos relacionales como MySQL.

En este TFG se ha utilizado Elasticsearch¹³, que es un motor de búsqueda y análisis de código abierto, escrito en Java y basado en Lucene. Asimismo, es una biblioteca de Java que proporciona funciones de indexación y búsqueda, entre otras. Elasticsearch está orientado a documentos JSON¹⁴, los cuales están formados por un conjunto de pares clave-valor, donde la clave es una cadena de texto y el valor puede ser de diferentes tipos como un texto o una lista [22].

Elasticsearch se caracteriza por ser una herramienta rápida permitiendo una búsqueda de texto completo bastante eficiente. Gracias al poco tiempo transcurrido entre la indexación (grabación de datos en Elasticsearch) y la posterior búsqueda se ha considerado que es una herramienta adecuada para la grabación de las sondas de Unibotics y la posterior consulta de la información.

Una de las ventajas de las bases de datos no relacionales es que están implementadas para permitir un escalado horizontal, es decir se puede dividir la base de datos en diferentes servidores, de una manera más sencilla que con las bases de datos relacionales.

Los diferentes datos están agrupados en lo que se llama *shards*, en los cuales se aplican técnicas de réplica para ser tolerante a los fallos. Además, si Elasticsearch tiene algún fallo es capaz de detectarlo y reorganizar la información. Existen diferentes módulos en diferentes lenguajes de programación que permiten una interacción sencilla con Elasticsearch [23].

¹²*Not only SQL*

¹³<https://www.elastic.co/>

¹⁴JavaScript Object Notation

Elasticsearch está formado por *clusters* que son un conjunto de nodos, en los cuales se almacenan todos los datos, también puede estar formado por un único nodo. Al *cluster* se le asigna un nombre y un identificador único [24].

Los nodos que se encuentran en un *cluster* son unos servidores únicos que almacenan documentos y ayudan en las capacidades de indexación del *cluster*. Al nodo también se le asigna un nombre y un identificador. Dentro de los nodos se pueden encontrar uno o más índices que son una colección de documentos con características similares, tienen que ser nombrados en minúsculas y este será el nombre al que se hará referencia para realizar las diferentes operaciones. Los documentos que se almacenan en un índice están formados por la información básica que se desea indexar [24].

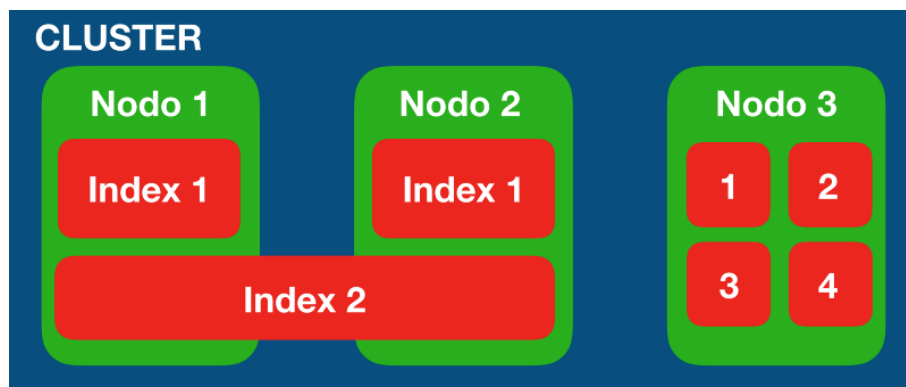


Figura 3.7: Arquitectura Elasticsearch

En el caso de que se guarde una gran cantidad de documentos sobrepasando los límites de almacenamiento de un nodo, Elasticsearch divide cada índice en diferentes fragmentos (*shards*), la distribución de cada fragmento por los diferentes nodos se realiza de forma automática. Para evitar fallos en los nodos se hacen réplicas de los *shards* y se almacenan en un nodo diferente al *shard* primario. Así en el caso de que se produzca un fallo en un nodo se podrá seguir trabajando.

En este TFG se ha utilizado la versión 7.12.0 de Elasticsearch.

3.7. DASH

En este TFG se ha decidido utilizar el entorno Dash¹⁵ para visualizar los datos recogidos. Dash es un *low-code framework*, es decir que permite la creación de aplicaciones de manera rápida y eficiente haciendo un menor uso de la programación manual, está escrito sobre Plotly.js y React.js. Dash está disponible en lenguajes de programación como Python, Julia, R o F. Dash es multiplataforma, por lo que puede ser utilizado desde cualquier dispositivo [25].

En las aplicaciones de Dash no es necesario escribir ningún documento HTML, JavaScript o CSS ya que Dash proporciona una abstracción pura en Python mediante el uso de la biblioteca de `dash_html_components`. Otra biblioteca de Python que se utiliza en una aplicación Dash es `dash-core-components` la cual incluye una serie de componentes para una interfaz de usuario interactiva, como un menú desplegable.

La estructura de los datos que se van a visualizar en Dash son *dataframes* de la biblioteca de Pandas¹⁶. Un *dataframe* es una tabla en la que los datos guardados en cada columna representan las diferentes variables.

Las aplicaciones de Dash están formadas por dos partes. La primera es la llamada *layout* donde se describe cuál va a ser el aspecto de la aplicación, esto son los menús y las gráficas que se visualizan, aquí es donde se utilizaran las bibliotecas mencionadas anteriormente. La segunda parte es la interactividad con los usuarios, para ello se utiliza `dash.dependencies` donde los componentes interactivos crean una entrada y a través de *callbacks* modifican las gráficas, de tal manera que se pueden integrar filtros para generar diferentes visualizaciones en base a las necesidades del usuario.

Para este TFG se ha utilizado la versión 1.17.0 de Dash.

¹⁵<https://dash.plotly.com/>

¹⁶Python Data Analysis Library

Capítulo 4

Analíticas automáticas en Unibotics

En este capítulo se explica la recogida que se ha programado de las sondas en Unibotics, su posterior guardado en la base de datos Elasticsearch, la visualización de dichos datos con el entorno web Dash y la validación experimental del proceso completo.

4.1. Diseño

Para conseguir las analíticas automáticas en Unibotics, se han diseñado varios cambios en la plataforma con las nuevas tecnologías involucradas: Elasticsearch y Dash. Con estas extensiones estructurales se consigue la nueva funcionalidad de recogida y visualización automáticas de datos

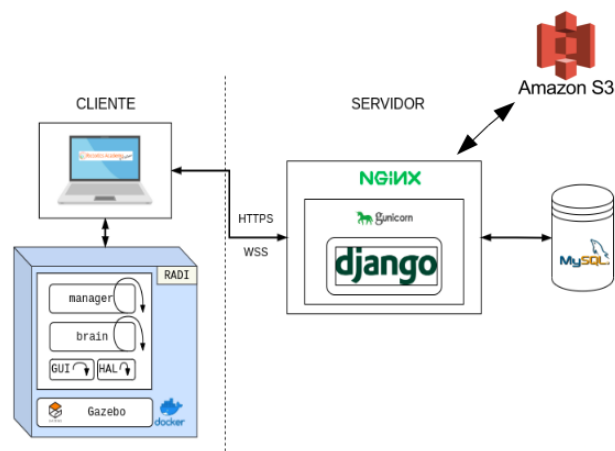


Figura 4.1: Antigua arquitectura de Unibotics

4.1. DISEÑO

En la Figura 4.1 se presenta la arquitectura de Unibotics previa a este Trabajo de Fin de Grado. Como se describe en la sección 1.5, Unibotics está formado por un *frontend*, donde interactúa el cliente conectado a un contenedor RADI que tiene el simulador robótico. La parte del *backend* está compuesta por Django, el cual se conecta a la base de datos MySQL y a la nube de Amazon, S3.

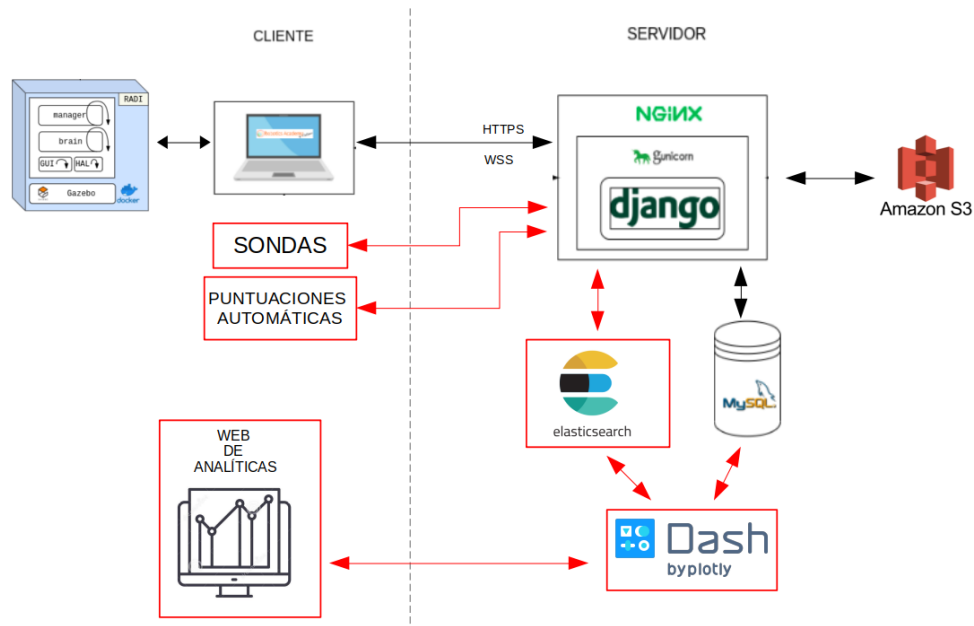


Figura 4.2: Arquitectura Unibotics con las nuevas tecnologías

La Figura 4.2 muestra cómo es la arquitectura nueva de Unibotics después de este TFG. En la parte del *frontend* se ha añadido la recogida de sondas con código JavaScript, al igual que las puntuaciones automáticas obtenidas por los estudiantes en algunos ejercicios. Además, está la página web de analíticas dónde se ven las gráficas. Por otro lado, en el *backend* se ha añadido la base de datos Elasticsearch y la herramienta de Dash para generar esas visualizaciones.

Lo primero que hace un usuario cuando quiere entrar en la plataforma es registrarse en ella, guardando ese registro en la base de datos MySQL. Una vez registrado, podrá iniciar sesión en ella, en el momento que desee. Cuando se encuentre dentro de la plataforma, podrá elegir un ejercicio y acceder a él, dónde podrá programar la solución correspondiente. Dentro de los ejercicios algunos tienen la opción de pedir una evaluación automática de estilo y de eficacia del código programado. Para abandonar la plataforma se puede cerrar sesión explícitamente o de manera implícita, cerrando el navegador o la pestaña.

Cuando un usuario interactúa con la plataforma de Unibotics, la herramienta de Elasticsearch recoge la información de estas interacciones, entrada y salida en la plataforma, entrada y salida de los ejercicios, puntuaciones automáticas, etc. Las visualizaciones de las sondas recogidas se hacen a través de Dash, herramienta independiente a Django. Dash utiliza la base de datos de MySQL y la base de datos no relacional Elasticsearch.

4.2. Recogida de sondas

La primera parte de este proceso ha consistido en la programación del *software* que recoge de diferentes sondas, detectándolas a través de código JavaScript. Una vez detectada la sonda es enviada al *webserver*, el cual es el encargado de guardar la sonda en la base de datos, de modo persistente. Para realizar esta tarea, se ha decidido utilizar la base de datos Elasticsearch por las ventajas que ofrece, explicada en el capítulo 3.

4.2.1. Tipos de sondas

Las sondas, como ya se ha mencionado anteriormente, son los eventos interesantes que se quieren recoger. Este proyecto consta de cuatro tipos de sondas:

1. Entrada y salida de un usuario a la plataforma de Unibotics. Es útil para saber la duración de una sesión, desde dónde accede cada usuario, qué sistema operativo utiliza y desde qué navegador entra a la plataforma.
2. Entrada y salida de un usuario a cada ejercicio que hay en la plataforma. Su principal utilidad es saber durante cuánto tiempo un usuario está realizando un ejercicio en concreto.
3. Puntuación automática de estilo que tiene el código fuente de un usuario para un ejercicio.
4. Puntuación automática de eficacia del código fuente de un usuario para un ejercicio.

4.2.2. Base de datos de eventos interesantes

Para definir y configurar los índices donde se guarda las sondas se ha creado un nuevo archivo, llamado `probe.py`. Al definir un índice, se determinan los nombres de cada campo (información que se quiere almacenar) y el tipo de campo que es, por ejemplo: un texto, un número, una fecha, entre otros. También se puede configurar el índice, por ejemplo, poniendo el número de *shards* o el número de réplicas. Un ejemplo de la definición de un índice sería este:

```
1  from django_elasticsearch_dsl import Document, Date, Text, Double
2
3  class SessionDocument(Document):
4      username = Text()
5      start_date = Date()
6      end_date = Date()
7      duration = Double()
8      client_ip = Text()
9      browser = Text()
10     country = Text()
11     alpha_2 = Text()
12     alpha_3 = Text()
13     continent = Text()
14
15     class Index:
16         name = 'session_log'
17         settings = {
18             'number_of_shards': 1,
19             'number_of_replicas': 0
20         }
```

Para este trabajo se han definido cuatro índices diferentes:

- `session_log`: índice que recoge las sondas relativas a las sesiones. Consta de los campos de inicio y fin de sesión, duración de la sesión, la IP, el *browser* (aporta información sobre el sistema operativo, navegador y dispositivo utilizado), el continente y país del usuario, así como el nombre del usuario.
- `exercises_log`: índice que recoge las sondas relativas a los diferentes ejercicios. Está compuesto por la fecha de entrada en un ejercicio, la fecha de salida, la duración total en el ejercicio, el nombre del ejercicio y el usuario.

- `style_log`: índice que recoge los datos sobre la evaluación del estilo del código de los ejercicios que introducen los estudiantes. Este índice está formado por el campo de la fecha en la que se realiza la evaluación, el nombre del ejercicio, la puntuación sobre 100 y el nombre del usuario.
- `efficacy_log`: índice que recoge los datos sobre la evaluación de la eficacia del código de los ejercicios que introducen los estudiantes. Los campos son iguales que en el índice de `style_log`.

4.2.3. Detección de las sondas

Las sondas se han detectado en el navegador web, en el *frontend* con código JavaScript. La sonda número 1 de la subsección 4.2.1 ha sido capturada haciendo uso de la solicitud que se manda al *webserver* al hacer *login*. Cuando un usuario introduce su *username*, su contraseña y clicla en el botón de *login*, se detecta la entrada del usuario.

Un usuario puede abandonar Unibotics de manera explícita, utilizando el botón de *logout*, o de manera implícita, por inactividad. Cuando la salida es explícita se detecta en el mismo instante que clicla en *logout*. Si no se detecta alguna pulsación del teclado o algún clic del ratón en la plataforma, durante un *timeout* declarado de 30 minutos, se considerará que el usuario es inactivo. Pasado el *timeout*, se detectará como salida del usuario.

La entrada a un ejercicio se detecta cuando se manda una petición de la vista de un ejercicio determinado al *webserver*. La detección de la salida de un ejercicio ocurre cuando se llama a la vista de Django de liberar el docker. También se detecta si se está recargando el ejercicio o si sale directamente, ya que el docker se libera en ambas ocasiones.

Para detectar una recarga, se ha hecho uso del evento *beforeunload*, qué detecta cuándo un usuario va a abandonar la página en la que se encuentra. En ese evento, aparte de llamar a la función que liberará el docker, se almacena en local la fecha en la que se ha activado el evento y el nombre del ejercicio. El código es el siguiente:

4.2. RECOGIDA DE SONDAS

```
1 <script type="text/javascript">
2   window.addEventListener("beforeunload", function (e) {
3     e.preventDefault();
4     var date = new Date().getTime();
5     localStorage.setItem('lastTime', date);
6     localStorage.setItem('exercise', '{{exercise}}');
7     freeDocker();
8   });
9 </script>
```

Todos los ejercicios poseen el evento *DOMContentLoaded*, el cual se activa cuando el documento HTML es cargado completamente. Una vez activado el evento, se comprueba si existe una fecha almacenada en local. La fecha representa cuándo fue la última vez que ese usuario ha entrado en un ejercicio. Si la dirección del ejercicio al que se accede contiene el nombre del ejercicio guardado anteriormente y, además, ha pasado menos de tres segundos desde que se accedió por última vez, se considera una recarga de página. A continuación, se muestra el código de detección de recarga.

```
1 window.addEventListener("DOMContentLoaded", function (e) {
2   console.log("Todos los recursos terminaron de cargar!");
3   var current_location = window.location.href.split('/')
4   var lastTime = localStorage.getItem('lastTime')
5   var exercise = localStorage.getItem('exercise')
6   if(lastTime){
7     lastTime = Number(lastTime)
8     var now = new Date().getTime();
9     var difference = (now - lastTime)
10    if (current_location[current_location.length-2]==exercise) {
11      if(difference < 3*1000){
12        reload()
13      }
14    }
15  }
16 });
```

Para detectar la puntuación recibida en la evaluación de eficacia se ha creado un nuevo botón en los ejercicios. Una vez pulsado el botón, empieza a ejecutar el código durante un tiempo determinado, según el ejercicio. Pasado ese tiempo se envía la nota obtenida del ejercicio al *webserver*. Si el usuario pulsa una segunda vez a ese botón antes de que pase el tiempo de medición se considera que el usuario desiste y no desea la evaluación automática, por lo cual la sonda no se grabará. En la plataforma ya constaba un botón de evaluación de estilo, el cual envía al *webserver* el código fuente del ejercicio.

4.2.4. Procesamiento y grabación de las sondas

Una vez definidos en código fuente los diferentes índices, se importan en el archivo `views.py` para poder crearlos. Las sesiones de los usuarios se guardan en el índice `session_log`. A través de las sondas creadas, se recoge cuándo el usuario entra en la plataforma y cuándo finaliza su actividad en ella. Se recoge la sonda de la siguiente manera:

```
1 probe_session = SessionDocument(username=username, start_date=datetime.now(),
2                                 end_date=datetime.now(), duration=0, client_ip=ip,
3                                 browser=request.META['HTTP_USER_AGENT'],
4                                 country=location_info["country_name"],
5                                 alpha_2=location_info["alpha_2"],
6                                 alpha_3=location_info["alpha_3"],
7                                 continent=location_info["continent"])
8 probe_session.save()
```

Gracias al objeto *HTTP Request* que recibe el fichero `views.py`, se obtiene la información del nombre del usuario, la IP y el *user-agent*. En el *user-agent* se encuentra el sistema operativo, el navegador o el dispositivo que utiliza el usuario para acceder a la web de Unibotics. Para la localización espacial se ha creado una función que, a partir de la IP, muestra la ubicación geográfica. Cuando el usuario entra, los campos de fin de sesión y duración se inicializan con la fecha actual y 0 respectivamente. Una vez que el usuario haga *logout* o finalice su sesión por inactividad estos campos se modificarán como se muestra a continuación:

```
1 s = Search(index="session_log").query('match', username=request.user.username) \
2     .sort({"start_date": {'order': 'desc'}})[0]
3 for hit in s:
4     end = datetime.now()
5     Elasticsearch(settings.ELASTICSEARCH_DSL['default']['hosts']) \
6         .update(index="session_log", id=hit.meta.id,
7               body={"doc": {'end_date': end,
8                             'duration': (end - datetime.strptime(hit.start_date, '%Y %m %dT %H: %M: %S. %f')) \
9                             .total_seconds()}})
```

Realizar los cálculos de la duración y la posición del usuario evita que a la hora de visualizarse se tengan que hacer dichos cálculos de todos los datos, ahorrando tiempo.

4.2. RECOGIDA DE SONDAS

Las sondas relativas a los ejercicios se guardan cuando el usuario accede a un ejercicio concreto. Como ocurre con las sesiones, cuando el usuario abandone el ejercicio se modificarán los datos de duración y fin del ejercicio. Se ha tenido en cuenta que al recargar un ejercicio se crea una nueva sonda no deseada. Dichas sondas se eliminan de la siguiente forma:

```
1 es = Search(index="exercises_log").query('match', duration=0) \
2     .query('match', username=request.user.username) \
3     .sort({"start_date": {'order': 'desc'}})
4 for hit in es:
5     Elasticsearch(settings.ELASTICSEARCH_DSL['default']['hosts']) \
6         .delete(index="exercises_log", id=hit.meta.id)
```

En el *webserver* se evalúa el código enviado con el botón de evaluación de estilo, dando unas recomendaciones para mejorar el estilo del código al usuario y una puntuación. La puntuación es recogida en el índice de `style_log`. Por otro lado, la puntuación de eficacia es enviada al *webserver* y se guarda directamente en el índice de `efficacy_log`.

En este proceso de recogida de sondas y su grabación ha sido muy útil la utilización de la API que proporciona Elasticsearch para poder depurar y comprobar los datos que se estaban almacenando. Utilizando por ejemplo la URL `http://127.0.0.1:9200/session_log/_search/?size=10000pretty`, se comprueban las sondas de sesiones.

4.2.5. Despliegue

En este proyecto se ha utilizado la imagen oficial de Docker de Elasticsearch debido a que permite que funcione en cualquier sistema operativo y replicar su instalación en cualquier máquina es directo. Primero hay que descargar la imagen de Elasticsearch con la versión deseada, en este caso se ha utilizado la 7.12.0. El comando para descargar la imagen es:

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:7.12.0
```

4.2. RECOGIDA DE SONDAS

Para ejecutar el contenedor con la imagen descargada se utiliza el siguiente comando:

```
docker run --name=AcademyElastic -p 9200:9200 -p 9300:9300
    -e "discovery.type=single-node"
    docker.elastic.co/elasticsearch/elasticsearch:7.12.0
```

Una vez puesto en marcha el contenedor de Elasticsearch, se tiene lanzada la base de datos donde se guardarán las sondas. Es posible realizar solicitudes directamente desde el navegador web accediendo a la URL donde esté desplegada la base de datos, por ejemplo: `http://localhost:9200/`.

El siguiente paso es la integración de Elasticsearch en el servidor basado en Django. Para ello se hará uso de la librería `django-elasticsearch-dsl`. Además, se ha modificado el archivo de configuración del proyecto, `settings.py`. Se ha incluido la librería a las aplicaciones instaladas y creado una nueva variable, llamada `ELASTICSEARCH_DSL`. Se ha añadido en `settings.py`, ya que las variables declaradas en ese fichero se pueden utilizar en cualquier parte del servidor. En esa variable se indica cuál es el servidor de Elasticsearch con el que se tiene que conectar y sincronizar.

Previamente, para comprobar el funcionamiento de Elasticsearch se generaron datos de prueba para comenzar a trabajar antes de tener los datos reales, los cuales lleva tiempo recoger. La base de datos *Elasticsearch dummy* se ha creado gracias a las librerías de Python Faker¹ y Tornado², y en ella se pueden modificar las sondas. Por ejemplo, el número de sondas, de réplicas o de *shards*. Esto ayudará a futuros desarrolladores a utilizar la base de datos Elasticsearch. En la Figura 4.3 se muestra el resultado de la sonda de evaluación de estilo de la base de datos de prueba.

¹<https://faker.readthedocs.io/en/master/>

²<https://www.tornadoweb.org/en/stable/>

```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
      "relation" : "gte"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "style_log",
        "_type" : "doc",
        "_id" : "0Zp7EHoB05z3ipWglPPg",
        "_score" : 1.0,
        "_source" : {
          "username" : "Dora",
          "date" : "2021-06-07T09:32:32.00",
          "exercise" : "drone_cat_mouse",
          "score" : 6
        }
      },
      {
        "_index" : "style_log",
        "_type" : "doc",
        "_id" : "0pp7EHoB05z3ipWglPPg",
        "_score" : 1.0,
        "_source" : {
          "username" : "Regina",
          "date" : "2021-06-02T06:34:13.00",
          "exercise" : "obstacle_avoidance",
          "score" : 30
        }
      }
    ]
  }
}
```

Figura 4.3: Elasticsearch dummy

4.3. Visualización de la información

Como se explica en el capítulo 3, Dash es un entorno web que permite crear tableros web interactivos con visualizaciones dinámicas, para poder hacer análisis. En este Trabajo de Fin de Grado se ha decidido utilizar esta herramienta para la visualización de las sondas recogidas en Elasticsearch.

Dash recibe los datos crudos guardados en Elasticsearch. Los datos se han recogido en producción desde el 5 de mayo de 2021. Como resultado se ha obtenido la información de usuarios reales desde esa fecha. Hay algunas visualizaciones que requieren de datos que no están guardados en Elasticsearch, por lo que Dash tiene que calcularlos. Por ejemplo, en algunas visualizaciones se representa la media por día, la cual tiene que calcularse.

En la plataforma de analíticas se podrá filtrar y agregar los datos que se representan. Se puede acceder a la plataforma de analíticas a través de la dirección <https://analytics.unibotics.org/>.



Figura 4.4: Esquema de Dash

Solo los usuarios registrados en Unibotics podrán acceder a las visualizaciones. Dependiendo del tipo de usuario, se tiene disponible unas visualizaciones u otras. Para que un administrador o un usuario normal inicie sesión en Dash, se ha hecho uso de los usuarios ya guardados en la base de datos MySQL de la que depende Django de Unibotics, en la que se encuentra la información sobre el tipo de usuario (*staff*, *admin*, *user*...). En el caso de los administradores podrán ver la información de todas las sondas de todos los usuarios, de manera individual y principalmente de manera agregada, mientras que un usuario normal de Unibotics solo podrá acceder a las puntuaciones de estilo y de eficacia obtenidas en cada ejercicio por él mismo. La Figura 4.5 muestra el menú disponible para los administradores.

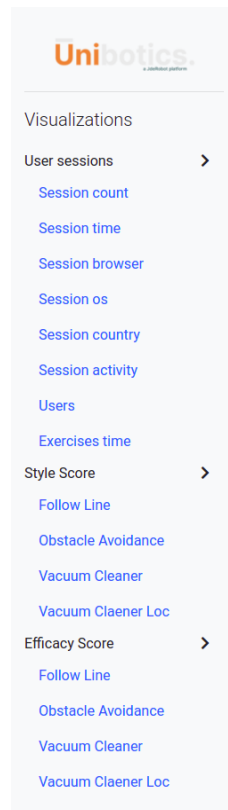


Figura 4.5: Menú de un administrador en Dash

Gracias a las bibliotecas de Dash, mencionadas en el capítulo 3, se ha dado estilo y formato a la aplicación web. También se ha hecho uso de las *cookies* del navegador, guardadas al inicio de sesión de Dash, para comprobar si se está autorizado y si es un administrador de la plataforma.

Dash trabaja con *dataframes*, por lo que es necesario realizar una primera transformación de los datos de Elasticsearch a esta estructura. Esta transformación se realiza gracias a la biblioteca Pandas. Un ejemplo de esta conversión es:

```
1 s = Search(using=es, index="session_log")
2 results = [hit.to_dict() for hit in s.scan()]
3 df = pd.DataFrame(results)
```

4.3.1. Filtros habituales

Con la biblioteca `dash_core_components` se han creado los filtros que algunas de las gráficas poseen. Estos filtros cambian de forma dinámica las visualizaciones en base a las opciones elegidas en dicho filtrado. Uno de los filtros que más se ha utilizado es el *filtro por fechas*, estableciendo una fecha de inicio o de fin o ambas. Es posible conocer la situación de Unibotics en un periodo de tiempo concreto. El código de filtrado es:

```
1 if start_date is not None and end_date is not None:
2     df=df.loc[(df['start_date'] > start_date) & (df['start_date'] <= end_date)]
3 elif start_date is not None:
4     df=df.loc[(df['start_date'] > start_date)]
5 elif end_date is not None:
6     df=df.loc[(df['start_date'] <= end_date)]
```

Otro filtro recurrente en la mayoría de las visualizaciones es el *filtro por usuario*. Este filtro solo aparece en las gráficas accesibles por los administradores, de forma que se puedan visualizar los datos de los diferentes usuarios concretos de la plataforma. Para este filtro se hace uso de las sondas de sesiones, recogiendo los nombres de los usuarios de forma única y añadiendo un 'Total' en los casos que se quiera visualizar las sondas de todos los usuarios. En resumen, el filtro podrá filtrar por usuario único, un grupo de usuarios o por el total de los usuarios, `df = df[df.username.isin(user)]`. El código fuente creado para conseguir los nombres de los usuarios y añadir la opción de 'Total', es el siguiente:

```
1 s = Search(using=es, index="session_log")
2 results = [hit.to_dict() for hit in s.scan()]
3 df = pd.DataFrame(results)
4 df = df[df['username'].notna()]
5 users = df['username'].unique()
6 if not exercises:
7     users = np.insert(users,0,'Total')
8 return users
```



The image shows a user interface for filtering data. At the top, there is a date range selector with two input fields labeled 'Start Date' and 'End Date' separated by a right-pointing arrow. Below this, the text 'Users:' is followed by a dropdown menu. The dropdown menu is currently open, showing 'Total' as the selected option, with a small downward arrow at the end of the list.

Figura 4.6: Filtros utilizados en Dash

Dash utiliza el módulo Plotly para generar las visualizaciones. Plotly ofrece una gran variedad de gráficas que se pueden emplear en Dash. Adicionalmente se puede combinar varias gráficas en los mismos ejes, haciendo más sencilla la correlación entre datos.

En las siguientes subsecciones se detallan los resultados de las analíticas que se han programado para este TFG con datos de usuarios reales de la plataforma de Unibotics. Se muestran las diferentes gráficas creadas tanto para administradores como para los usuarios normales, así como una explicación de lo que representan.

4.3.2. Visualización del número de registros y usuarios de la plataforma

Los administradores tienen la opción de ver el número total de usuarios activos, que se definen como aquellos que han iniciado sesión en los últimos 2 meses. Además, se muestran las gráficas de línea de registros por cada día (Figura 4.7), registros acumulados por días (Figura 4.8) y usuarios activos (Figura 4.9). Cada una de estas gráficas contiene su propio filtrado por fechas.

Para la gráfica de usuarios activos, empezando por la primera sesión recogida en `session_log` hasta el día actual, se comprueba cuántos usuarios han iniciado sesión desde 2 meses atrás hasta el día que se está comprobando. El código fuente para crear una gráfica de línea es el siguiente, en este caso del número de usuarios activos por día:

```
1 def get_temporal_figure_active_users(df):
2     fig = px.line(df, x='Date', y='Active Users')
3     fig.add_trace(go.Scatter(x=df["Date"].tolist(),
4                             y=df["Active Users"].tolist(),
5                             mode="markers", textposition="top center",
6                             name="Active Users",
7                             text=df["Active Users"].tolist()))
8     return fig
```

Las gráficas de las Figuras 4.7 y 4.8 han sido elaboradas con la base de datos MySQL. Se ha realizado una lista de todas las fechas de los registros en Unibotics. Esta lista se ha convertido en *dataframes*. Se ha contado el número de repeticiones de cada fecha y su acumulación, dando como resultado dichas gráficas.

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

En marzo de 2021, la Figura 4.7 refleja el registro de los alumnos de 'Visión en Robótica' en el Master de visión artificial URJC, curso 2020-2021. Y la subida de septiembre-octubre de 2021 refleja el registro de los estudiantes de 'Robótica móvil' y 'Robótica de servicio' en el grado de Ingeniería Robótica Software, curso 2021-2022.

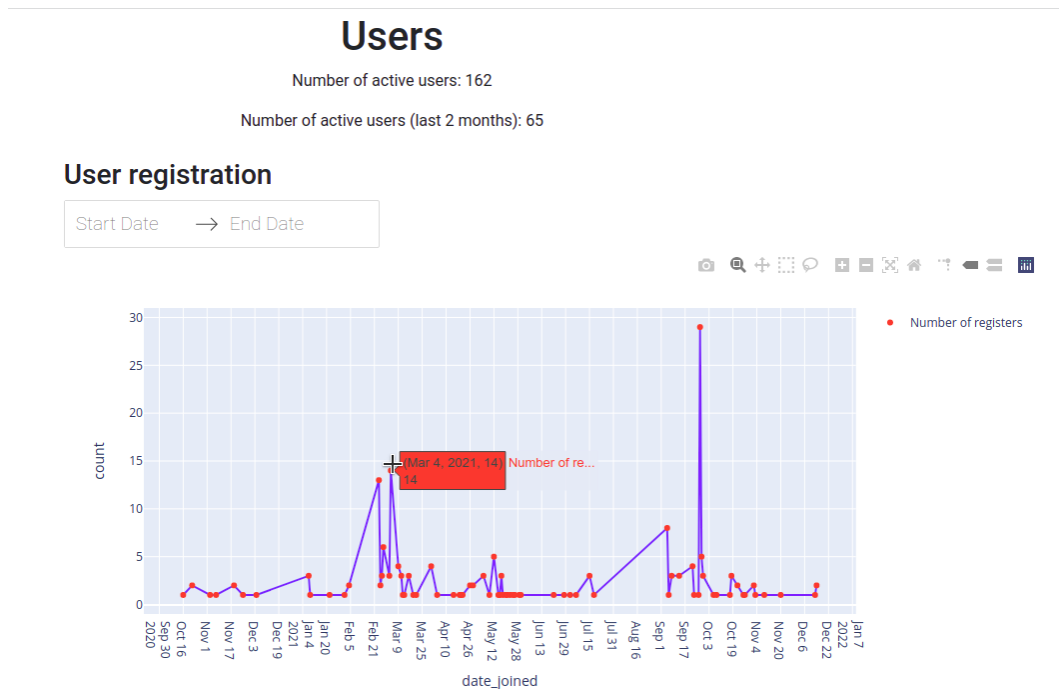


Figura 4.7: Gráficas relativas a los usuarios

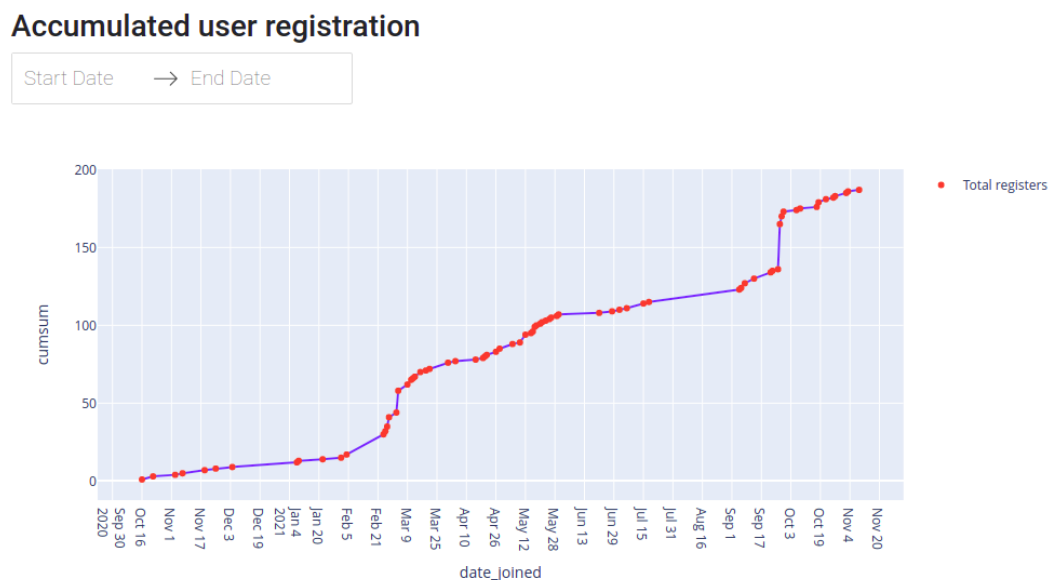


Figura 4.8: Gráfica registro de usuarios acumulado

Active users

Start Date → End Date

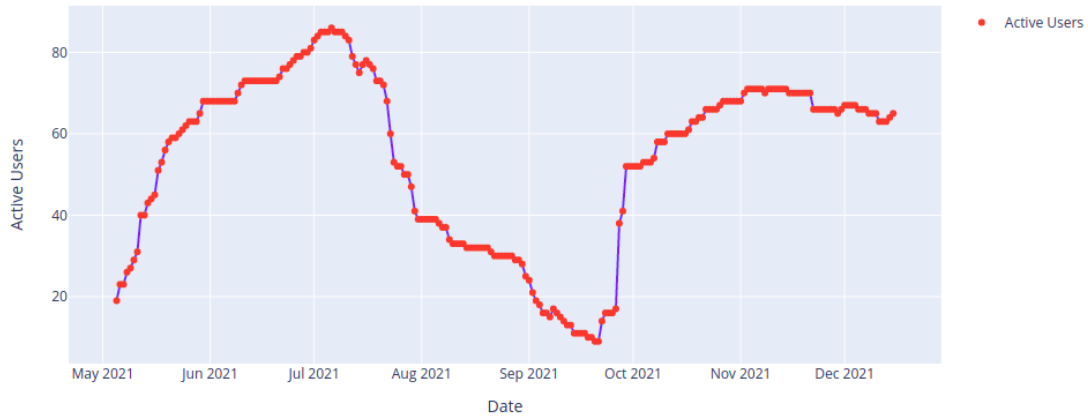


Figura 4.9: Gráfica de evolución de usuarios activos cada día

En la Figura 4.9 se muestra cómo en los meses de verano han disminuido los usuarios activos. Esto se debe a la finalización de los periodos lectivos de la universidad, donde se utiliza Unibotics.

En todas las gráficas de Dash si se pone el cursor sobre uno de los puntos se puede ver la información con mayor detalle, como se muestra en la Figura 4.7. Además, Dash añade varias opciones para interactuar con la gráfica, por ejemplo, se puede descargar la gráfica, hacer *zoom* o seleccionar una parte de ella.

4.3.3. Visualización de actividad en la plataforma Unibotics

La gráfica creada que se muestra en la Figura 4.10 representa en el eje X el tiempo y en el eje Y el número de sesiones, dando como resultado una gráfica de línea, con énfasis en los puntos para una mejor visualización. Esta gráfica muestra el número de sesiones iniciadas por día. En la gráfica se observa qué días ha habido más *logins*, viéndose cómo en los meses de verano dicho número se ha reducido. Hace uso del índice `session_log`, contando cuántas veces el campo `start_date` se repite.

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

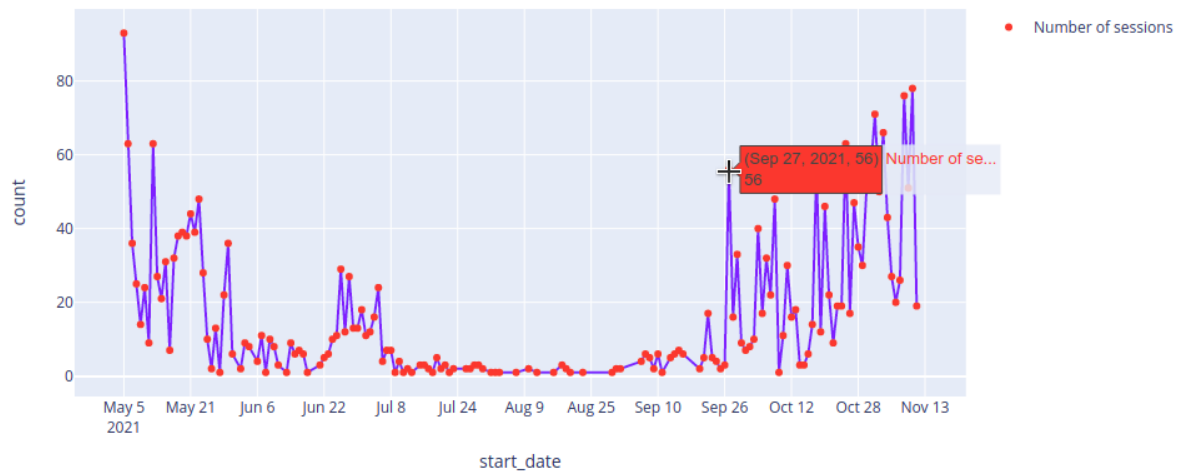


Figura 4.10: Gráfico sesiones totales por día

Se ha creado otra gráfica lineal de sesiones por día, pero en este caso solo se tiene en cuenta una sesión al día como máximo por cada usuario, es decir se mide el número de usuarios únicos que han entrado en Unibotics en cada día. La gráfica está creada de la misma manera que la gráfica del total de sesiones por día y con los mismos filtros (fechas y usuarios). Para que sean usuarios únicos, se ha eliminado los *dataframes* cuya columna de *username* se repetía. En la Figura 4.11 se muestra una parte ampliada de dicha gráfica.

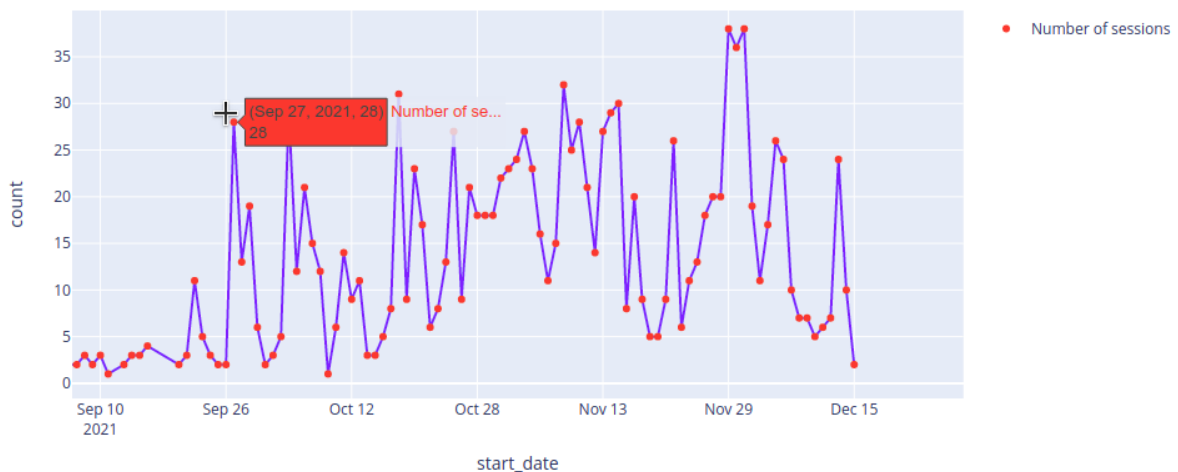


Figura 4.11: Gráfico sesiones únicas por día

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

La Figura 4.12 muestra una gráfica de barras en la cual se representa el tiempo mediante el eje X y la duración total de la sesión de los usuarios en minutos en el eje Y. Además, se ha incluido una línea que representa la duración media de las sesiones. Aquí podemos comprobar que a veces la media coincide con la duración total debido a que solo ha habido una sesión en ese día. Para realizar esta gráfica se han utilizado los campos de *duration* y *start_date* del índice *session_log*. Para crear la gráfica de barras junto con la gráfica de línea de la media, se utiliza el siguiente código:

```
1 def get_temporal_figure__session_time(df):
2     fig = px.bar(df, x='start_date', y='duration')
3     fig.add_trace(go.Scatter(x=df['start_date'],
4                             y=df['mean'],
5                             line=dict(color='red'),
6                             name='Mean'),
7                     row=1, col=1)
8     return fig
```

Total Time

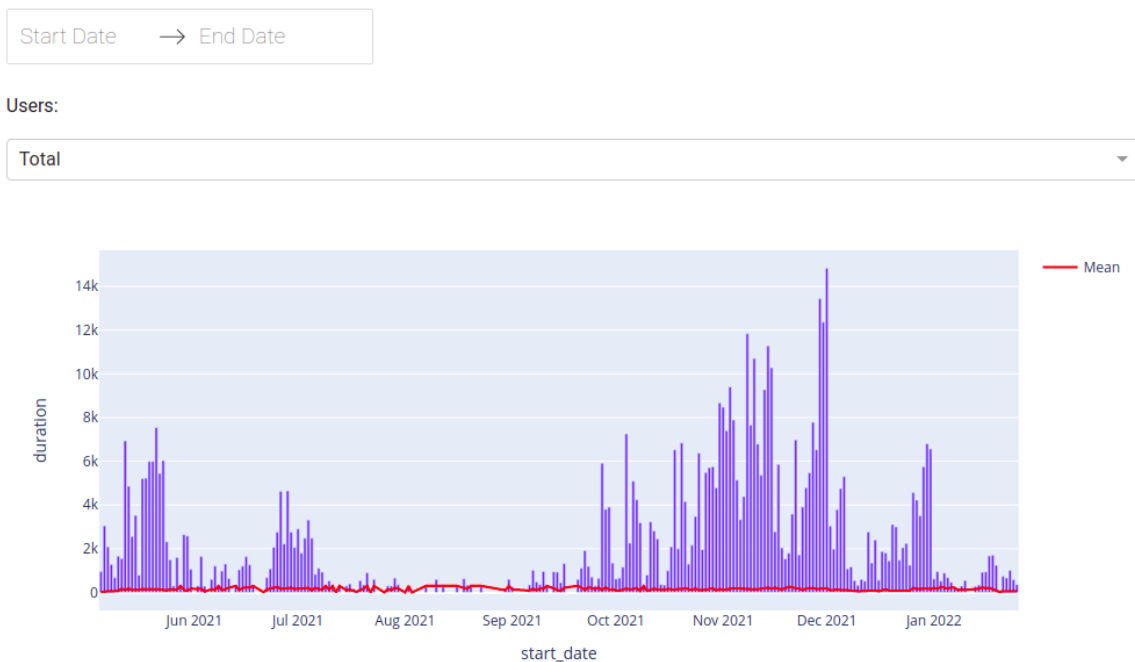


Figura 4.12: Gráfico de tiempo de uso en Unibotics

Esta gráfica se puede filtrar tanto por fechas como por usuarios, pudiendo ver el tiempo dedicado de un usuario y la media total del tiempo en minutos que usa Unibotics dicho usuario, como se puede ver en la Figura 4.13.

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

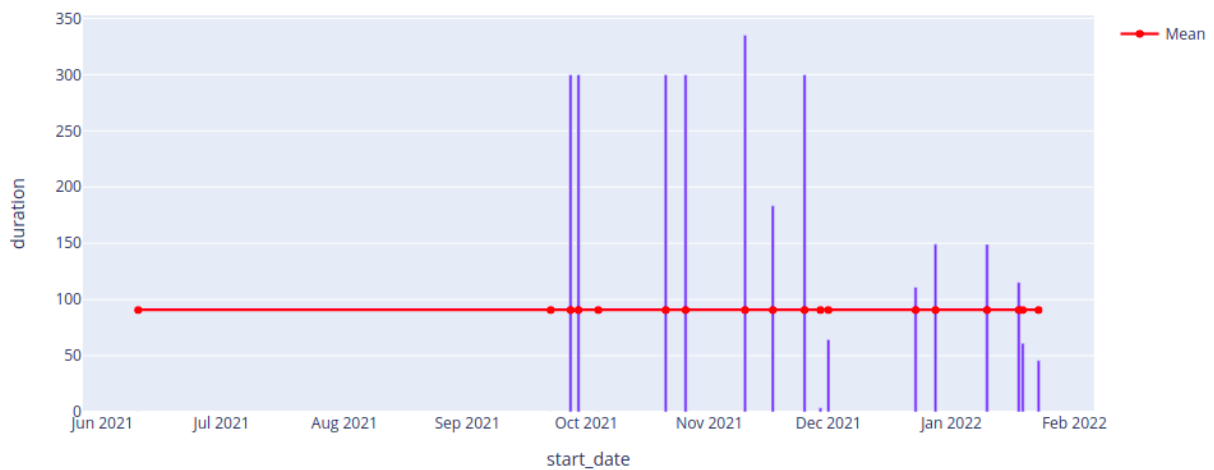


Figura 4.13: Gráfico de tiempo en Unibotics de un usuario concreto

A fin de poder hacer un análisis de la media, la desviación típica o la moda se ha creado un histograma de las duraciones de las sesiones, como se ve en la Figura 4.14. En el eje X, se encuentran los diferentes intervalos de duraciones de los usuarios. En el eje Y, representa cuántos usuarios han utilizado la plataforma durante el mismo tiempo. Para hacer el histograma de duración, se ha programado el código:

```
1 def get_temporal_figure__histogram_time(df):
2     fig = px.histogram(df, x='duration')
3     fig.update_layout(bargap=0.1)
4     return fig
```

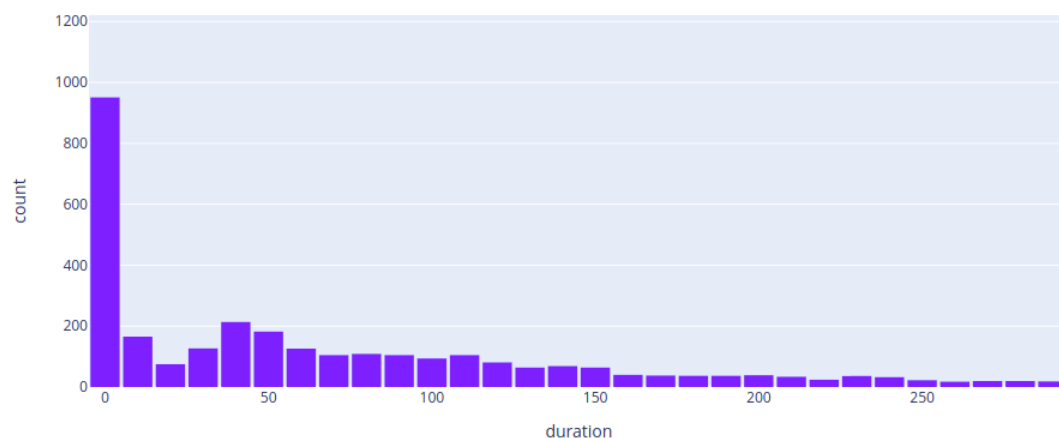


Figura 4.14: Histograma de las duraciones de sesiones

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

Con el propósito de conocer el tiempo que invierten los usuarios *en cada ejercicio* se ha elaborado un histograma de las duraciones de los ejercicios. Esta gráfica dispone de un filtro del ejercicio que se desea comprobar, con el filtro de usuario y el de fechas. Por ejemplo, la Figura 4.15 es un histograma del ejercicio *follow_line* de un grupo de usuarios y unas fechas concretas. Para realizar el histograma, se ha utilizado el índice de `exercises_log`, en concreto los campos de *exercise* y *duration*.

Time histogram in an exercise

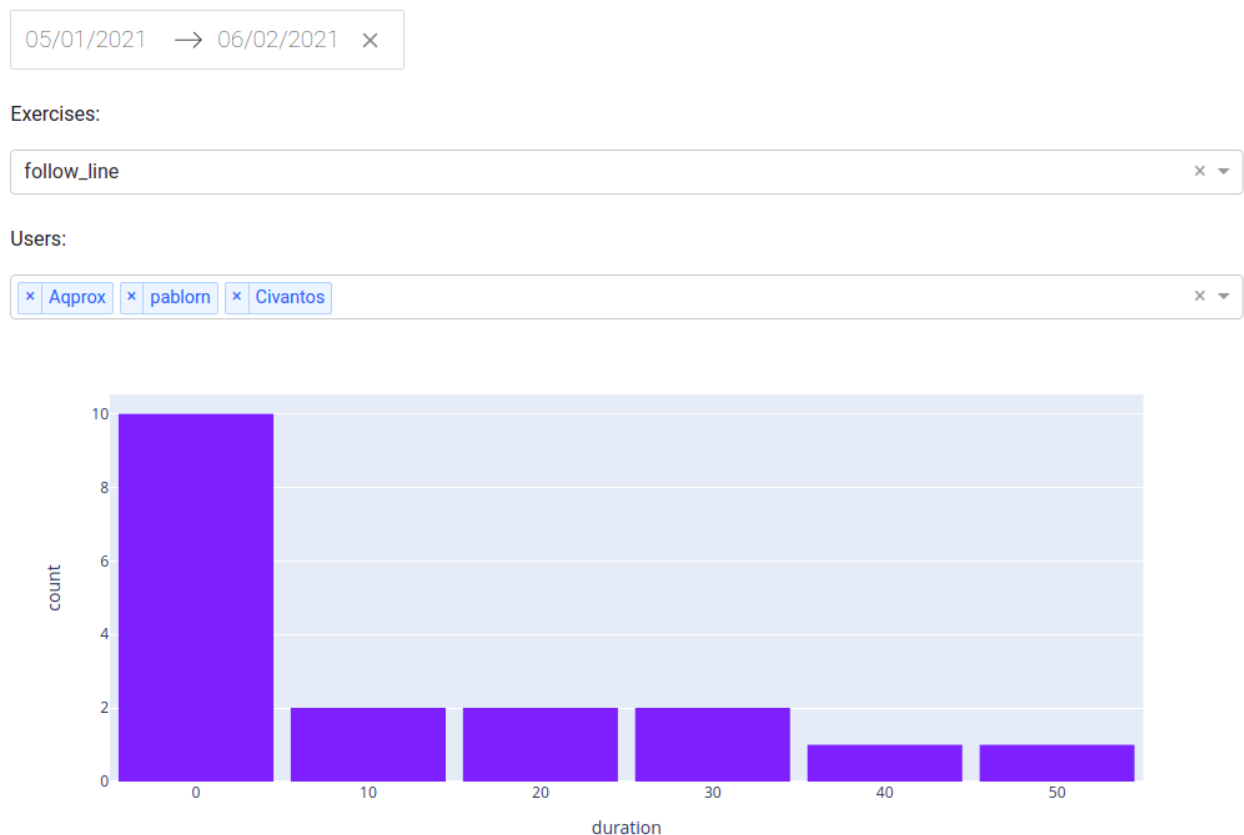


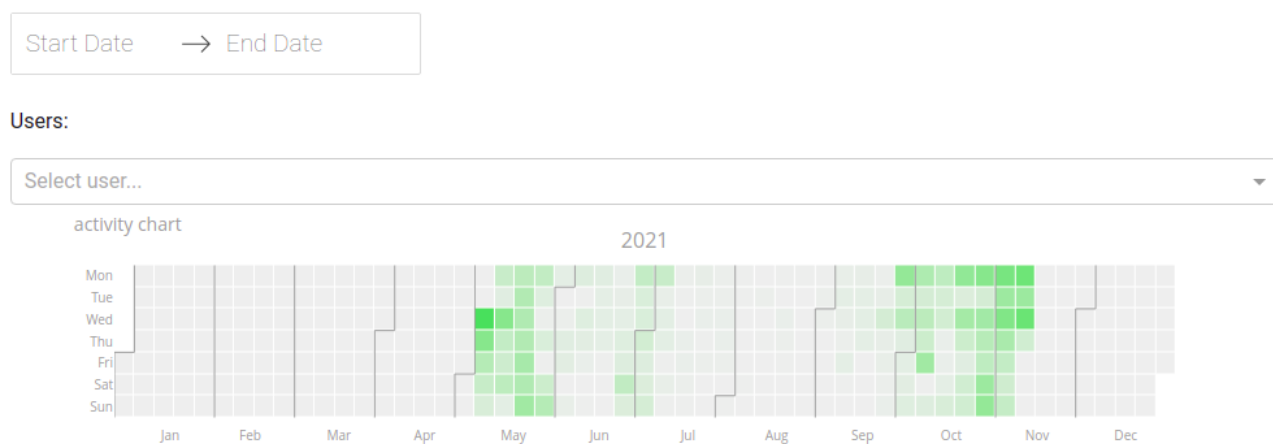
Figura 4.15: Histograma del ejercicio `follow_line` de un grupo de usuarios

La Figura 4.16 representa un mapa de calor con la actividad de los usuarios. Estas gráficas están inspiradas en las que usa GitHub para mostrar la actividad de un desarrollador de software en sus repositorios. Está dividido en cuadrados que representan cada día de un año, cuanto más intenso es el color verde más sesiones ha habido. A medida que disminuyen las sesiones la intensidad también baja. Aparte del filtrado por fechas, tiene un filtro por usuario para ver la actividad por usuarios únicos o un grupo de ellos. Como ocurría con la gráfica lineal de sesio-

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

nes, el primer mapa de color cuenta el total de las sesiones y el segundo mapa de color cuenta las sesiones únicas por usuario. Las sesiones se han contado a través del campo de *start_date* del índice de *session_log*. En las gráficas se refleja cómo en los periodos lectivos hay una mayor actividad que en verano.

Total users activity



Total activity of unique users

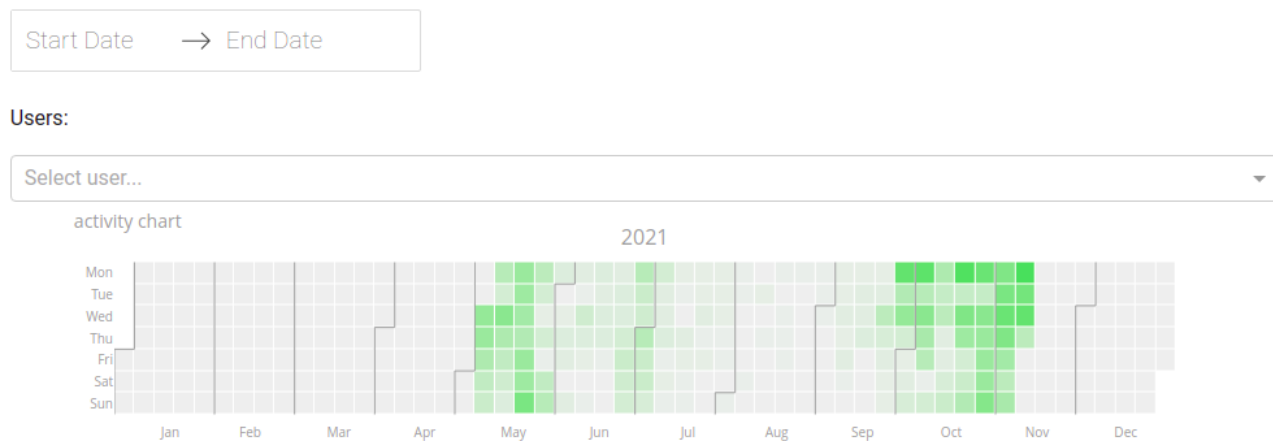


Figura 4.16: Mapas de calor sesiones

4.3.4. Visualización de los metadatos de los usuarios

En la Figura 4.18 se muestra un mapa geográfico con la localización espacial de los usuarios que acceden a Unibotics. La gráfica está preparada para cuando Unibotics incorpore usuarios de todas las partes del mundo. El tamaño de los puntos depende de la cantidad de sesiones, cuanto mayor sea el punto más sesiones hay. A la derecha se encuentra una leyenda con los países, se puede seleccionar uno o varios países en la leyenda para que solo se vean ellos en el mapa. Se puede filtrar por fechas. Para esta gráfica se han necesitado los campos de *country*, *continent* y *alpha_3* (código de los países), del índice de *session_log*. Para hacer el mapa se ha programado el siguiente código fuente:

```
1 def get_temporal_figure__country(df):  
2     fig = px.scatter_geo(df, locations="alpha_3", color="country",  
3                         hover_name="country", size="count",  
4                         projection="natural earth")  
5     return fig
```

Start Date → End Date



Figura 4.17: Mapa geográfico de sesiones

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

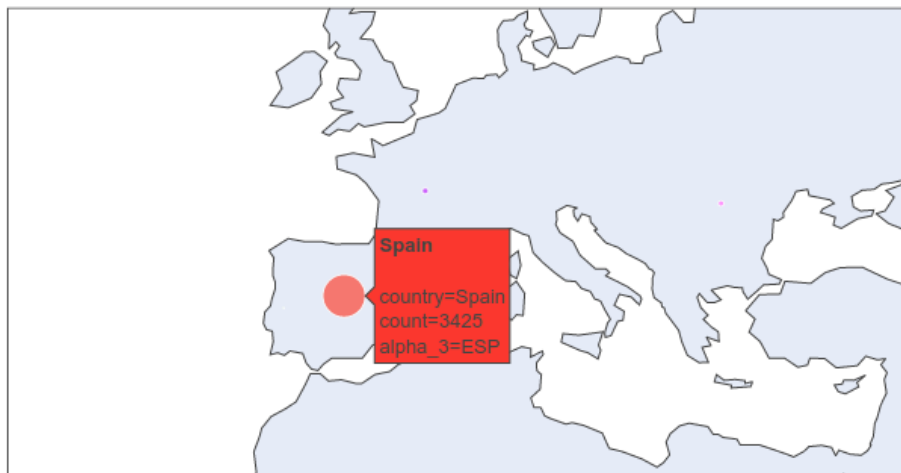


Figura 4.18: Mapa geográfico de sesiones de España

La Figura 4.19 trata de una gráfica con los distintos navegadores que utilizan los usuarios para acceder a Unibotics, en formato porcentajes. En este caso se ve que una gran parte de inicio de sesiones es a través del navegador de Chrome.

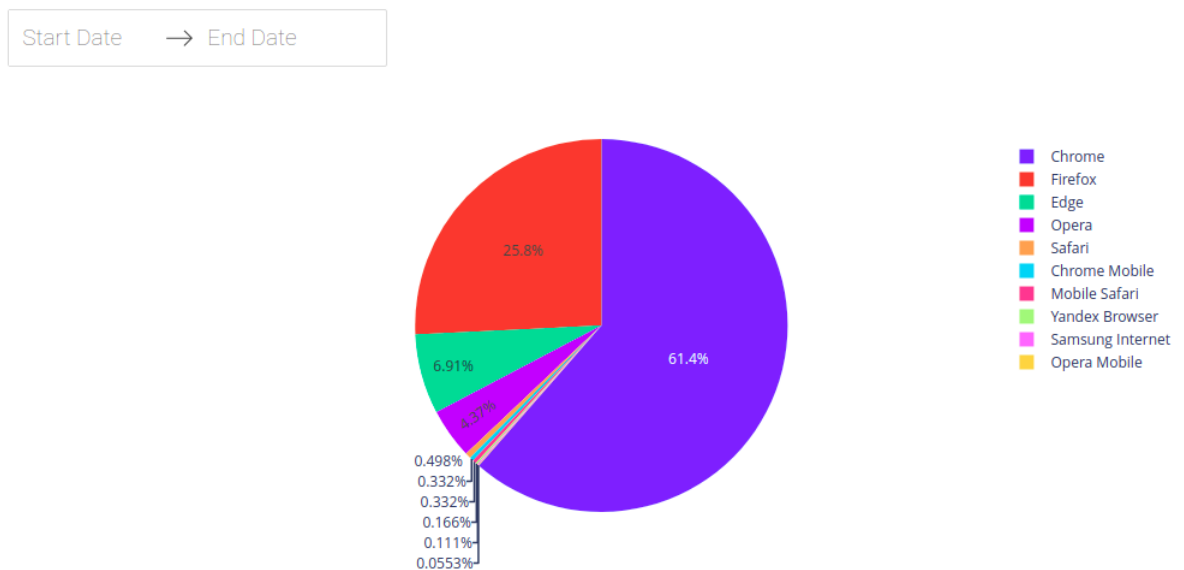


Figura 4.19: Gráfico de navegadores usados por los estudiantes

Se ha creado una gráfica similar a la anterior para representar los Sistemas Operativos utilizados por los usuarios que acceden a Unibotics, como se puede ver en la Figura 4.20. Ambas gráficas tienen filtro por fechas.

4.3. VISUALIZACIÓN DE LA INFORMACIÓN

En ambas gráficas circulares se ha hecho uso del campo *browser* del índice de *session_log*, el cual da la información sobre el sistema operativo y el navegador utilizado. Para realizar dichas gráficas circulares se programa el siguiente código:

```
1 def get_temporal_figure__browser(df):  
2     fig = px.pie(df, values='count', names='browser')  
3     return fig
```

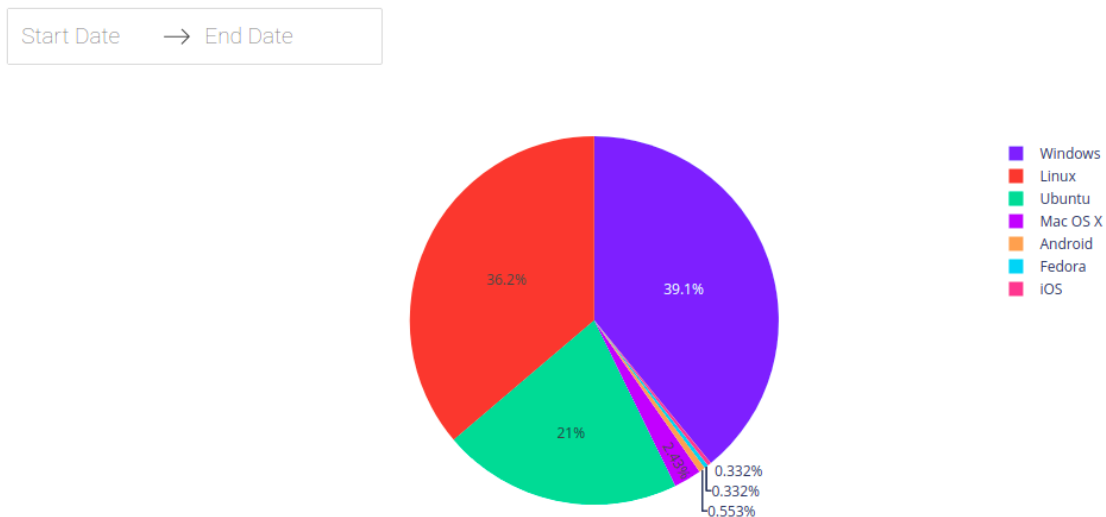


Figura 4.20: Gráfico de sistemas operativos

4.3.5. Visualizaciones de las puntuaciones automáticas

Las siguientes gráficas son las puntuaciones de estilo y eficacia que podrán ser vistas tanto por los usuarios (que pueden acceder a sus propias puntuaciones) como por los administradores (que pueden acceder a las puntuaciones de todos los usuarios). Actualmente, las evaluaciones solo están disponibles en cuatro ejercicios. Se representa en una gráfica en la que cada punto es una evaluación solicitada por el usuario. Las gráficas mostradas en las Figuras 4.21 y 4.22 son un ejemplo de las notas de estilo y eficacia de un usuario de la base de datos de prueba en el ejercicio *follow_line*. Las gráficas de los demás ejercicios son iguales. Para crear estas gráficas se utilizan todos los campos del índice de *style_log* y *efficacy_log*. El código de ambas gráficas es:

```
1 def score_fig(df):  
2     fig = px.scatter(df, x="date", y="score")  
3     return fig
```

Style Follow Line

Users:

Custodio

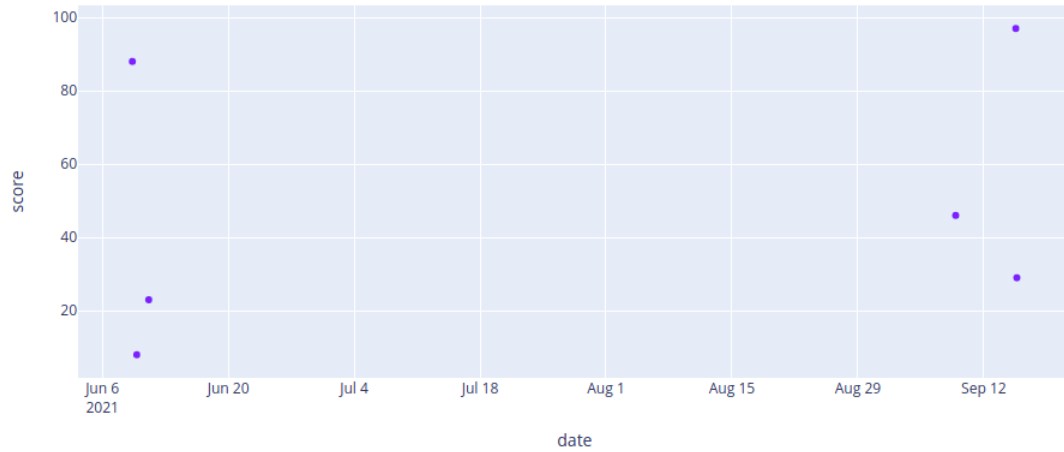


Figura 4.21: Gráfica de puntuación de estilo

Efficacy Follow Line

Users:

Itziar

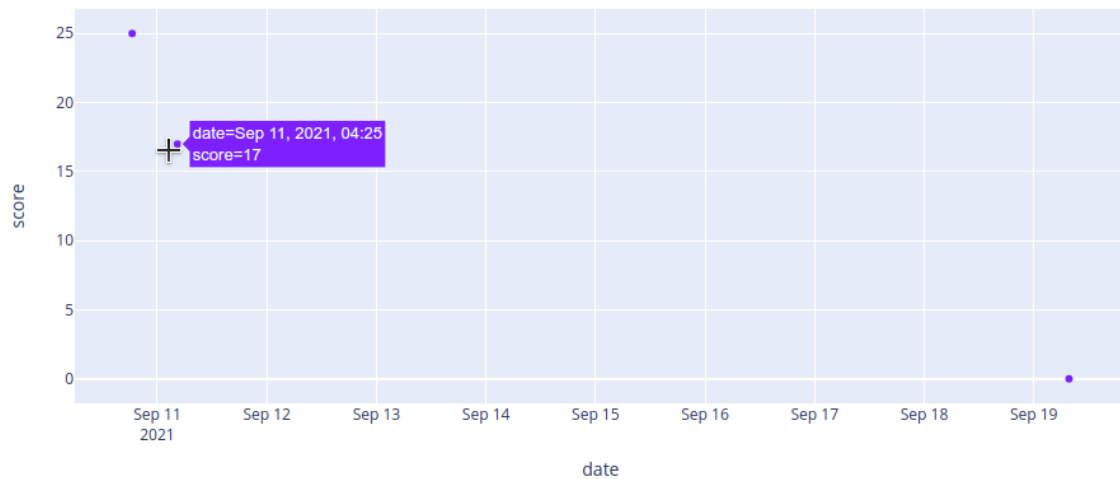


Figura 4.22: Gráfica de puntuación de eficacia

4.3.6. Despliegue

Para desplegar Dash en producción se ha añadido su contenedor oficial al *docker compose* del sistema Unibotics. Se indica el puerto por el cual va a ser lanzado y las dependencias entre servicios, en este caso entre Dash y Elasticsearch. El código es el siguiente:

```
1 dash:
2     image: unibotics
3     working_dir: /unibotics-webserver/unibotics_dash
4     restart: always
5     ports:
6         - "6123:6123"
7     container_name: "dash_unibotics"
8     command: gunicorn -w 1 -b :6123 app:server
9     environment:
10         - ELASTICSEARCH_HOST=elasticsearch
11     depends_on:
12         - elasticsearch
```

Capítulo 5

Conclusiones y trabajos futuros

En este último capítulo se detallan las principales conclusiones alcanzadas, así como las competencias adquiridas al realizar este TFG y los futuros trabajos que pueden extender la plataforma y las contribuciones actuales.

5.1. Conclusiones finales

El objetivo principal de este TFG se ha cumplido, ya que se ha conseguido integrar con éxito analíticas automáticas en Unibotics. Desde la propia herramienta se detecta el tipo de usuario que accede, de tal manera que es posible controlar y mostrar una visualización u otra dependiendo de este tipo de usuario. Analizando los subjetivos mencionados en el capítulo 2, llegamos a las siguientes conclusiones:

El subjetivo 1 se ha alcanzado con éxito. Se ha podido capturar y guardar las interacciones de los usuarios en la plataforma. Las sondas que se han recogido han sido: el inicio y fin de sesión de un usuario, entrada y salida de los ejercicios y las evaluaciones de eficacia y estilo del código de cada ejercicio. A través de código JavaScript, las sondas se capturan en el *frontend* y se envían al servidor, el cual se encarga de grabarlas en la base de datos.

5.2. COMPETENCIAS ADQUIRIDAS

El subjetivo 2 se ha conseguido con la utilización de la herramienta Elasticsearch. El servidor web de Django es el encargado de grabar las sondas en la base de datos no relacional de Elasticsearch. Para guardar las sondas se han creado previamente cuatro índices, los cuales se explican en la subsección 4.2.2.

El subjetivo 3 se ha logrado con la integración de las visualizaciones de la información en el servidor web gracias al entorno Dash. Ha permitido de una manera sencilla y potente la visualización de las sondas recogidas en Elasticsearch. Estas gráficas representan el uso de Unibotics de estudiantes reales desde mayo del 2021. Ello ha permitido a los administradores comprender mejor el uso real de la plataforma por parte de los estudiantes.

Se han añadido en varias gráficas el filtro de fechas y el filtro por usuario. Las primeras gráficas son las basadas en el número de registros y usuarios en la plataforma. Aquí se encuentran tres gráficas lineales que representan los registros, los registros acumulados y los usuarios activos cada día.

El siguiente bloque de gráficas enseña la actividad en la plataforma. En él se encuentran las gráficas de sesiones por día, tanto totales como por usuario único. Estas gráficas se pueden encontrar en formato lineal o en mapa de calor, como se describe en la subsección 4.3.2. También está la gráfica del tiempo total empleado en la plataforma cada día y su histograma. Para ver la actividad en cada ejercicio, se ha creado un histograma de las duraciones en ellos, con los filtros mencionados en la subsección 4.3.1.

Por último, la visualización de los metadatos, donde se halla la ubicación geográfica de los usuarios, el sistema operativo y navegador que utilizan para acceder a la plataforma.

5.2. Competencias adquiridas

Durante la realización del TFG he adquirido las siguientes competencias:

- Ampliado mis conocimientos sobre las tecnologías web, tanto el entorno de Django como HTML, CSS y JavaScript.

- Conocer cómo funciona una base de datos relacional en un proyecto real, en el caso de MySQL, y saber desplegar e integrar una nueva base de datos no relacional, Elasticsearch. Comprender la importancia de la información para monitorizar un servicio web.
- Entender las tecnologías de visualizaciones automáticas de la información gracias a Dash, la cual es una herramienta rápida y eficiente.
- Aprender a utilizar GitHub como repositorio donde desarrollar proyectos en equipo, haciendo uso de incidencias y parches.
- Trabajar en una plataforma que está en continuo desarrollo, donde se han creado nuevas funcionalidades. Trabajar en equipo con otros desarrolladores de áreas diferentes a la mejorada en este proyecto.

5.3. Trabajos futuros

En esta sección se proponen algunas futuras líneas para mejorar las analíticas automáticas:

- Enriquecer las sondas que se encuentran en Unibotics añadiendo nuevas o añadiendo nuevos campos a los índices creados para recabar más información sobre el uso de la plataforma. Actualmente las sondas de puntuación de estilo y de eficacia solo se encuentran en cuatro ejercicios, así que se podrían añadir a los nuevos ejercicios que sean creados. Para recoger nuevas sondas será necesario la creación de nuevos índices en Elasticsearch y nuevas visualizaciones automáticas en la aplicación de Dash.
- Incorporar a Elasticsearch medidas de rendimiento en el propio servidor web en producción. El servidor está desplegado en *Amazon Web Services* en producción por lo que se pueden añadir métricas de memoria ocupada o CPU consumida, entre otras. Luego se podrá correlar esas métricas con las sondas actuales, tales como el número de usuarios activos en la plataforma.
- En este TFG se han visualizado las sondas con los datos directamente recogidos, el siguiente paso que se podría hacer es correlacionar varias de las sondas recogidas y así enriquecer el conocimiento sobre el uso real que los estudiantes hacen de la plataforma y

sobre sus procesos de aprendizaje. Por ejemplo, saber si el tiempo que dedican en realizar un ejercicio afecta a la nota obtenida en el ejercicio. Esto nos permite hacer análisis estadísticos o investigaciones científicas, como el efecto de la *gamificación* en los procesos de aprendizaje.

Bibliografía

- [1] Juan González Gómez. *Introducción a las tecnologías web*. 2018. URL: <https://github.com/myTeachingURJC/2018-19-CSAAI/wiki/Introducci%C3%B3n-a-las-tecnolog%C3%ADas-web> (visitado 15-09-2021).
- [2] James Sanchez. *Laravel: ventajas del framework PHP de moda*. 2016. URL: <https://www.freelancer.es/community/articles/ventajas-del-framework-moda-laravel> (visitado 25-10-2021).
- [3] Carlos Travieso Merino. “Scratch”. En: *Crisalis* (2012). URL: <http://static.esla.com/img/cargadas/2267/Documentaci%C3%B3n%20Scratch.pdf> (visitado 24-11-2021).
- [4] Marcos Merino. “Seis aplicaciones gratuitas para aprender robótica y programación”. En: *Genbeta* (2020). URL: <https://www.genbeta.com/desarrollo/seis-aplicaciones-gratuitas-para-aprender-robotica-programacion> (visitado 15-09-2021).
- [5] *Why ROS?* 2021. URL: <https://www.ros.org/blog/why-ros/> (visitado 24-11-2021).
- [6] Raúl García Jerez. “Integración de Planificación Automática y ROS para el control autónomo de dos robots en el juego del Sokoban”. Tesis doct. Escuela Politécnica Superior. Universidad Carlos III de Madrid, 2014.
- [7] *CoppeliaSim*. URL: <https://www.coppeliarobotics.com/> (visitado 24-11-2021).
- [8] Marian Körber y col. *Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning*. 2021. arXiv: 2103.04616 [cs.RO].

- [9] Javier Velasco Seguido-Villegas. “Análisis y comparación de las principales plataformas de simulación robótica y su integración con ROS”. Sep. de 2019. URL: <http://oa.upm.es/56724/>.
- [10] *Gazebo*. URL: <http://gazebo.org/> (visitado 24-11-2021).
- [11] *Riders.ai*. URL: <https://riders.ai/about/courses/intro-to-robotics#aboutus/> (visitado 24-11-2021).
- [12] JdeRobot organization. *Robotics Academy*. URL: <http://jderobot.github.io/RoboticsAcademy/> (visitado 27-09-2021).
- [13] *Conceptos básicos de HTML*. URL: https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics (visitado 28-09-2021).
- [14] Uniwebsidad. *Etiquetas y atributos*. URL: <https://uniwebsidad.com/libros/xhtml/capitulo-2/etiquetas-y-atributos> (visitado 28-09-2021).
- [15] Juan González Gómez. *Sesión 2: HTML*. 2018. URL: <https://github.com/myTeachingURJC/2018-19-CSAAI/wiki/Sesi%C3%B3n-2:-HTML> (visitado 28-09-2021).
- [16] Juan González Gómez. *Sesión 3: CSS*. 2018. URL: <https://github.com/myTeachingURJC/2018-19-CSAAI/wiki/Sesi%C3%B3n-3:-CSS#reglas-de-aplicaci%C3%B3n-del-estilo> (visitado 28-09-2021).
- [17] Juan González Gómez. *Sesión 4: Introducción a Javascript*. 2018. URL: <https://github.com/myTeachingURJC/2018-19-CSAAI/wiki/Sesi%C3%B3n-4:-Introducci%C3%B3n-a-Javascript> (visitado 29-09-2021).
- [18] *JavaScript*. URL: <https://developer.mozilla.org/es/docs/Learn/JavaScript> (visitado 29-09-2021).
- [19] Django. *Modelos*. URL: <https://docs.djangoproject.com/en/2.2/topics/db/models/> (visitado 29-09-2021).
- [20] W3school. *Introducción a SQL*. URL: https://www.w3schools.com/sql/sql_intro.asp (visitado 29-09-2021).

BIBLIOGRAFÍA

- [21] Micho García. *Conceptos básicos de SQL*. 2013. URL: https://postgis.readthedocs.io/es/latest/conceptos-sql/conceptos_sql.html# (visitado 29-09-2021).
- [22] Victor Cuervo. *¿Qué es Elasticsearch?* 2019. URL: <https://www.arquitectoit.com/elasticsearch/que-es-elasticsearch/> (visitado 04-10-2021).
- [23] *¿Qué es Elasticsearch?* URL: <https://www.elastic.co/es/what-is/elasticsearch> (visitado 04-10-2021).
- [24] Victor Cuervo. *Conceptos Básicos Elasticsearch*. 2019. URL: <https://www.arquitectoit.com/elasticsearch/conceptos-basicos-elasticsearch/> (visitado 04-10-2021).
- [25] *Introduction to Dash*. URL: <https://dash.plotly.com/introduction> (visitado 04-10-2021).