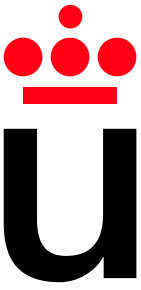

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Diseño y Desarrollo de Videojuegos

Curso 2021-2022

Trabajo Fin de Grado

AUTOMATIZACIÓN DE COMPETICIONES EN UNA PLATAFORMA WEB DE PROGRAMACIÓN DE ROBOTS

Autor: Raúl Fernández Ruíz
Tutores: Jose María Cañas Plaza y Daniel Palacios Alonso

Agradecimientos

A mi madre y hermano por apoyarme siempre, aunque la vida no nos lo ha puesto fácil siempre hemos salido adelante. Estoy seguro que no sería la persona que soy hoy sin vosotros, seguramente me habría rendido, por ello, os estaré siempre agradecido.

Quiero recordar a mis abuelos, que aunque fallecieron, sé que estarían orgullosos de lo que mi hermano y yo hemos logrado. Nunca os olvidaré.

También agradezco a mis tutores José María Cañas y Daniel Palacios Alonso por dejarme contribuir a este proyecto y por el apoyo prestado a lo largo del trabajo.

Por último, agradecer a David Valladares su amabilidad y ayuda durante estos meses.

Resumen

La ludificación es una metodología de enseñanza en auge en los últimos años que favorece la retroalimentación e implicación de los estudiantes. Su correcta implementación en un programa educativo puede fomentar la motivación e interés del alumnado de manera significativa.

La automatización de tareas permite eliminar la posibilidad del fallo humano y aumentar la velocidad de realización en gran variedad de casos. Integrar la automatización en el entorno educativo ofrece grandes ventajas como la agilidad en los procesos o la obtención de retroalimentación.

Este TFG se ha desarrollado en la plataforma web Unibotics, centrada en la enseñanza de robótica a estudiantes universitarios. Hemos creado la infraestructura necesaria para la realización de torneos entre usuarios, transmitidos vía Twitch, que ofrecen resultados en tiempo real. Para agilizar la realización de los mismos hemos automatizado este proceso permitiendo a cualquier desarrollador de Unibotics retransmitir sus propios torneos.

Se han empleado las siguientes herramientas: Django como framework para el desarrollo web, ya que ofrece los métodos necesarios para la gestión del lado del cliente y servidor. La automatización se ha desarrollado con Selenium por su flexibilidad y facilidad de uso. También se han empleado (en menor medida) subprocesos de Python para el manejo de Docker y *WebSockets* para establecer la conexión con el programa OBS.

Índice de contenidos

Índice de figuras **XI**

1. Introducción **1**
1.1. Ludificación en el aprendizaje 1
1.2. Tecnologías web en el aprendizaje de programación 3
1.2.1. Plataformas de programación 3
1.2.2. Tecnologías web *frontend* y *backend* 5
1.3. Robótica 6
1.4. Tecnologías web en el aprendizaje de robótica 9
1.4.1. En educación pre-universitaria 9
1.4.2. En ingeniería 10
1.5. Herramientas de automatización 13

2. Objetivos y planificación del TFG **15**
2.1. Objetivos 15
2.2. Metodología 16
2.3. Plan de trabajo 16

3. Herramientas **18**
3.1. Django 18
3.2. ElasticSearch 20
3.3. Tecnologías web del lado del cliente 21
3.4. Selenium 23
3.5. Docker 25
3.6. Python 26
3.7. Twitch 27
3.8. Tecnologías WebSocket 28
3.9. Unibotics 29

4. Automatización de torneos en Unibotics **33**
4.1. Diseño de los torneos y arquitectura 33
4.2. Infraestructura de torneos 35
4.2.1. Maestro de ceremonias 35

4.2.2. Vistas personalizadas dependiendo del rol 36
4.2.3. Plantillas dinámicas 38
4.2.4. Obtención de los códigos fuente de cada usuario 39
4.2.5. Enviar al servidor los resultados de la ejecución del código de un usuario 40
4.3. Automatización 40
4.3.1. Lanzamiento de contenedores Docker 41
4.3.2. Conexión WebSocket con OBS 41
4.3.3. Acceso a la página web como maestro de ceremonias . . . 43
4.3.4. Establecer conexión con el ejercicio 45
4.3.5. Ejecución del torneo 46
4.3.6. Detener los contenedores Docker y servidor 47
4.4. Gestión de torneos 48
4.4.1. Creación de un torneo 48
4.4.2. Editar un torneo 49
4.4.3. Registro de un usuario en un torneo 50
4.4.4. Retransmisión en directo 51
4.5. Validación experimental 52

5. Conclusiones **55**
5.1. Conclusiones 55
5.2. Trabajos futuros 56

Bibliografía **56**

Índice de figuras

1.1. Mecánicas, dinámicas y estética. 2
1.2. Nivel de CodeCombat (Fuente) 4
1.3. Tabla de seguimiento en Codemonkey (Fuente) 4
1.4. Editor de Scratch (Fuente) 5
1.5. Minijuego en Code.org (Fuente) 5
1.6. Player Project (Fuente) 7
1.7. RT-middleware (Fuente) 8
1.8. Ejercicios en Kibotics (Fuente) 10
1.9. Ejemplo de proyecto en Riders.ai (Fuente) 11
1.10. Tabla de puntuaciones en RiderAI (Fuente) 11
1.11. Control remoto de robots en The Construct (Fuente) 12
1.12. Ejercicio sigue lineas en Unibotics (Fuente) 13

3.1. Ejemplo índice invertido (Fuente) 21
3.2. Ejemplo de un documento HTML 22
3.3. Arquitectura Selenium WebDriver 25
3.4. Arquitectura Docker 26
3.5. Instalación de paquetes en PyCharm 27
3.6. Api de Twitch en Unibotics (Fuente) 28
3.7. Arquitectura de Unibotics (Fuente) 30
3.8. Ejercicios en Unibotics (Fuente) 30
3.9. Ejercicios en Unibotics (Fuente) 31

4.1. Arquitectura de los torneos automáticos 34
4.2. Comunicación entre los componentes del sistema de torneos automáticos 35
4.3. Página de administración Django 36
4.4. Cambio de rol 36
4.5. Urls en Django 37
4.6. Vista inicial que se muestra al maestro de ceremonias 38
4.7. Tareas automatizadas 41
4.8. Error al iniciar el programa OBS desde el .exe 42
4.9. Configuración servidor OBS 42

4.10. Funcionamiento básico de Selenium 43
4.11. Inicio de sesión en Unibotics 44
4.12. Vista inicial del maestro de ceremonias 44
4.13. Selección automatizada de ejercicio 45
4.14. Desplegable del ejercicio 45
4.15. Botones de conexión en Unibotics 46
4.16. Creación de un nuevo torneo 48
4.17. Vista para la edición de torneos 49
4.18. Vista para la inscripción en un torneo 50
4.19. Retransmisión en directo en la página de Unibotics 51
4.20. Vista de un torneo finalizado 52
4.21. Inicio de emisión de directo OBS 52
4.22. Inicio de sesión con el maestro de ceremonias en D3 53
4.23. Obtención de concursantes del torneo 53
4.24. Ejecución del código fuente de un usuario 54
4.25. Página de retransmisión en vivo 54

1 Introducción

El tema central de este Trabajo Fin de Grado (TFG) es el diseño de torneos para la plataforma web Unibotics utilizada en el grado de Ingeniería de Robótica y Software y en el Máster en Visión Artificial de la URJC. En estos torneos los alumnos deberán resolver un ejercicio de programación. Las puntuaciones obtenidas se añadirán a una tabla de clasificación. Este tipo de formación cuenta con una componente competitiva que mejorará el rendimiento de los estudiantes. Todo el proceso del torneo estará automatizado.

1.1. Ludificación en el aprendizaje

La ludificación, o *gamification* en inglés, es una metodología de aprendizaje basada en la aplicación de mecánicas de juego en entornos no lúdicos que permite mejorar la motivación y la cooperación dentro del ámbito educativo gracias a la interacción del sujeto con los elementos gamificados. En los últimos años se ha cuestionado la calidad de la enseñanza en los diferentes niveles educativos, muchos de ellos basados en proporcionar gran cantidad de información con el objetivo de que los alumnos superen una prueba que verifique que han adquirido esos conocimientos. En la práctica esas pruebas premian la buena memoria sin necesidad de entender los conceptos e, incluso, la mayoría de los alumnos olvidan gran parte de lo aprendido al cabo de pocas semanas, ya que sólo han memorizado lo necesario para superar la prueba. La ludificación debe plantearse como una metodología que despierte el interés del alumnado por aprender. Algunas instituciones lo consideran únicamente una forma "de hacer divertido el aprendizaje", sin embargo, la finalidad de este método es replantear la manera en que se transmiten conocimientos al alumno. (Parente, D., 2016)

Para ludificar el aprendizaje es necesario añadir mecánicas y elementos de juego que permitan a los alumnos aprender siguiendo una curva de dificultad. Esto requiere implementar mecánicas de retroalimentación que permitan al usuario obtener resultados sobre las acciones que realiza y mecánicas de progresión centradas en la obtención de nuevas habilidades usando las aprendidas previamente. Al igual que en un videojuego, el valor de las mecánicas depende de cómo permitimos al usuario aplicarlas.

Existen tres elementos clave para incluir en el sistema: mecánicas, dinámicas y estética. El análisis *MDA (Mechanics, Dynamics y Aesthetics)*, que proviene del mundo del videojuego, se basa en ellos (ver Figura 1.1). Los desarrolladores emplean este método para realizar un análisis de su videojuego durante el desarrollo, permitiéndoles tener una visión general de lo que ofrecen al jugador. (Hunicke, R., LeBlanc, M., & Zubek, R., 2004)

Los tres elementos son:

- **Mecánicas**: reglas que permiten a los alumnos enfrentarse a los diferentes retos dando sentido al juego. Los más conocidos son: sistemas y tablas de puntuación, recompensas, logros y retos en equipo.
- **Dinámicas**: motivan e implican al alumno en la realización de una tarea al convertirla en una necesidad/objetivo. Las más típicas son historia o inmersión.
- **Estéticas**: interacción entre las mecánicas y dinámicas creadas, entendido como la sensación que experimenta el alumno al adentrarse en un entorno gamificado. Existen diversos tipos: desafío (cuando sentimos que nos enfrentamos a un reto difícil pero que con esfuerzo, podremos superar), competición (superar a otros usuarios), etc.

Figura 1.1: Mecánicas, dinámicas y estética.

Para ludificar un sistema hay una serie de pautas que se deben seguir:

- **Entender al público objetivo y contexto:** en este caso estamos en un entorno universitario, es un público joven de 18 hasta unos 22-23 años. Esto nos puede ayudar a identificar los "*pain points*", factores que pueden llevar al estudiante a no completar el programa. En un entorno universitario los puntos más frecuentes suelen ser falta de motivación o de conocimientos.

- **Objetivos del aprendizaje:** es el motivo por el que los alumnos están ahí, aumentar su conocimiento en al ámbito de la robótica. Estos conocimientos serán evaluados mediante una prueba escrita.

- **Estructurando la experiencia:** definimos cómo se estructurará el proceso de aprendizaje. En primer lugar se estudian unos conocimientos teóricos en el aula y posteriormente, se celebra un torneo basado en un ejercicio de Unibotics relacionado con estos conocimientos.

- **Identificar los recursos:** una vez definida la estructura, crearemos sistemas para medir el progreso de los estudiantes, por ejemplo, ver la puntuación que ha obtenido el alumno en los diferentes torneos para tener una idea general de lo aprendido.

En resumen, el objetivo de este trabajo es el desarrollo de una herramienta formativa que, mediante el juego, facilite el aprendizaje y permita mejorar las habilidades de programación de los estudiantes de robótica de la URJC.

1.2. Tecnologías web en el aprendizaje de programación

Desde su origen en 1990, cuando los trabajadores del CERN Tim Berners-Lee y Robert Cailliau crearon la primera página de Internet, la tecnología web ha experimentado una enorme evolución, Actualmente cualquier dispositivo tiene acceso a internet.

El avance de la tecnología web ha facilitado la ludificación de las páginas web permitiendo mostrar el contenido de manera atractiva para los usuarios.

1.2.1. Plataformas de programación

- **CodeCombat**[1]: página web destinada a introducir a los jóvenes en el mundo de la programación. Aprenden lenguajes como Python, C++ o JavaScript. El código desarrollado se traduce en acciones en un mundo de aventura (ver Figura 1.2), tiene 110 niveles que van aumentando en dificultad, además, podemos escoger el lenguaje de programación y cuenta con varios

[1] https://codecombat.com/

elementos que ayudan al usuario a sentirse en un videojuego, como personalización de personajes con diferentes estadísticas. Todo esto sin necesidad de descargarnos ningún programa, todo desde una página web.

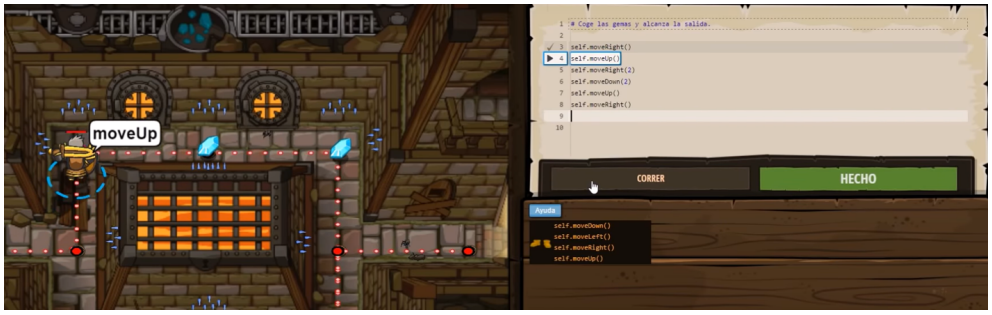

Figura 1.2: Nivel de CodeCombat (Fuente)

- CodeMonkey[2]: la finalidad es la misma que CodeCombat, pero en este caso va dirigido a un público más joven. Enfocado a ser utilizado en clase, cuenta con herramientas que permiten seguir el progreso de los alumnos o ejercicios completados (ver Figura 1.3).

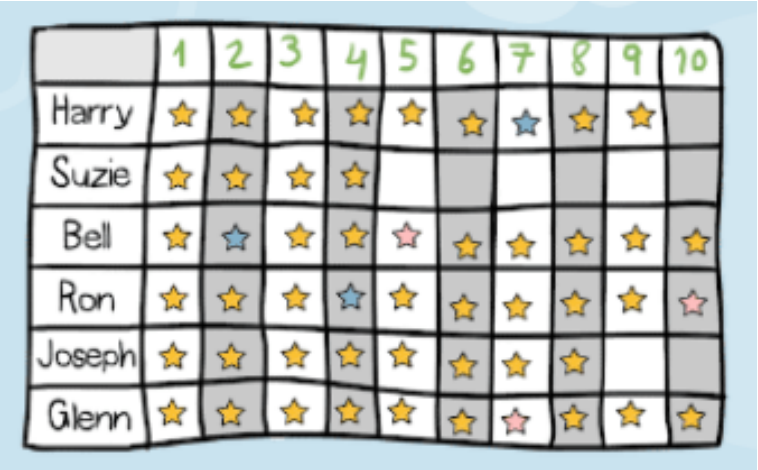

Figura 1.3: Tabla de seguimiento en Codemonkey (Fuente)

- **Scratch**[3]: lanzada en 2003 con el objetivo de introducir a los niños en el mundo de la programación, ayuda a desarrollar capacidades como el pensamiento computacional sin necesidad de profundizar en el código. Se basa en combinar diferentes bloques que pueden ser eventos, sonidos u operadores (ver Figura 1.4), permitiendo crear diferentes proyectos basados en una historia, música, puzzle, etc.

[2]https://www.codemonkey.com/
[3]https://scratch.mit.edu/

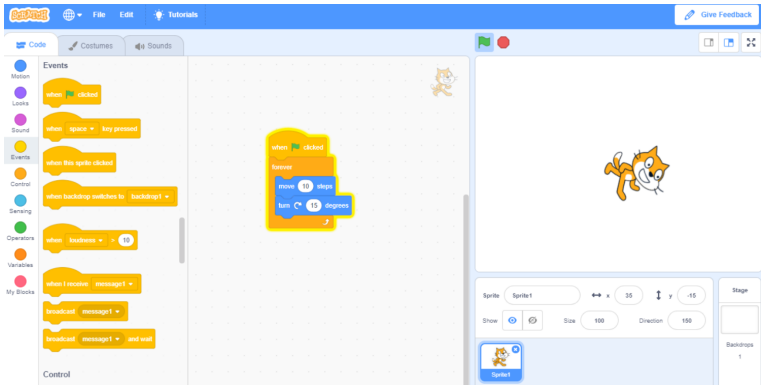

Figura 1.4: Editor de Scratch (Fuente)

- **Code.org**[4]: al igual que Scratch, está enfocada a enseñar programación a niños con ejercicios basados en bloques, pero es más guiado que Scratch. Existen diferentes niveles de dificultad, además cuenta con juegos en español e inglés. Los cursos permiten al alumno aprender cómo expresar sus ideas y resolver problemas con un lenguaje de programación. Se compone de diferentes juegos basados en Minecraft o Angry Birds para ludificar el aprendizaje traduciendo los movimientos del estudiante en acciones del juego (ver Figura 1.5).

Figura 1.5: Minijuego en Code.org (Fuente)

1.2.2. Tecnologías web *frontend* y *backend*

El desarrollo de las páginas web conlleva el uso de tecnologías que permitan crear interfaces de usuario y establecer el paso de mensajes con el servidor, además de gestionar las peticiones del cliente y generar las respuestas correspondientes. Existen tecnologías del lado del cliente (*frontend*) y del lado del servidor (*backend*). Algunos ejemplos de tecnología del servidor son:

[4]https://code.org/

- **Django**[5]: entorno de alto nivel basado en Python, facilita el rápido desarrollo de sitios web con su modelo vista-modelo-plantilla. Otras ventajas son la seguridad (uso de csrf tokens) y la escalabilidad ya que cada parte de la aplicación es independiente de otras.
- **PHP**[6]: lenguaje usado para el paso de información entre un sitio web y el servidor, permite trabajar con bases de datos como MySQL y facilita la modificación y escritura de datos en las mismas.
- **Laravel**[7]: basado en PHP, arquitectura Modelo-Vista-Controlador. Gran facilidad para el manejo de bases de datos, permite modificar bases de datos existentes sin necesidad de borrarlas.
- **Node.js**[8]: entorno de ejecución en tiempo real que permite ejecutar programas escritos en JavaScript. Destaca en la creación de aplicaciones de red rápidas que manejan gran cantidad de conexiones de manera simultánea permitiendo un buen nivel de rendimiento.

Tecnologías del lado del cliente:

- **HyperText Markup Language(HTML)**: define el significado y estructura del contenido web, utiliza etiquetas para diferenciar los múltiples elementos que componen la página web.
- **JavaScript**: lenguaje de programación que permite dotar de interactividad a nuestra página web, desde animar contenidos y respuesta a eventos, hasta paso de mensajes con el servidor.
- **CSS**: Cascade Style Sheets, lenguaje declarativo que controla el aspecto visual de las páginas web.
- Bootstrap: entorno formado por archivos CSS y JavaScript, es una herramienta que permite estilizar elementos de manera sencilla y ofrece interactividad como menús de navegación, tablas, filtrados...

1.3. Robótica

La robótica esta presente en nuestro entorno diario en gran variedad de ámbitos:

- Tareas del hogar desde Roombas hasta la domótica.

[5]https://www.djangoproject.com/
[6]https://www.php.net/manual/es/intro-whatis.php
[7]https://laravel.com/
[8]https://nodejs.org/es/

- Conducción de coches automáticos, sistemas de aparcado o herramientas que faciliten la conducción.
- Logística, uso de robots para transportar cargas en los almacenes o realizar entregas.
- Medicina, dispositivos como el robot DaVinci que permite a los cirujanos realizar cortes complejos con gran precisión.

Los robots constan de dos partes: la parte física (hardware) compuesta por sensores, actuadores y computadores y el software, donde reside la inteligencia del robot, definiendo su comportamiento y funcionamiento. La importancia de la programación del robot ha aumentado en los últimos años a medida que las tareas se ha vuelto más complejas. El Roomba es un ejemplo claro de la importancia del software en un robot, si el software está mal diseñado, el robot dejará zonas de la casa sin limpiar o se golpeará con los muebles.

La robótica ha avanzado mucho gracias a herramientas que facilitan el desarrollo del software del robot, entornos donde se simula un modelo antes de pasar a las fases de construcción, herramientas de código abierto... . En primer lugar hablaremos de las herramientas conocidas como *middleware* robótico, que permite al desarrollador centrarse en el software del robot y abstraerse de las complejidades del hardware:

- **Player Project**[9]: es uno de los *middlewares* de código abierto más populares, formado por dos proyectos. El primero es *Player* que funciona como la capa de abstracción del robot, el segundo es *Stage*, un entorno de simulación 2D (ver Figura 1.6), que soporta varios robots al mismo tiempo y permite al usuario manejarlo.

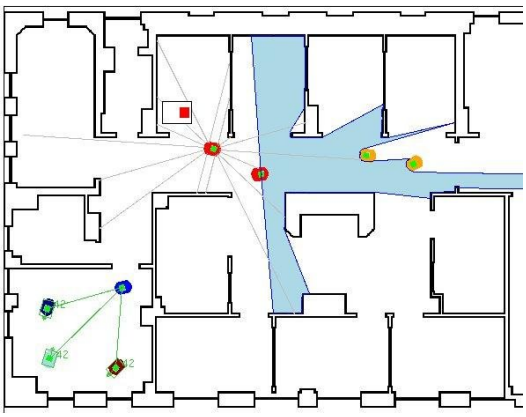

Figura 1.6: Player Project (Fuente)

[9]http://playerstage.sourceforge.net/

- **RT-middleware**[10]: se caracteriza porque elementos como los actuadores del robot son considerados como componentes-RT y el sistema robótico se construye conectando esos componentes-RT (ver Figura 1.7). Esta arquitectura distribuida del robot permite al desarrollador reutilizar componentes, cada componente cuenta con un puerto para comunicarse con las otras piezas y permite a los que son de un mismo tipo, conectarse entre ellos. Para el control de cada componente se cuenta con una máquina de estados donde los más usuales son activo, inactivo o error.

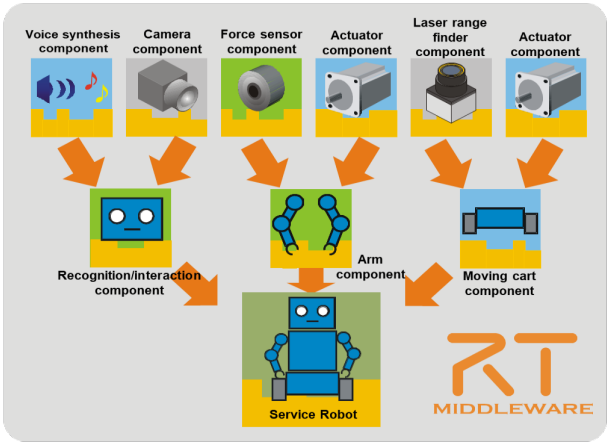

Figura 1.7: RT-middleware (Fuente)

- **ROS (Robotics Operating System)**[11]: es el *middleware* más extendido por la gran variedad de herramientas que ofrece: abstracción del hardware, control del dispositivo a un bajo nivel, funcionalidades comunes en el desarrollo de un robot ya implementadas, paso de mensajes entre procesos y manejo de paquetes. Aunque ROS fue desarrollado para UNIX, su versión actual ROS2 es multiplataforma soportando gran variedad de sistemas operativos.

  Debido a que ROS se usa en Unibotics para el desarrollo de los robots de los ejercicios, explicaremos en detalle su funcionamiento. Se basa en un modelo de grafo formado por nodos donde cada uno representa un proceso y cuenta con un nombre único. Para el paso de información entre nodos se usan tópicos, un nodo debe indicar a qué tópico mandará la información y los nodos que deseen información de ese tópico, se deben suscribir a él, pero los nodos no saben qué nodo está mandando la información al tópico.

Como hemos visto hay *middlewares* que integran un entorno donde poder simular nuestro robot como Player Project, también existen herramientas especializadas, ya que simular el robot en un entorno puede ser clave para detectar

[10] https://www.openrtm.org/openrtm/en/doc/aboutopenrtm/rtmiddleware
[11] https://www.ros.org/

errores de diseño o seguridad solventándolos antes de pasar a fases más avanzadas, los más conocidos son:

- **Gazebo**: simulador robótico 3D más extendido de código abierto. Permite usar múltiples motores de físicas de alto rendimiento como ODE, Bullet o Simbody. Deja modelar sensores que perciben el escenario como cámaras o láseres y cuenta con múltiples herramientas ya desarrolladas gracias al apoyo que ha recibido el proyecto. Se encuentra en estado de mantenimiento y no de desarrollo haciéndolo un producto más estable, además permite una integración fácil con ROS.
- **OpenRAVE**: especializado en ofrecer un entorno para el desarrollo de algoritmos de búsquedas de caminos, la simulación se centra en la geometría y movimiento del robot para crearlos.
- **Webots**: simulador 3D de código abierto muy usado en el sector industrial, ofrece una gran cantidad de robots, actuadores y sensores todos modificables. Además nos permite importar modelos desde 3D CAD.

1.4. Tecnologías web en el aprendizaje de robótica

Gracias al desarrollo de las tecnologías web existen múltiples plataformas que se pueden emplear para el aprendizaje de distintas disciplinas, en nuestro caso nos centraremos en las dirigidas a la enseñanza de robótica. Lo dividiremos en dos grupos, el primero dirigido a un público joven para introducirlo en el mundo de la robótica, otro grupo adulto, que cuenta con conocimientos previos.

1.4.1. En educación pre-universitaria

Veremos diferentes plataformas web dirigidas a la enseñanza de robótica a los más jóvenes:

- **OpenRoberta**[12]: permite programar diferentes modelos de robots, el editor se utiliza de manera online sin necesidad de instalar software adicional para su funcionamiento. Utiliza su lenguaje de bloques NEPO, similar a Scratch. La plataforma incluye un simulador donde visualizar el resultado del ejercicio o descargar nuestro código en el robot, un ejemplo es el EV3 de *LEGO MINDSTORMS*.
- **Simulador Miranda**[13]: permite a los alumnos usar robots existentes o creados por ellos mismos. Se pueden crear diferentes desafíos y compartir-

[12]https://lab.open-roberta.org/
[13]https://www.roboticavirtual.com/miranda

los con la comunidad, se basa en Python y Scratch. Por último, Miranda ofrece torneos y ligas donde los alumnos pueden participar, por ejemplo la Liga CoderZ Junior, donde los alumnos forman equipos para superar los diferentes desafíos a la vez que desarrollan su capacidad para resolver problemas.

- **GearsBot**[14]: ofrece diferentes retos que los alumnos deben de superar. Podemos desarrollar el código usando Scratch o Python y crear nuestros propios ejercicios.

- **Kibotics**[15]: plataforma dedicada a la enseñanza de los principios básicos de programación de robots a niños y adolescentes (ver Figura 1.8). Ofrece dos tipos de eventos, torneos de *ranking* donde los usuarios desarrollan su código y, en base a su puntuación, obtienen una posición en la tabla de clasificación. En los torneos de enfrentamiento dos usuarios compiten de manera directa, el que obtiene mejor puntuación pasa a la siguiente fase. La página se presenta de manera amigable para resultar más llamativa, tratando de presentar los conocimientos de una manera divertida.

Figura 1.8: Ejercicios en Kibotics (Fuente)

1.4.2. En ingeniería

Existen plataformas web centradas en la enseñanza de robótica para un público con conocimientos previos. En el primer punto vimos como la ludificación varía dependiendo del público objetivo, en las plataformas dirigidas a un público más joven se fomenta en mayor medida el aspecto lúdico y cooperativo, sin embargo, cuando se orienta a un público experto se intensifica el aspecto competitivo.

- **Riders.ai**[16]: página de enseñanza online sobre robótica. Es una aplicación *freemium*, donde el contenido gratuito nos ofrece participar en diferentes torneos mientras que los cursos se dividen en diferentes apartados, siendo

[14] https://gears.aposteriori.com.sg/
[15] https://kibotics.org/
[16] https://riders.ai/about

el primero gratis y el resto de pago. Esta página va dirigida a un público más amplio que Unibotics, ya que algunos de los premios del torneo son entrevistas de trabajo con grupos o empresas.

Nos centraremos en el apartado de torneos viendo cómo plantean la estructura, actualización de la información, etc. Lo primero que llama la atención al entrar a un torneo, es la división de la información en tres apartados: clasificación, torneo y reglas del mismo. Es un proyecto muy avanzado, cuenta con un editor de código y simulador Gazebo ya integrado en el navegador sin necesidad de instalaciones adicionales (ver Figura 1.9), además de tener un diseño cuidado y homogéneo.

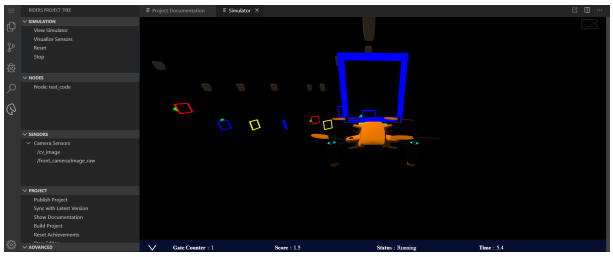

Figura 1.9: Ejemplo de proyecto en Riders.ai (Fuente)

Otro apartado de interés es la tabla de clasificación, que pretendemos actualizar en tiempo real. Sin embargo, al observar el código fuente de la página, se puede apreciar que no actualiza en tiempo real lo cual es comprensible teniendo en cuenta que la página tiene muchos usuarios (ver Figura 1.10). Antes de continuar vamos describir la estructura de los torneos, el torneo permanece abierto todo el año, sin embargo el premio se otorga a los usuarios que tengan el puesto más alto en las listas en los primeros seis meses del torneo. Los otros seis meses permanecen abiertos para que los usuarios puedan mejorar sus puntuaciones.

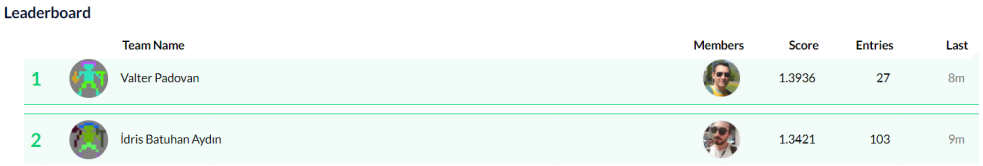

Figura 1.10: Tabla de puntuaciones en RiderAI (Fuente)

Durante el torneo no hay contacto entre los usuarios y la organización, nuestro objetivo en Unibotics es retransmitir el torneo en vivo fomentando la interacción entre usuarios y plataforma.

- **The Construct**[17]: página web enfocada a la enseñanza de robótica siendo

[17]https://www.theconstructsim.com/

también *freemium* con un plan de pago para obtener todas las mejoras y accesos a cursos posibles. En este caso la página no ofrece torneos donde los usuarios pueden participar, se centran más en el aspecto formativo contando con nuevo contenido semanal. Respecto al editor de proyectos es muy similar al anterior teniendo todo integrado en el navegador.

Un aspecto innovador de esta página es la posibilidad de probar códigos desarrollados por los usuarios en robots reales de manera *online*, reservando una hora en la agenda, el usuario puede disponer de un robot para probar su proyecto (ver Figura 1.11).

Figura 1.11: Control remoto de robots en The Construct (Fuente)

- Unibotics[18]: dirigida a universitarios con conocimientos más avanzados, completamente gratuita. Ofrece gran variedad de ejercicios, desde conducción autónoma como el "sigue lineas" hasta visión artificial. En cada ejercicio tenemos un apartado donde se explica el conocimiento teórico que se debe aplicar, esto permite al alumno la resolución del ejercicio y la posibilidad de profundizar en el tema, además de contar con un foro de dudas donde compartir soluciones y resultados.

  La página permite usar gran variedad de software dedicado a la programación de robots como Gazebo. Con un único comando de instalación, nos bajamos la imagen Docker necesaria para establecer la conexión con el ejercicio y mediante el uso de WebSockets, obtenemos una vista del simulador donde se encuentra el robot, permitiendo visualizar nuestros resultados de manera atractiva (ver Figura 1.12). Esta plataforma se emplea en varias asignaturas del grado en robótica de la Universidad Rey Juan Carlos, además de permitir el desarrollo de TFGs y TFMs añadiendo nuevas funcionalidades en la plataforma.

[18]https://unibotics.org/

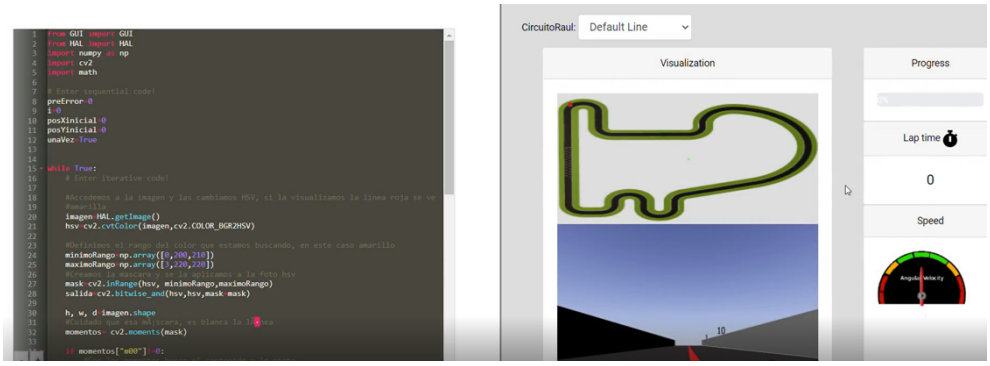

Figura 1.12: Ejercicio sigue lineas en Unibotics (Fuente)

1.5. Herramientas de automatización

Las herramientas de automatización se emplean en labores de ejecución de pruebas y para automatizar tareas repetitivas, pero necesarias. Por ejemplo, si en un proyecto varias personas comparten repositorio, es fácil que modifiquen el mismo archivo de código y que a la hora de combinarlos surjan conflictos, por ello existen herramientas automatizadas que se encargan de esta labor que puede ser difícil en proyectos grandes:

- **Integración continua**: permite validar de manera automática el código mediante pruebas unitarias y de integración, tanto el código añadido como el resto de partes del proyecto.
- **Distribución continua**: el código que ha superado la fase de integración se pública de manera automática en un entorno de producción, permitiendo que el código del proyecto siempre esté disponible para su integración.

Los ejemplos anteriores sirven para casos en los que queremos unir códigos de varias personas de manera sencilla y garantizar su correcto funcionamiento. En nuestro caso nos debemos centrar en herramientas que permitan automatizar páginas web con dos finalidades, la primera garantizar que la aplicación sigue funcionando al añadir cambios y segundo automatizar tareas repetitivas. Existen diversos *frameworks* para esto:

- **Selenium**[19]: nos permite automatizar aplicaciones web para labores de pruebas y tareas de administración. Su principal ventaja es que cuenta con varias versiones del mismo software orientado a diferentes propósitos como reproducción de errores, automatización de tareas y pruebas...

[19]https://www.selenium.dev/

- **CasperJS**[20]: empleado para *scripting* y *testing*, de acceso sencillo al usar como JavaScript como lenguaje.
- **Screenster**[21]: herramienta para pruebas de interfaz de usuario sin necesidad de programar. Ofrece una interfaz para diseñar las pruebas que compara el resultado que deberíamos obtener en la *UI*, con lo que sucede durante la prueba. Al final tenemos una serie de capturas con todo lo sucedido durante el proceso

[20] https://www.casperjs.org/
[21] https://screenster.io/

2

Objetivos y planificación del TFG

Se desarrollará la infraestructura software necesaria para la gestión de los torneos (creación, registro...) y la automatización de los mismos (control del torneo y emisión en directo).

2.1. Objetivos

El objetivo general de este TFG es la incorporación de ludificación automatizada a un entorno educativo web donde los alumnos puedan desarrollar sus habilidades de programación de robots y reforzar los conocimientos adquiridos en el aula. Los participantes competirán para resolver un ejercicio robótico de la forma más eficiente. El torneo se retransmitirá en directo lo cual permitirá ver cómo se van probando los diferentes códigos en vivo.

1. Realización de torneos: aplicar los cambios necesarios en las bases de datos para la creación del usuario instrumental "maestro de ceremonias" y dotándole de las funciones necesarias, por ejemplo, poder obtener códigos de otros usuarios.

2. Automatización de los torneos empleando Selenium y Docker.

3. Gestión de torneos: implementación tanto en su lado *backend* como en su lado *frontend* (creación de plantillas Django, accesos a las bases de datos, eventos...).

4. Validación: realización de un torneo de prueba en producción donde se verifique su correcto funcionamiento.

2.2. Metodología

Una vez estudiadas todas las tecnologías necesarias se comenzó con el desarrollo software. Se mantuvieron reuniones semanales vía Teams, en las que se establecían los objetivos a cumplir esa semana. Se creó un blog de desarrollo en Github [1], donde se anotaban los avances en el proyecto.

El proyecto se ha desarrollado en el repositorio de Unibotics siendo la metodología la siguiente: se creaba una incidencia con la funcionalidad a desarrollar y se generaba una rama (*branch*) donde se implementaba. Una vez terminada se crea un parche (*pull request*) para fusionarlo con la rama *master* que debía ser revisado por el *committer* del equipo.

En Unibotics existen tres despliegues: D1 (desarrollo local), D2 (ejecución en un ordenador remoto de la URJC) y D3 (en producción, nuestro código se fusiona con la rama *master*, siendo visible en la página oficial de Unibotics). Primero se desarrollaba en D1, a continuación se pasaba a D2, el cual es muy similar a D3 con la diferencia que en D2 usamos una base de datos de pruebas. Y por último abríamos un parche para fusionar nuestro código con la rama *master*, pasando nuestro código al despliegue en producción permitiendo el acceso a los códigos de usuarios reales almacenados en los servidores de Amazon (*Amazon Web Services - AWS*).

Por último, en la plataforma Slack hay un grupo de desarrolladores de Unibotics que permite consultar dudas a lo largo de la semana.

2.3. Plan de trabajo

Durante el primer mes se profundizó en las tecnologías necesarias para el desarrollo del código, ya fueran nuevas (Python, ElasticSearch, Selenium, Django, Docker y Python) o conocidas (HTML y JavaScript). También se estudió el código fuente de Unibotics.

1. Estudio de las tecnologías necesarias, siendo el orden:
    - Python como lenguaje base de Django.

[1] https://github.com/RoboticsLabURJC/2022-tfg-raul-fernandez

- Docker para entender el lanzamiento y terminación de los contenedores donde se encontraban las bases de datos de ElasticSearch.
- Selenium para la automatización de tareas.
- Django como entorno para todo el desarrollo web.
- ElasticSearch para almacenar todos los datos de los torneos, puntuaciones, usuarios registrados.
- HTML/JavaScript: HTML para el diseño de las plantillas Django que se muestran a los usuarios y JavaScript para el paso de mensajes.

2. Estudio del código fuente de Unibotics: es necesario entender el funcionamiento de Django con su modelo de Urls-Vistas-Plantilla y con las bases de datos concretas utilizadas en Unibotics, tanto relacionales (para usuarios y ejercicios) como no relacionales (ElasticSearch para eventos circunstanciales).

3. Desarrollo web

   - Creación del maestro de ceremonias.
   - Plantilla dinámica de torneos, gestión de torneos (creación, edición, etc).
   - Poder pedir códigos fuente de los diferentes participantes en el torneo.
   - Página de resultados en vivo durante el torneo.
   - Páginas de los torneos (inscripción, plantillas Django).
   - Paso de mensajes con el servidor con la respuesta correspondiente y escritura/lectura/modificación de la base de datos en ElasticSearch.

4. Automatización

   - Lanzamiento de los contenedores Docker y Servidor.
   - Registrarse con el maestro de ceremonias y cargar el torneo correspondiente.
   - Usar los códigos de los usuarios registrados en el torneo y enviarlos al servidor.
   - Retransmisión en Twitch usando un WebSocket con OBS.

3

Herramientas

En este apartado introduciremos brevemente las tecnologías utilizadas en este trabajo tanto las orientadas al cliente web, como al servidor web. Asimismo, herramientas de automatización y, finalmente, Docker.

3.1. Django

Framework de alto nivel para el desarrollo de servidores web [1]. En el TFG se ha utilizado la versión 2.2.2 para gestionar las peticiones del usuario en el lado del servidor y desarrollo de las plantillas HTML necesarias. Proporciona funcionalidades ya hechas permitiendo al desarrollador centrarse en su aplicación. Al ser un *framework* muy popular, cuenta con múltiples foros donde consultar dudas o problemas y permite construir una aplicación con las siguientes características:

- **Completo**: ofrece prácticamente todo lo que el desarrollador necesita para realizar un desarrollo web, evitando tener que combinar varios *frameworks* en el proyecto que puedan causar conflictos.
- **Versátil**: permite construir gran variedad de sitios web, tanto redes sociales, páginas de contacto, etc. Además soporta una gran variedad de tecnologías del lado del cliente y del servidor, también puede devolver mensajes en diferentes formatos.
- **Seguro**: ofrece maneras robustas de proteger datos confidenciales del cliente

[1] https://www.djangoproject.com/

como contraseñas o número de contacto. Por ejemplo, las contraseñas se almacenan en un *hash* de contraseñas. Además, protege contra diferentes tipos de ataques como inyección SQL, falsificación de solicitudes, etc.

- **Escalable**: Django utiliza una arquitectura distribuida denominada *shared-nothing*, donde cada parte es independiente de las otras permitiendo remplazarlas sin complicaciones.
- **Mantenible**: se basa en en el principio "*Don't Repeat Yourself (DRY)*" para evitar código repetido, permite ordenar nuestra aplicación en módulos que aportan diferentes funcionalidades a la aplicación siendo independientes entre ellos. Por lo tanto, si un día queremos actualizar una parte de la aplicación, no comprometemos los otros módulos.
- **Portable**: está escrito en Python permitiendo así a Django ser multiplataforma.

Django cuenta con varias capas que podemos manejar, nos centraremos en las más importantes para el desarrollo web:

- **Capa de modelos**: Django organiza la información de la aplicación en modelos. Cada modelo contiene varios campos donde se almacena diferente información, entendiendo un modelo como una tabla de una base de datos *relacional*. Los modelos son representados con una clase de Python donde cada variable es un atributo de columna de la tabla. Una gran ventaja de Django es que ofrece una API para poder realizar las operaciones necesarias sobre los modelos. Todos los modelos se almacenan en el archivo `models.py`.
- **Capa de vistas**: las vistas nos permiten definir la lógica que procesará y dará respuesta a las diferentes peticiones de los usuarios. Django nos ofrece dos archivos, `urls.py` y `views.py`. En el archivo `urls.py` definiremos múltiples urls asociadas a diferentes acciones de la aplicación como inicio de sesión o pedir un archivo. Cada url invoca un método del `views.py` el cual dará respuesta a la petición del usuario, devolviendo la información correspondiente.
- **Capa de plantillas**: Django permite diseñar las páginas que visualizará el usuario usando lenguajes como HTML o CSS, igualmente nos permite establecer condiciones, iterar sobre la información pudiendo mostrar la misma página de diferentes maneras dependiendo de las condiciones que definamos.

En el flujo de ejecución normal en Django, un usuario realiza una petición que estará asociada con una url del archivo `urls.py`, esta url tiene asociada un método del archivo `views.py` donde se procesa la petición, por ejemplo, recuperar la información del usuario almacenada en un modelo. A continuación, el usuario recibirá la respuesta del servidor.

3.2. ElasticSearch

Base de datos NoSQL, implica que no usa tablas para almacenar su información. En el TFG se ha utilizado la versión 7.12 que permite almacenar la información asociada a los torneos (participantes, puntuaciones, etc). La estructura utilizada para guardar la información determina el tipo de base de datos. ElasticSearch [2] emplea documentos para almacenar la información, por lo tanto, es de tipo documental. Las principales ventajas de ElasticSearch son:

- **Acceso a los datos**: han desarrollado su propio "*Query DSL*", lenguaje específico para una aplicación determinada, en este caso, para realizar las consultas. El lenguaje se basa en JSON, donde los documentos son pares de datos clave-valor.
- **Escalabilidad**: usa una arquitectura distribuida que permite su escalado horizontal, se basa en tener varios servidores que reparten la carga de trabajo. Este conjunto de servidores se denomina *Cluster*. Nos da un gran escalado en nuestro sistema y una alta disponibilidad de los datos.
- **Documental**: como ya habíamos comentado, es una base de datos NoSQL basada en documentos JSON, esto ofrece una característica llamada "*schemaless*"permitiendo que datos almacenados en un mismo documento tengan diferentes campos, dando más flexibilidad a la hora del desarrollo.

ElasticSearch utiliza nodos que pueden realizar diferentes tareas, en nuestro caso nos interesan los del tipo "*Data Node*", los cuales sirven para almacenar datos y realizar las búsquedas necesarias. Los nodos se almacenan en *Clusters* y los nodos de un mismo *Cluster* conocen la existencia del resto, permitiendo el intercambio de información entre sí. Los nodos almacenan índices que permiten la agrupación de documentos con características similares, dentro de un mismo nodo puede haber diferentes índices.

ElasticSearch emplea la técnica del *sharding* que permite dividir los diferentes índices en *shards*, réplicas de la información almacenadas en diferentes nodos, dotando a ElasticSearch de tolerancia al fallo y escalada horizontal. En caso de pérdida de un nodo, permite reorganizar la información garantizando su disponibilidad.

Para realizar búsquedas de texto completo ElasticSearch, usa una estructura de datos llamada índice invertido, entendido como un *hashmap* al cual le damos una palabra o número y nos devuelve la referencia de los documentos donde se almacena (ver Figura 3.1).

[2]https://www.elastic.co/es/

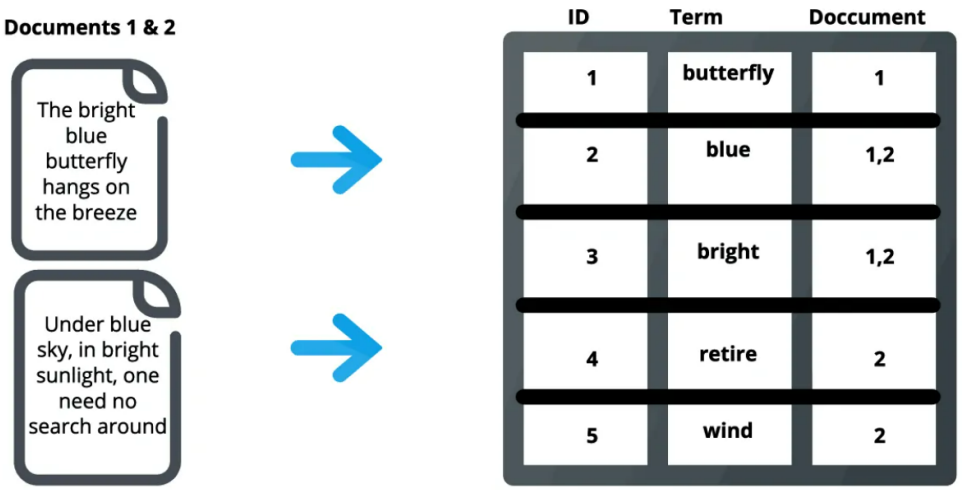

Figura 3.1: Ejemplo índice invertido (Fuente)

3.3. Tecnologías web del lado del cliente

En este apartado se van a revisar las tecnologías más frecuentes del lado del cliente HTML, CSS y JavaScript.

HTML o "Lenguaje de Marcas de Hipertexto" en castellano. Como su nombre indica, no es un lenguaje de programación sino de marcado. Con "hipertexto" hace referencia a la capacidad de vincular un documento con otros, ya que enlazar información es un aspecto base de la web. La versión utilizada en el TFG ha sido HTML5.

El lenguaje HTML posibilita estructurar el contenido del sitio web mediante el uso de etiquetas que permiten diferenciar los elementos que forman la página como imágenes, texto o título. Los elementos HTML se distinguen por el nombre de la etiqueta, el cual se expresa entre "<>". El nombre de un elemento no es sensible a las mayúsculas o minúsculas (*case sensitive*), pero existen normas de estilo que recomiendan desde la W3C. Se emplean atributos para definir el comportamiento de un elemento, no se puede usar cualquier atributo en un elemento HTML, cada uno tiene definido una serie de ellos que puede usar.

Los documentos HTML cuentan con elementos que definen su estructura (ver Figura 3.2):

- **<!DOCTYPE html>**: indica al navegador el tipo de documento a esperar, siempre se sitúa en la primera línea. No funciona como un elemento HTML, por lo tanto, es sensible a mayúsculas y minúsculas.
- **<html></html>**: el contenido de la página va dentro de esta etiqueta.

- **<head></head>**: aquí se sitúa la información (metadatos) no visible para el usuario, como el título o enlaces a hojas de estilo.
- **<body></body>**: recoge el contenido que vamos a mostrar a los usuarios.
- **<meta charset="utf-8">**: define el formato de codificación de caracteres, el más usado es "utf-8" que incluye la mayoría.
- **<title></title>**: título de la página que aparece en la barra del navegador y siempre va en la etiqueta <head>.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Mi pagina de prueba</title>
  </head>
  <body>
    <img src="images/firefox-icon.png" alt="Mi imagen de prueba">
  </body>
</html>
```

Figura 3.2: Ejemplo de un documento HTML

La siguiente tecnología es JavaScript (se ha utilizado la versión ECMAScript 6), lenguaje de programación popular en el desarrollo web para el lado del cliente. Permite la manipulación de la página web a través del *DOM*, representación estructurada de la página que permite realizar modificaciones gracias a JavaScript. Sus principales características son:

- **Orientado a objetos**: un objeto pertenece a una clase, la cual tiene un serie de datos y métodos que nos permiten utilizarlo. La programación orientada a objetos (POO) aporta diferentes beneficios: *encapsulación* (implica que otros objetos pueden no tener autoridad para realizar cambios sobre la clase), *abstracción* (uso de interfaces que permiten usar sus métodos sin necesidad de conocer cómo funcionan de manera interna), *herencia* (posibilidad de crear relaciones entre clases ya existentes, pudiendo reutilizar sus métodos o sobrescribir un comportamiento determinado) y *polimorfismo* (el programa puede interpretar los objetos de una manera u otra dependiendo del contexto, por ejemplo, definir una lista de la clase padre y añadir clases hijas dentro de esa lista y serían interpretadas como la clase padre).

- **Basado en prototipos**: no es necesario definir la clase de manera explícita si no que agregamos nuevas propiedades a instancias de una ya existente.

- **Imperativo**: las instrucciones del programa se ejecutan de manera secuencial.

- **Dinámico**: los tipos de las variables son determinados en tiempo de ejecución.

Por último, CSS (*Cascading Style Sheet*) es un lenguaje de estilos que permite definir la apariencia de nuestra página web. HTML no fue creado con la intención de definir estilos, sino para describir el contenido de la página, por ello se creó CSS permitiendo su definición de manera más sencilla, además de poder aplicarlo a varios elementos de manera simultánea mediante selectores. Permite reutilizar estilos que debemos definir en un archivo aparte. Hay tres maneras de añadir CSS a nuestro archivo HTML:

- **Atributo style**: podemos añadir la palabra *style* dentro de nuestro elemento HTML para definir sus distintas propiedades. Método muy ineficiente, ya que sólo lo aplicamos a ese elemento y es necesario repetirlo en el resto.

- **Etiqueta style**: utilizando la etiqueta *style* en la cabecera del documento HTML (recordar que aquí se definían los metadatos) podemos definir los estilos de los elementos. Es más ordenado que el caso anterior pero no es eficiente si queremos reutilizar el mismo estilo en otros documentos.

- **Archivo externo**: podemos crear un archivo CSS externo para definir el estilo de los elementos que forman la página y vincularlo con la etiqueta *link* en la cabecera del documento HTML. Este método es el más correcto dejando un código limpio y estructurado, además de permitir la reutilización del archivo .css en varias páginas.

3.4. Selenium

Permite la automatización de tareas y pruebas en navegadores web para verificar si el software devuelve los resultados esperados. Simula acciones del usuario humano de un navegador web, como clicar con el ratón en algún botón, introducir texto, etc y permite hacerlo programáticamente. En el TFG se ha utilizado Selenium[3] para la automatización de los torneos. Selenium está formado por 3 partes:

[3]https://www.selenium.dev/

- **Selenium RC**: permite realizar pruebas sobre la aplicación web independientemente del lenguaje de programación utilizado, es una herramienta muy flexible para ejecutar pruebas externas. La finalidad de Selenium RC es verificar que la aplicación funciona en los diferentes navegadores.

- **Selenium Webdriver**: versión más avanzada de Selenium RC, sirve para la automatización de pruebas o tareas sobre una interfaz de usuario, simulando el comportamiento de un usuario real en la página. Cuenta con una arquitectura más moderna, eliminando el servidor que actúa como proxy para controlar el navegador web y se comunica de manera directa con el navegador.

- **Selenium Grid**: al igual que Selenium Webdriver era una evolución respecto a la ejecución de pruebas en Selenium RC, Selenium Grid es una versión mejorada respecto a la ejecución de pruebas en diferentes dispositivos, en este caso permite realizar pruebas de manera simultánea y probar la aplicación en diferentes sistemas operativos.

La versión 4.1.2 de Selenium Webdriver ha sido la versión más utilizada en este trabajo, se explicará con mayor nivel de detalle. Se ha comentado que esta versión se especializa en la ejecución de pruebas y automatización de tareas en navegadores web, pero tiene limitaciones, no es capaz de interactuar con elementos ajenos al navegador como consolas de comando. Esta versión de Selenium está formada por cuatro componentes(ver Figura 3.3):

- **Librería de cliente**: permite utilizar Selenium con diferentes lenguajes, sólo se debe instalar la extensión necesaria.

- **Protocolo JSON**: estándar de comunicación para la transferencia de información entre el cliente y servidor web, los mensajes son escritos en formato JSON.

- **Drivers del navegador**: Selenium ofrece diferentes drivers para cada navegador, se debe instalar el driver correspondiente que se comunicará con el navegador. Cuando ejecutamos una tarea con Selenium WebDriver se realizan los siguientes pasos: para cada comando de Selenium se genera una petición HTTP, la cual es gestionada por el driver del navegador. A continuación, el driver ejecuta las acciones correspondientes sobre el navegador y devuelve el estado al programa.

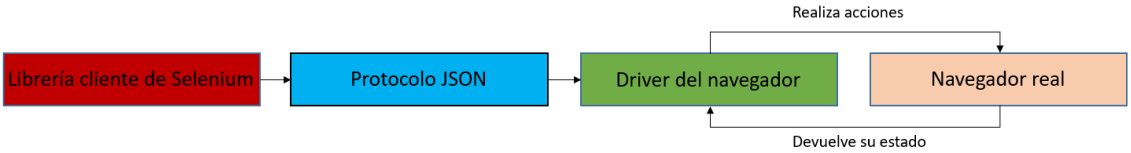

Figura 3.3: Arquitectura Selenium WebDriver

3.5. Docker

Docker [4] permite realizar un paquete de nuestra aplicación y ejecutarlo en un entorno conocido como contenedor. Para trabajar con Docker seguimos los siguientes pasos: desarrollamos nuestra aplicación usando contenedores y los separamos en partes para realizar pruebas individuales sobre ellos. Por último, desplegamos toda nuestra aplicación en un mismo contenedor. En el TFG se ha utilizado la versión 20.10 de Docker, evitando la instalación y configuración de ElasticSearch, MySQL y RADI (*Robotics-Academy Docker Image*) permitiendo crear las bases de datos correspondientes mediante un contenedor Docker.

La arquitectura de Docker está formada por los siguientes componentes (ver Figura 3.4):

- Cliente Docker: permite a los usuarios interactuar con la aplicación Docker, encargándose de enviar los comandos del usuario al Docker daemon correspondiente, un cliente Docker puede comunicarse con varios Docker daemon.
- Docker daemon: responde a las peticiones de los clientes y se encarga del manejo de imágenes y contenedores. También puede comunicarse con otros Docker Daemon.
- Escritorio Docker: aplicación que simplifica el manejo de Docker, permite arrancar o eliminar contenedores mediante su interfaz. Esta aplicación engloba los dos componentes anteriores.
- Imágenes: conjunto de instrucciones a seguir para construir un contenedor Docker, solamente se pueden leer, no admiten modificaciones. Normalmente usamos plantillas creadas previamente y añadimos nuestras modificaciones. Para construir la imagen usamos un archivo llamado "Dockerfile" donde definimos los pasos para generar la imagen, estas imágenes son muy ligeras y rápidas comparadas a otros métodos de virtualización (usando software podemos replicar hardware en un sistema virtual, permitiendo tener diferentes sistemas operativos en un mismo dispositivo). Cuando realizamos

[4] https://www.docker.com/

cambios en algunos de los comandos del Dockerfile al volver a construir la imagen, se mantienen las partes asociadas a los comandos que no han sido modificados.

- Contenedores: cuando ejecutamos una imagen Docker se crea un contenedor, pudiendo almacenar información en él o crear nuevas imágenes basadas en su estado. Por defecto, un contenedor está aislado de otros contenedores y de la máquina *host*, podemos modificarlo si lo indicamos en la imagen a la hora de su creación. Los contenedores pueden ser detenidos y reanudar su ejecución posteriormente, pero si lo eliminamos, la información almacenada en él se perderá.

- Registros: sirven para almacenar las imágenes Docker, los registros se encuentran almacenados en Docker Hub donde podemos decidir si hacer nuestro registro público o privado. Por ejemplo, para usar Unibotics como desarrollador necesitamos descargar la imagen Docker de MySQL, por lo tanto, cuando descargamos la imagen estamos accediendo al registro alojado en Docker Hub para obtenerla.

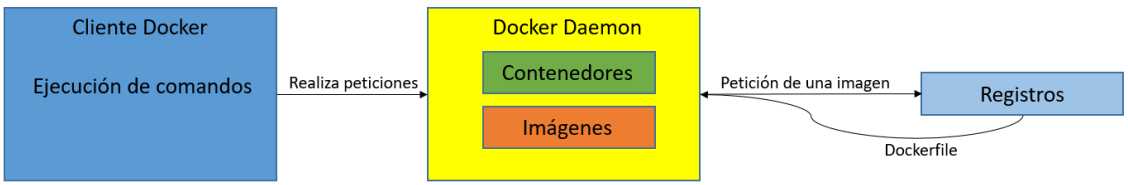

Figura 3.4: Arquitectura Docker

3.6. Python

Django es un framework que emplea Python, por tanto, este lenguaje es con el que se ha desarrollado la mayoría del TFG. La versión utilizada de Python [5] ha sido la 3.8. Algunas características de este lenguaje de programación son:

- Alto nivel: tiene una alta abstracción del hardware de la máquina.

- Interpretado: el código no necesita ser compilado antes de su ejecución, las instrucciones se leen en tiempo real y se ejecutan.

- Tipado dinámico: el tipo de una variable puede cambiar dependiendo del dato guardado en ella, esta ventaja viene de ser un lenguaje interpretado y no compilado.

[5] https://www.python.org/

- Recolector de basura: sistema automático para liberar espacio de la memoria ocupada por el programa, se liberan las partes que ya no son referenciadas lo que permite al programador abstraerse del manejo manual de la memoria del programa.

- Instalación de paquetes: la facilidad que ofrece Python para añadir paquetes desarrollados por otras personas es muy grande gracias al soporte que ofrece la comunidad. Podemos encontrar multitud de paquetes lo que facilita el desarrollo (ver Figura 3.5).

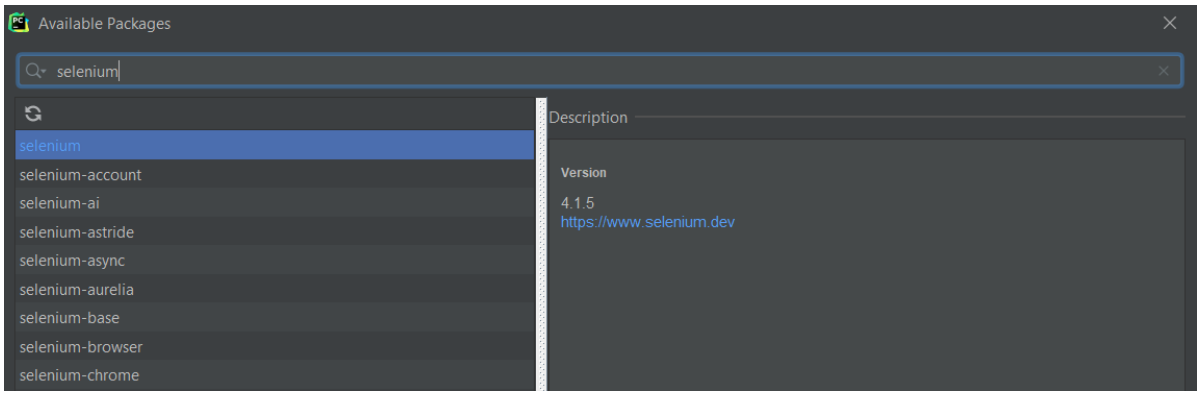

Figura 3.5: Instalación de paquetes en PyCharm

3.7. Twitch

Es una de las plataformas más conocidas para la emisión de directos en vivo. En el TFG hemos utilizado Twitch[6] para compartir los torneos desde la perspectiva del maestro de ceremonias, que muestra cómo se van ejecutando los diferentes códigos fuente en el robot del torneo. Hay dos maneras muy difundidas de emisión en directo:

- *Streaming multicast*: se define como "uno para muchos", el servidor manda la información a los clientes conectados en un área determinada. Esto tiene una serie de ventajas: es una comunicación segura, alta calidad de imagen gracias a que existen técnicas de predicción del ancho de banda necesario, ya que todo el mundo visualiza la misma imagen en el directo. Sin embargo, un problema de esta técnica es que asume que todos los espectadores disponen de conexiones y dispositivos similares, esto puede ser un problema en redes muy heterogéneas, dificultando el escalado y al ofrecer la información a espectadores con anchos de banda menores.

[6]https://www.twitch.tv/

- *Pseudo-streaming unicast*: se define como "uno a uno", el contenido puede pedirlo un único usuario y entregárselo de manera específica. Esto sirve en servicios de *streaming* como Netflix. Una de las ventajas de esta técnica es que se ajusta al ancho de banda disponible del cliente, para ello el vídeo emitido en directo se transmite a varios flujos con diferentes calidades. Sin embargo, el *unicast* no soporta la pérdida de paquetes pudiendo provocar retardos en la transmisión de la información.

Twitch ofrece una API para poder añadir una ventana de emisión en directo en nuestra página web, únicamente necesitamos añadir la API de Twitch en nuestro archivo html y cumplir unas condiciones necesarias, como indicar el canal que querremos retransmitir. Como hemos vinculado la cuenta de Twitch de Unibotics con el programa OBS, al comenzar directo se transmite vía Twitch y con esta API podemos visualizar el directo en nuestra página web (ver Figura 3.6).

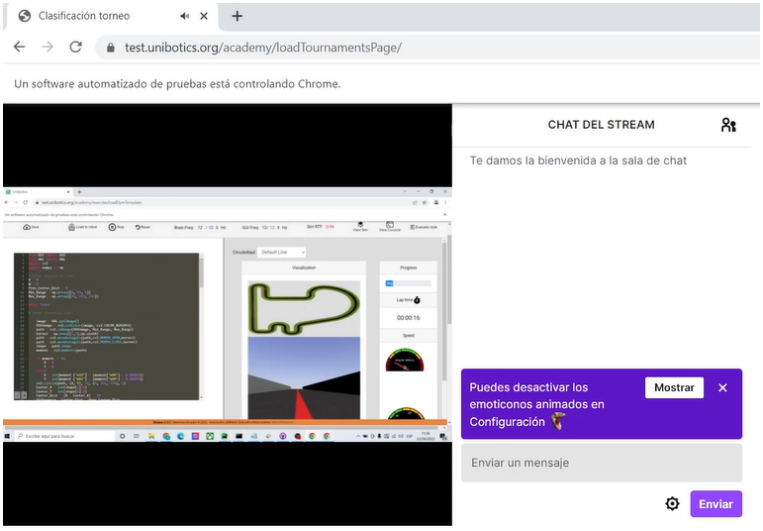

Figura 3.6: Api de Twitch en Unibotics (Fuente)

3.8. Tecnologías WebSocket

La tecnología WebSocket surgió como alternativa a las limitaciones de HTTP. En HTTP la comunicación es unidireccional, significa que el servidor no va a enviar ningún dato al cliente sin una petición previa. Para solventar este problema, HTTP usaba la técnica de sondeo largo o "*long-polling*", aquí la respuesta del servidor no es inmediata, este tiempo se necesita para buscar los datos. Pero tiene una limitación, incluso si no hay datos disponibles para mandar al cliente deberá esperar el tiempo indicado para recibir la respuesta. Los WebSockets permiten mandar la información mediante TCP asegurando la entrega de la información, aunque usa HTTP como proceso inicial para establecer la comunicación. Una vez

establecida, mantiene una conexión TCP permitiendo el intercambio en tiempo real entre cliente y servidor sin la necesidad del "*long polling*".

Las comunicaciones en WebSocket son bidireccionales, lo que implica que son capaces de mandar peticiones por iniciativa de cualquiera de los extremos. Si alguno de los miembros abandona la conexión, se imposibilita el paso de mensajes. El proceso para establecer la comunicación es:

- *Handshake*: el cliente establece la conexión con el servidor para obtener un WebSocket, esta petición se realiza mediante HTTP y el servidor indica si es posible establecerla.
- Comunicación: si la respuesta del servidor en el paso anterior es satisfactoria, cliente y servidor usarán la conexión establecida mediante HTTP para el paso de mensajes mediante un protocolo WebSocket.

Los mensajes WebSocket tienen una serie de valores que se deben incluir siempre:

- *Connection*: valor que controla si la conexión debe seguir activa una vez terminada la petición actual, normalmente tiene el valor "*keep-alive*" indicando que la conexión siga activa.
- *Upgrade*: usado por los clientes para establecer el tipo de protocolo de comunicación, normalmente tendrá el valor "WebSocket"
- *WebSocket-Version*: versión del protocolo de comunicación aceptada para el WebSocket, no aceptará versiones diferentes.

3.9. Unibotics

La página web de Unibotics ha sido la base para el desarrollo del TFG, se explicará aquí su arquitectura y características. También se verá el estado en el que se encontraba la página antes del inicio de este trabajo.

Unibotics utiliza un cliente y un servidor web. El servidor web utiliza el *framework* Django para el diseño de la página web, incluye comunicación cliente-servidor y las plantillas que se muestran al usuario, también utiliza bases de datos de ElasticSearch y MySQL en las que almacena diversa información, como perfiles de usuarios o ejercicios, y los servidores de Amazon S3 en la nube, donde se guardan los códigos fuente de los usuarios desarrollados en los ejercicios.

Para evitar que el usuario deba instalar los archivos necesarios para el intercambio de mensajes con el robot o el simulador robótico Gazebo para la visualización del ejercicio, el cliente ejecutará un contenedor Docker. El cual establece

una conexión con el ejercicio del navegador del usuario mediante WebSockets, permitiendo su ejecución (ver Figura 3.7).

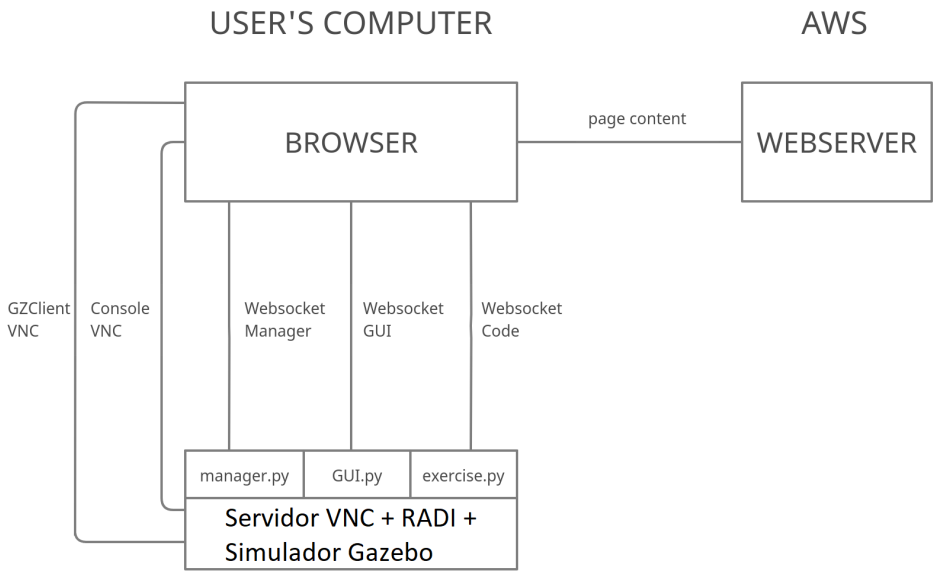

Figura 3.7: Arquitectura de Unibotics (Fuente)

Cuando un usuario accede a la página de Unibotics se muestran los ejercicios disponibles, cada uno cuenta con una descripción del contenido teórico del ejercicio y su objetivo.

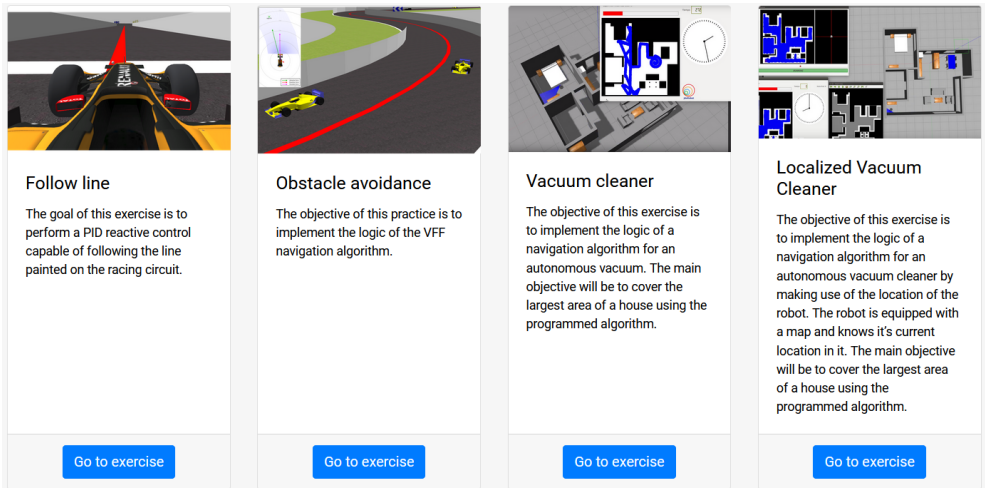

Figura 3.8: Ejercicios en Unibotics (Fuente)

Una vez se accede a un ejercicio se muestra la página web de esa unidad didáctica. Está compuesta por un editor de Python empotrado en el navegador, un

visor VNC del simulador, una botonera para cargar el código fuente como cerebro en un robot simulado, para ejecutar ese código, para reiniciar la simulación, varios *widgets* específicos para ver información sensorial, etc.

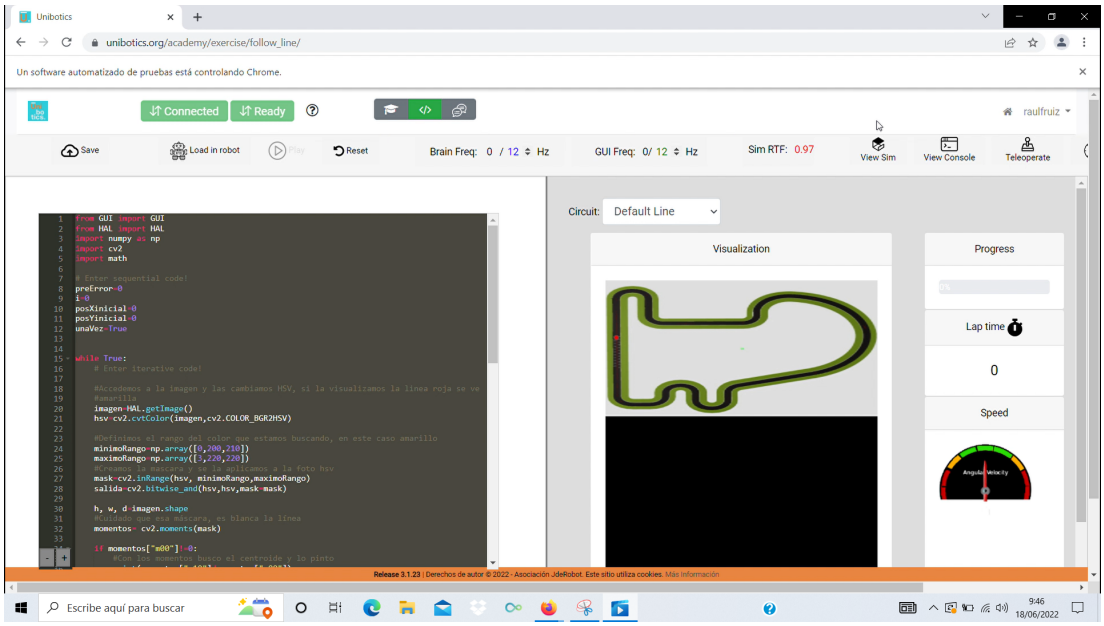

Figura 3.9: Ejercicios en Unibotics (Fuente)

Desde el punto de vista de desarrollo software, el código fuente de Unibotics se divide en dos repositorios, *Unibotics Exercises* y *Unibotics WebServer*. En el primero se almacenan los ejercicios de la página web y en el segundo el servidor Django, esto incluye los accesos a las bases de datos, respuestas a las peticiones del cliente, plantillas HTML no asociadas con los ejercicios (por ejemplo, página principal o registro), etc. Cuando Unibotics se despliega, el repositorio de *Unibotics Exercises* se incluye dentro del repositorio *Unibotics Webserver*.

Unibotics utiliza diferentes despliegues durante el desarrollo:

- Despliegue local para desarrolladores (D1): el servidor web y bases de datos se ejecutan en el ordenador local. Durante esta fase probamos los cambios en local en la dirección 127.0.0.1.

- Despliegue en preproducción o test (D2): en esta fase se hace uso de los ordenadores disponibles en los laboratorios de la URJC, esto requiere algunos pasos previos para lanzar este despliegue, la finalidad de esta fase es probar el código desarrollado en un entorno prácticamente igual al de producción, permitiendo detectar fallos que no se habían tenido en cuenta en el despliegue previo. El servidor web y las bases de datos se ejecutarán en un

ordenador del laboratorio, los cambios se prueban en una URL [7] especial de pruebas.

- Despliegue en producción (D3): para usar este despliegue nuestro código debe ser integrado en la rama *master*, lo que implica que el código ha sido supervisado por un miembro experimentado del equipo de desarrolladores de Unibotics. El servidor web y las bases de datos se ejecutan en un servidor en la nube de AWS. Los cambios realizados se reflejan en la página oficial de Unibotics[8].

[7] https://test.unibotics.org
[8] https://unibotics.org

4

Automatización de torneos en Unibotics

En este capítulo se verá el desarrollo software específico del TFG. Primero se describirá de manera general el diseño de todos los componentes desarrollados y su relación entre ellos. A continuación se describirá la infraestructura desarrollada para torneos, destacando especialmente el usuario instrumental, creado para ellos, que se llama "maestro de ceremonias". En la tercera sección se aborda la automatización de los torneos, mediante el uso de Selenium. Posteriormente, se describe la gestión de los torneos desde su creación y edición hasta la inscripción de participantes. Por último, se detalla la validación experimental que muestra que los torneos automáticos creados funcionan satisfactoriamente.

4.1. Diseño de los torneos y arquitectura

La arquitectura general de los torneos automáticos desarrollados en este TFG se divide en tres partes (ver Figura 4.1):

- Usuario instrumental: para el control de los torneos se necesitaba un nuevo usuario con permisos especiales, el "maestro de ceremonias". Este usuario es el encargado de ejecutar los códigos de los usuarios, cargar el código en el robot, creación de torneos, etc. Además necesitaba funciones especiales las cuales no existían en la plataforma, como poder obtener códigos fuente de otros usuarios.

- Automatización: este punto está compuesto por dos elementos. El primero son subprocesos que permiten el lanzamiento y detención de los contene-

dores Docker de manera automática. El segundo es Selenium, el cual ha posibilitado la automatización de la ejecución del torneo emulando a una persona que entra en la página web de Unibotics, presiona los botones, introduce valores en cajas de texto, etc, todo ello de manera programática y automatizada.

- Gestión de los torneos: elaboración de vistas donde los usuarios se pueden registrar en los torneos, resultados en vivo, edición de los torneos, etc.

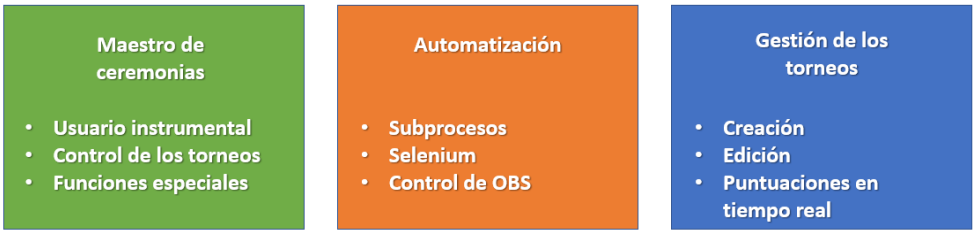

Figura 4.1: Arquitectura de los torneos automáticos

La relación entre los diferentes componentes de este trabajo se presenta en la Figura 4.2. Se explica a continuación someramente, y de modo más detallado en el resto de secciones de este capítulo:

- OBS: programa para controlar el directo emitido en Twitch, permite asignar un puerto del ordenador para comunicarse con otros programas. El cliente cuenta con una librería específica para la conexión WebSocket con OBS, permitiendo el envío de órdenes para controlar el directo (comenzar emisión, cambiar escena, terminar emisión).

- Cliente: navegador web donde se muestra la página de Unibotics y renderiza las plantillas Django, realiza peticiones. En nuestro caso el cliente será el navegador a través del cuál entra en Unibotics el usuario maestro de ceremonias, encargado del control del torneo.

- Selenium: utilizado para automatizar al maestro de ceremonias, permitiendo la ejecución autónoma del torneo y lanzamiento de los contenedores Docker necesarios para simular los ejercicios.

- Servidor: el servidor de Unibotics se encarga de dar respuesta a las peticiones del cliente y de acceder a la base de datos en ElasticSearch.

- ElasticSearch: base de datos donde se almacena la información de los torneos: participantes, clasificación, fechas...

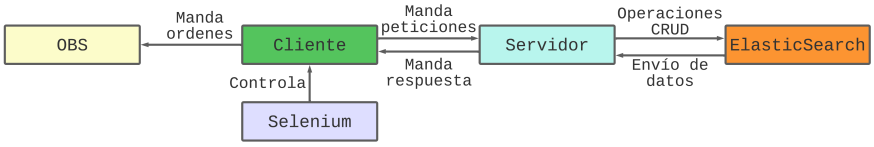

Figura 4.2: Comunicación entre los componentes del sistema de torneos automáticos

4.2. Infraestructura de torneos

En esta sección se explica el diseño del *usuario instrumental*, el cual tiene atribuidas funciones y plantillas especiales de las que sólo dispone él, por ejemplo, poder acceder al código fuente de otros usuarios.

4.2.1. Maestro de ceremonias

En la base de datos los usuarios se dividen por roles, por lo que el primer paso será crear un rol para el maestro de ceremonias, encargado de la gestión, creación y emisión de torneos. En Django existe un archivo llamado `models.py`, que contiene la estructura de las tablas de la base de datos, en la tabla *User* se almacenan los usuarios de Unibotics, por lo tanto, en el atributo roles añadimos uno para el maestro de ceremonias.

Para añadir el nuevo rol en la tabla de la base de datos, no basta con modificar el archivo `models.py`, es necesario aplicar este cambio en la base de datos. Gracias a Django no es necesario escribir los comandos SQL:

- `python manage.py makemigrations`, crea los comandos SQL correspondientes.
- `python manage.py migrate`, ejecuta los comandos SQL.

El siguiente paso consiste en crear un nuevo usuario con rol de maestro. Esta tarea puede resultar compleja si no se aprovechan las ventajas de Django. Este *framework* ofrece una url especial para el manejo de la base de datos llamada "*admin*"(ver Figura 4.3). Esta interfaz lee toda la información almacenada en los modelos Django y la muestra en tablas que permiten realizar modificaciones en los datos almacenados en los modelos. Este acceso está restringido a determinados usuarios.

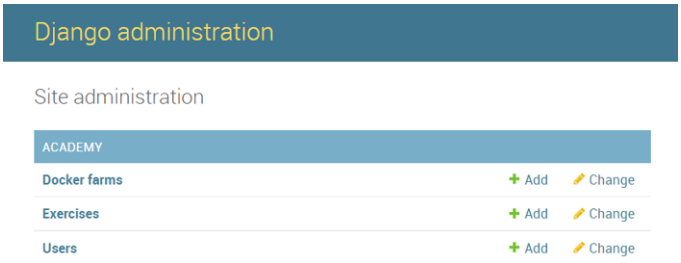

Figura 4.3: Página de administración Django

Desde aquí podemos modificar el rol de usuario creado y aplicar los cambios directamente (ver Figura 4.4).

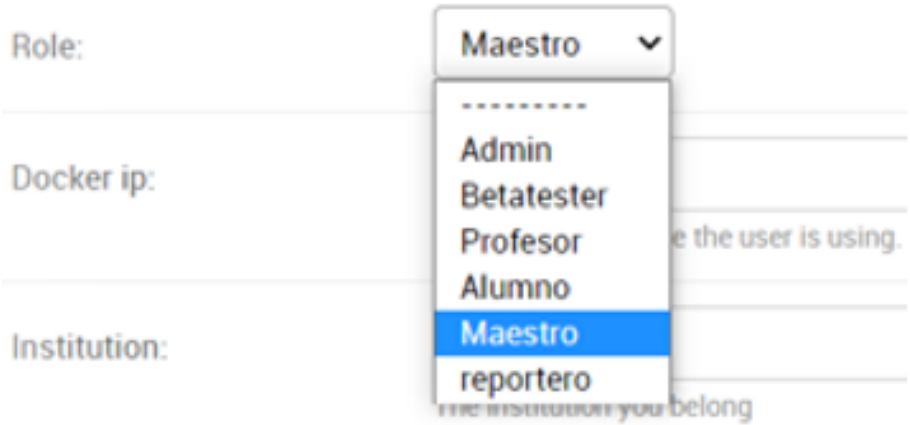

Figura 4.4: Cambio de rol

4.2.2. Vistas personalizadas dependiendo del rol

Cuando un usuario inicia sesión y accede a la página se verifica si su rol es el de maestro de ceremonias para así dirigirle a una página especial donde pueda realizar las acciones pertinentes a los torneos. Si no lo es, a ese usuario normal se le muestran las páginas de ejercicios de Unibotics.

En primer lugar se debe localizar dónde se gestiona el acceso del usuario a la página en el lado del servidor. En el archivo `urls.py` se sitúan todas las peticiones que puede recibir el servidor desde el cliente. Cada petición se asocia a una url determinada e invoca una función que gestiona la petición. Cada url está compuesta de tres elementos (ver Figura 4.5):

- El primero es la url asociada a esa petición, en el caso de *log-in* la url será `/academy/login` (en Django las urls van precedidas por el nombre del módulo, en este caso `academy`).

- El segundo valor es el método que se invocará cuando se reciba una petición usando esa url, en este caso el método "*user login*"del archivo `views.py`.

- El último valor es un prefijo que añadimos si estamos trabajando con más aplicaciones, porque tener dos urls iguales puede provocar un conflicto. Con este valor se puede identificar fácilmente la url asociada al inicio de sesión.

```
urlpatterns = [

    path("", views.index, name="index"),
    path('login', views.user_login, name="login"),
```

Figura 4.5: Urls en Django

En la Figura 4.5 se ve cómo se invoca el método *user login* que gestiona los accesos de los usuarios a la página web. Este método verifica que el usuario ha introducido un nombre de cuenta y una contraseña correctas. A continuación se consulta su rol, para ello se ha añadido una comprobación que detecta al maestro de ceremonias y lo redirige a una página especial.

La Figura 4.6 muestra la vista donde el maestro de ceremonias realiza muchas de sus acciones. Más adelante se explicará en detalle el funcionamiento de cada una de ellas:

- La primera opción permite al maestro de ceremonias cargar el ejercicio que quiera. Se trata de una vista especial, distinta de la de los usuarios normales. Se utiliza durante los torneos para evaluar los códigos fuente de los usuarios.

- Para crear nuevos torneos el maestro selecciona el ejercicio, el día límite para registrarse y fecha en la que se celebrará.

- La tercera línea indica al servidor que se está retransmitiendo en directo, provocando que cambie la vista de la página del torneo. La cuarta línea se usa cuando no estamos retransmitiendo en directo.

- La última línea es una vista donde el maestro puede editar los torneos si se ha cometido un error al seleccionar el ejercicio, modificar plazo de inscripción o el día de celebración del evento.

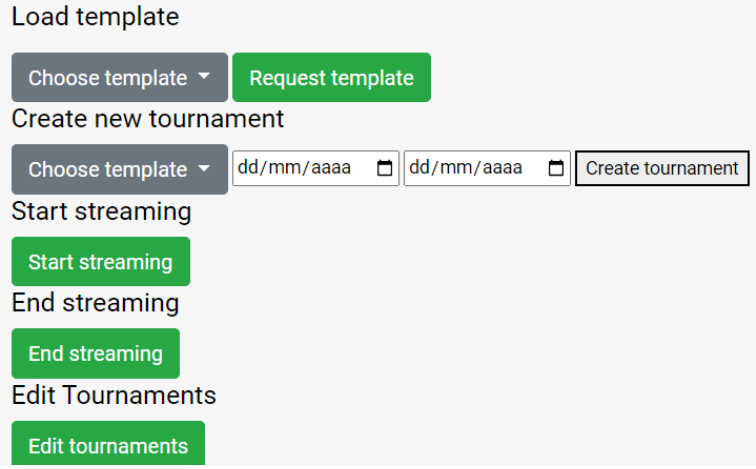

Figura 4.6: Vista inicial que se muestra al maestro de ceremonias

4.2.3. Plantillas dinámicas

Era necesario que el maestro de ceremonias pudiera cargar la interfaz del ejercicio que él quisiera añadiendo lo necesario para evaluar la eficacia del código del usuario. Es decir, si resuelve satisfactoriamente o no, y en qué medida, la tarea del ejercicio correspondiente. Este diseño debía hacerse de manera eficiente, ya que si se creaba una plantilla especial para cada ejercicio, se repetiría código innecesario y dificultaría su mantenimiento y escalabilidad.

El maestro de ceremonias selecciona en el primer desplegable el ejercicio que desea cargar y se envía la petición al servidor que invocará el método asociado a ese evento para gestionar la respuesta. En el servidor se obtiene la url del ejercicio pudiendo heredar de su plantilla. La vista que recibirá el maestro de ceremonias contendrá el ejercicio pedido y las funciones especiales para evaluar el código.

En la primera línea de la plantilla .html se usa la url del ejercicio para poder heredar de su plantilla incluyendo todos sus elementos. El resto de la plantilla son las funciones especiales para evaluar los códigos de los usuarios que veremos en otro punto.

```
{% include template_base %}
```

Se ha conseguido que, con una única plantilla, se pueda cargar el ejercicio y usar las herramientas para evaluar el código, en lugar de tener que crear un archivo para cada ejercicio. Este diseño es realmente útil de cara a la escalabilidad y mantenimiento de la aplicación.

4.2.4. Obtención de los códigos fuente de cada usuario

La obtención de los códigos fuente de otros usuarios era una funcionalidad muy importante para el desarrollo del maestro de ceremonias. Únicamente debe introducir el nombre del usuario para obtener su código fuente. La petición se maneja en el servidor de manera diferente dependiendo del despliegue. Los códigos de los usuarios reales se almacenan en servidores especiales S3 de Amazon que sólo son accesibles desde el despliegue D3. Para despliegues inferiores, se ha añadido una nueva carpeta en el repositorio donde se almacenan como archivos independientes algunos códigos fuente desarrollados por alumnos del máster en Visión Artificial de la URJC. En la respuesta al cliente se incluye un valor indicando si nos encontramos en el despliegue D3 o inferiores.

```
def cm_getUserCode(request):
 data = json.loads(request.body.decode("utf-8"))
 if settings.SERVER == "127.0.0.1":
       randomCodeId = random.randint(1, 10)
       path_to_code=f'academy/exampleCodesTournaments/usu{randomCodeId}.txt'
       with open(path_to_code, 'r') as f:
           codigotxt = f.readlines()
           code = str(codigotxt)
       context = {"codigo": codigotxt,"d3":0}
       return JsonResponse(context)
  else:
           username=data["userName"]
           idExercise=data["exerciseName"]
           user=User.objects.get(username=username)
           user_code =
               user.aws_pull_file(str(idExercise)).replace("\n",
               "/n")
           context = {"codigo": user_code,"d3":1}
           return JsonResponse(context)
```

Una vez se recibe el código fuente del usuario concreto en el *frontend* se comprueba el valor del despliegue en el que nos encontramos, en ambos casos el código es formateado pero de distintas maneras. En D2 cada línea viene precedida por el carácter “,” por lo tanto, hay que eliminarlos. Se usó una expresión regular para esta tarea, que va a recorrer cada línea del texto y borrar las comas que aparecen al inicio y final de cada línea. Para D3 se vuelve a usar una expresión regular, donde se sustituyen todos los “\n” por un salto de línea.

4.2.5. Enviar al servidor los resultados de la ejecución del código de un usuario

Esta función se emplea durante los torneos al terminar la ejecución del ejercicio. Se modifica el valor de la puntuación del usuario en el torneo almacenado en el documento de ElasticSearch. Por tanto, cuando llega la petición al servidor se localiza la entrada con el nombre del usuario y el identificador del torneo, actualizando la puntuación del mismo.

```
def cm_receivePuntuation(request):
    import json

    data = json.loads(request.body.decode("utf-8"))
    print("New puntuation received")
    print(data["score"])
    print(data["userName"])

    try:
        s = Search(index="tournamentUsers").query('match',
            username=data["userName"]).query('match',idtorneo=data["idTournament"])
        for hit in s:
            Elasticsearch(settings.ELASTICSEARCH_DSL['default']['hosts']).update(
            index= "tournamentUsers",id=hit.meta.id, body={
            "doc": {'score': data["score"]}})
            global updates
            updates = updates + 1
    except Exception as e:
        print("cm_ReceivePuntuacion ES error")
        print(e)
    return HttpResponse("OK")
```

4.3. Automatización

En la sección anterior se ha descrito toda la infraestructura desarrollada para que el usuario “maestro de ceremonias” , entrando manualmente en la web de Unibotics, pueda ejecutar el código fuente de los usuarios en el ejercicio del torneo y enviar al servidor la puntuación obtenida en su evaluación. Básicamente el proceso pasa por desplegar Unibotics en local utilizando contenedores con los subsistemas y utilizar Selenium para realizar automáticamente las tareas del maestro de ceremonias descritas en la sección previa. Por ejemplo ir cargando uno a uno el código fuente de todos los usuarios inscritos en el torneo, ejecutándolo y enviando al servidor su puntuación (ver Figura 4.7):

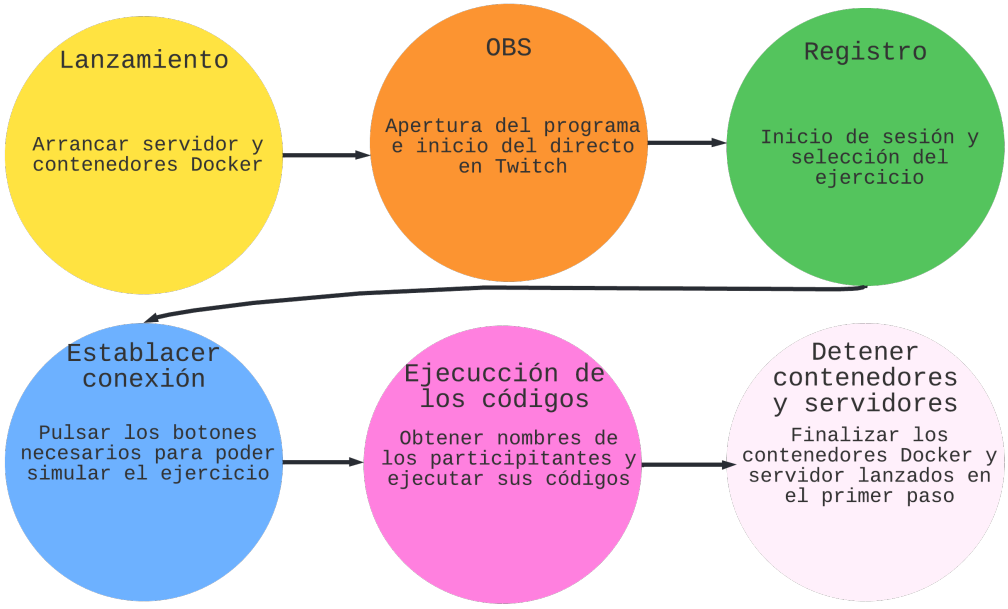

Figura 4.7: Tareas automatizadas

4.3.1. Lanzamiento de contenedores Docker

Para trabajar con la página de Unibotics en despliegue D1 es necesario lanzar una serie de contenedores Docker y el servidor. Primero se ejecutan dos subprocesos que contienen los comandos para activar los contenedores de ElasticSearch y MySQL. A continuación, es lanzado el servidor. Por el tipo de comando utilizado es necesario lanzar el subproceso desde el mismo directorio que el archivo `manage.py`. Por último, se arranca el *RADI (Robotics Academy Docker Image)*, necesario para simular los ejercicios y establecer una conexión con el robot.

4.3.2. Conexión WebSocket con OBS

El programa OBS se abre de manera automática. Para evitar los problemas de seguridad del programa en Windows y hacer que todas las personas que quieran usar el código deban darle permisos acceso, lanzamos la aplicación desde el acceso directo y no desde el .exe, ya que si usamos el ejecutable dará un error de acceso denegado (Figura 4.8).

```
start /d "C:\Program Files\obs-studio\bin\64bit" obs64.exe
Acceso denegado.
```

Figura 4.8: Error al iniciar el programa OBS desde el .exe

Se ha habilitado la función del servidor WebSocket incluido en el propio programa, permitiéndole recibir órdenes de programas externos (Figura 4.9). Por motivos de seguridad es preferible habilitar una autenticación para evitar conexiones no deseadas. También verificamos que el puerto usado por el WebSocket no produce conflicto con los otros puertos usados por Unibotics.

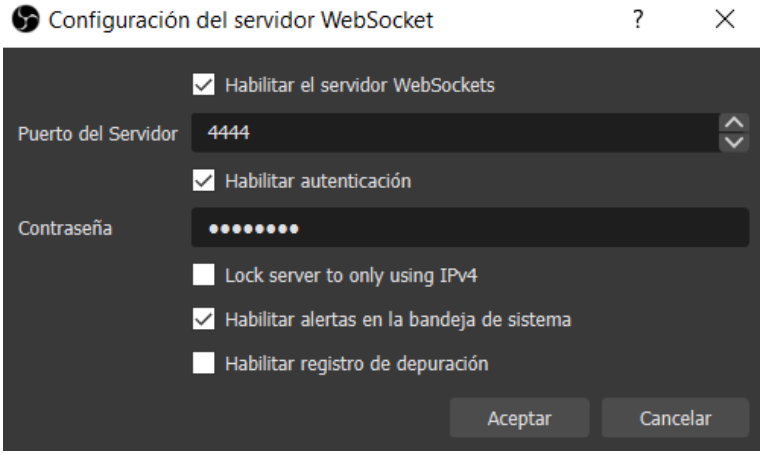

Figura 4.9: Configuración servidor OBS

Por último, era necesario encontrar la manera de mandar las órdenes al WebSocket de OBS. Se investigaron diferentes opciones y se optó por un proyecto de Github[1], el cual se encarga del paso de mensajes y establecer la conexión, por lo tanto, sólo se debe definir el *host*, el puerto donde se aloja el WebSocket y la contraseña para la autenticación.

Con estos elementos ya es posible abrir una conexión con el WebSocket y enviar diferentes peticiones permitiendo automatizar el control del OBS. Por último, señalar que una vez enviada la orden de comenzar directo, la conexión es cerrada. Por el momento esta es la única petición realizada, si en el futuro se añadieran más funcionalidades como cambiar de escena o comenzar a grabar, no sería necesario desconectar el WebSocket.

[1]https://github.com/Elektordi/obs-websocket-py

4.3.3. Acceso a la página web como maestro de ceremonias

En la sección 4.3.1 se vio cómo lanzar de manera automática los contenedores Docker y archivos necesarios para usar la aplicación de Unibotics. En la sección 4.3.2 se explicó cómo el WebSocket permite el paso de mensajes, por último, en esta sección se verá el uso del *framework* Selenium para el manejo automatizado de la página web, permitiendo simular todo el torneo sin necesidad de ser supervisado por una persona.

Para el funcionamiento de Selenium es necesario descargar el driver del navegador web concreto habilitando el uso automatizado del mismo. Automáticamente se abrirá el navegador que nos dirigirá a la página de Unibotics a través del método "*get*" ofrecido por Selenium. Se debe iniciar sesión con el usuario maestro de ceremonias. Para ello Selenium encontrará los elementos necesarios haciendo uso de los componentes que los distinguen. En este caso se localiza el botón de inicio de sesión con un selector CSS, el cual se identifica gracias a su atributo href con el valor "/academy/login".

Para comprender mejor el párrafo anterior vamos a mostrar un ejemplo. En la esquina superior izquierda de la Figura 4.10 aparecen los botones de registro e inicio de sesión en Unibotics. Selenium pulsa el botón de "*sign in*"para ir a la pestaña de inicio de sesión. Si se hace *click* derecho sobre el elemento y pulsamos la opción inspeccionar, se permite ver el código fuente de la página. Este elemento tiene un atributo que lo hace único y lo diferencia del resto (el atributo href con el valor "/academy/login").

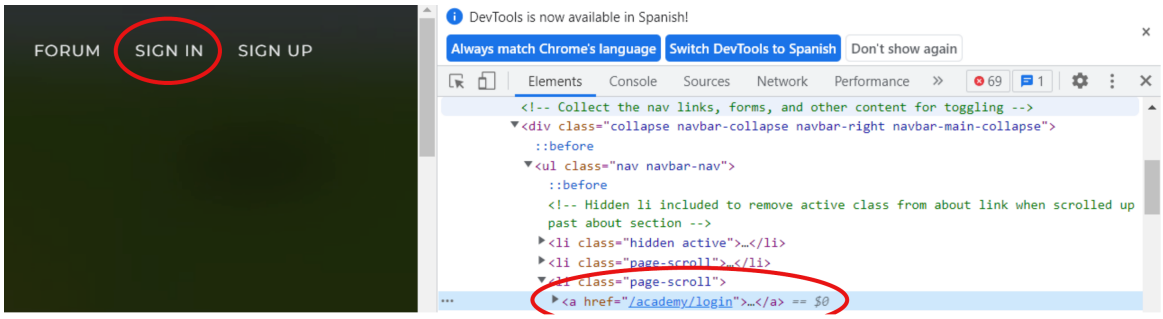

Figura 4.10: Funcionamiento básico de Selenium

A continuación el programa se sitúa en la ventana de inicio de sesión (ver Figura 4.11), en la que hay completar dos campos, el nombre del usuario y su contraseña, y se pulsa el botón para enviar el formulario. Se puede acceder a ellos haciendo uso del atributo id, es único para cada elemento HTML. Usando el método *send keys* es posible insertar texto en los elementos HTML, en este caso

se tiene guardado en una variable de Python el nombre y contraseña del usuario maestro de ceremonias, lo que permite a Selenium rellenar los campos.

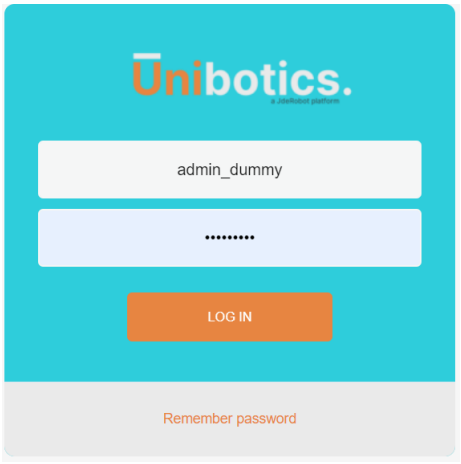

Figura 4.11: Inicio de sesión en Unibotics

El siguiente paso consiste en indicar al servidor que se va a iniciar directo, de manera que la vista del torneo cambia al modo de emisión (Figura 4.19). Se selecciona el botón de emisión mediante un identificador y se pulsa con el método "*click*" ofrecido por Selenium (Figura 4.12).

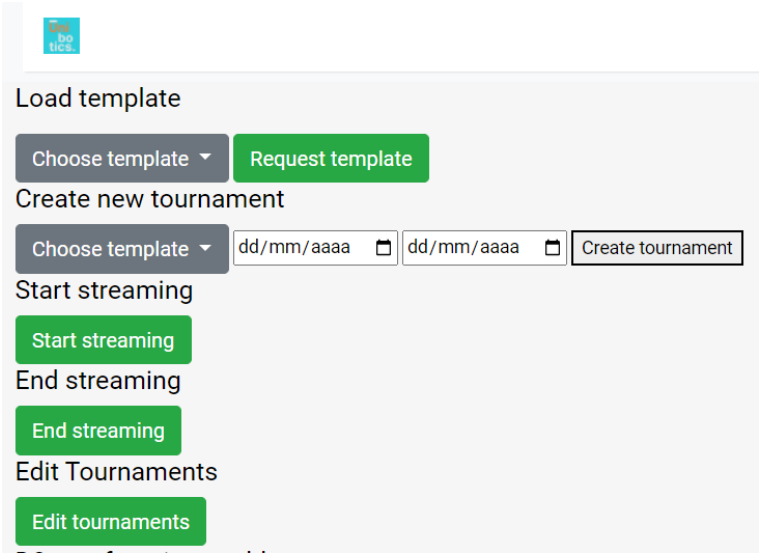

Figura 4.12: Vista inicial del maestro de ceremonias

En este punto se debe seleccionar del desplegable el ejercicio del torneo. El diseño HTML realizado facilita esta tarea ya que cada elemento de la lista cuenta con un método "*onClick*", donde el parámetro enviado es su propio nombre (ver

Figura 4.13). Por lo tanto, con un selector CSS se selecciona el elemento cuyo método "*onClick*" tenga como parámetro el nombre del ejercicio que buscamos. Una vez seleccionado se presiona el botón para cargar el ejercicio.

Figura 4.13: Selección automatizada de ejercicio

4.3.4. Establecer conexión con el ejercicio

En este momento el navegador se encuentra en la página web concreta del ejercicio, pero antes de lanzar su ejecución es necesario realizar algunos pasos. El primero es cerrar la ventana emergente (ver Figura 4.14) que indica las instrucciones de cómo comenzar manualmente el ejercicio. En este caso existía un problema al no haber ningún selector CSS o identificador para cerrarlo, por ello se usó XPATH, un lenguaje que permite recorrer los documentos teniendo en cuenta su estructura jerárquica. Esta solución debe ser la última, ya que si la página cambia su estructura en alguna actualización futura, la expresión XPATH podría dejar de funcionar.

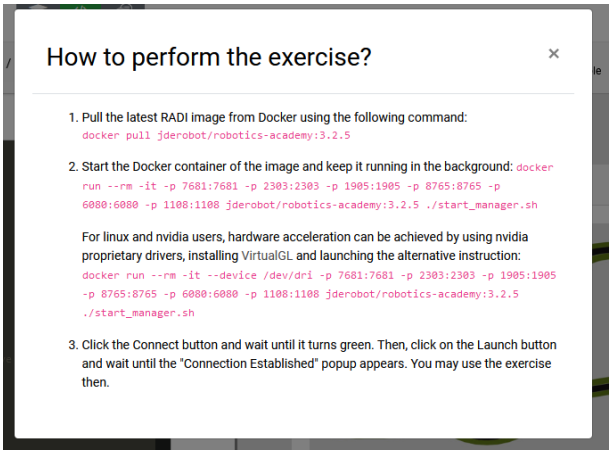

Figura 4.14: Desplegable del ejercicio

Selenium debe esperar a la aparición de la ventana emergente (tarda medio

segundo). Si no realiza esta espera, Selenium devolverá fallo al no encontrar el elemento. A continuación, se accede al elemento más cercano al botón de cierre dentro de la ventana emergente, y con la expresión XPATH se le indica que vuelva al elemento padre y seleccione el primer elemento hijo de la clase botón (en XPATH el primer elemento es el 1 y no el 0) y lo presione, cerrando la ventana.

En los ejercicios de Unibotics se debe establecer previamente la conexión con los WebSocket necesarios para el funcionamiento del ejercicio. El primero es el botón de conexión que conecta con el robot del ejercicio, para ello necesitamos tres componentes activos: los contenedores Docker de MySQL, ElasticSearch y el RADI. En nuestro caso no es un problema ya que se habían lanzado previamente. Este proceso de conexión lleva unos segundos, durante este tiempo en el botón aparecerá la palabra "*Connecting*" (ver Figura 4.15) y una vez haya terminado, "*Connected*". Se utilizarán estos términos para detectar que la conexión ha sido establecida en lugar de realizar una espera de tiempo fija, que en algunos casos se podría pasar de tiempo, o peor aún, continuar sin haber terminado de establecer la conexión lanzando un error. Con el otro botón realizamos un proceso similar, una vez presionado aparecerá una alerta indicando que el ejercicio está listo, por ello, una vez pulsado, el programa esperará a que aparezca una alerta en pantalla y una vez lo haga, la cierra.

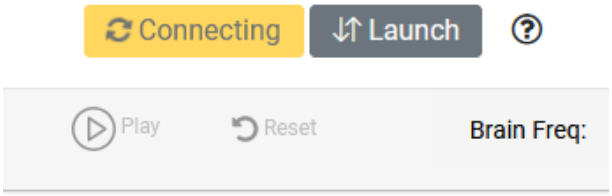

Figura 4.15: Botones de conexión en Unibotics

4.3.5. Ejecución del torneo

En este punto el ejercicio está listo para su ejecución, el siguiente paso es obtener el nombre de los usuarios que participan en el torneo. Para ello se envía una petición al servidor con el identificador del torneo, donde se comprueba el documento de ElasticSearch correspondiente, devolviendo todos los usuarios que aparezcan registrados. Una vez se reciben en el *frontend* se guardan en una lista. En este momento se realiza un bucle siguiendo estos pasos:

- Obtención del código: se escribe el nombre del usuario para obtener su código fuente.
- Cargar el código del usuario en el cerebro del robot: se carga el código del usuario en el robot, aparece un modal durante la carga del código que

desaparece una vez termina. Se debe indicar al programa que espere a que este elemento se vuelva invisible para continuar.

- Comprobación del código del usuario: si una vez cargado el código en el robot diera fallo por errores de sintaxis, aparecerá la ventana de error, en ese caso Selenium manda una puntuación de 0 y pasa directamente al siguiente participante. En caso de ser correcto, continua con la simulación. Esta funcionalidad ha añadido mucha robustez al desarrollo del torneo ya que los usuarios no siempre entregan un código fuente sin errores de sintaxis.

- Reiniciar la simulación: una vez el código de un usuario es cargado en el robot, se reinicia la simulación para que el ejercicio vuelva a empezar desde su situación de arranque. Mientras comienza de nuevo, el botón de *play* permanece como elemento no interactuable. Se usa esta propiedad para saber cuándo ha terminado de reiniciar el ejercicio y poder pulsar el botón de jugar.

- Mandar resultado al servidor: se deja ejecutar el ejercicio unos segundos, se detiene y se envía el resultado al servidor.

4.3.6. Detener los contenedores Docker y servidor

Una vez terminado el torneo, es decir, ejecutado el código fuente de todos los participantes, se detienen los contenedores Docker y el servidor lanzados en pasos previos:

- Detener contenedores Docker: el comando "deactivate" sirve para abandonar el entorno virtual de Python (no es obligatorio pero es más correcto realizarlo desde fuera). Se itera sobre todos los contenedores Docker guardados en la lista "tokens", éstos se obtienen con el comando "docker ps -q" , el cual nos devuelve los contenedores en ejecución. Por último, sobre el identificador obtenido se ejecuta el comando "docker stop"deteniendo el contenedor, pasando del estado en ejecución al estado inactivo (el contenedor no es borrado, sigue existiendo pero si se quiere volver a usar, se debe iniciar de nuevo).

```
def stopDockerContainers(self):
  subprocess.run('deactivate & FOR /f "tokens=*" %i IN
      (\'docker ps -q\') DO docker stop %i', shell=True)
```

- Detener servidor: es necesario el identificador del proceso (no es posible obtenerlo desde Python, solo se obtendría el identificador de la consola desde la que se lanzó el comando para arrancar el servidor sin posibilidad de detener el proceso). Con el primer comando se obtienen los procesos

alojados en el puerto 8000 (donde se encuentra el servidor). Al lanzar el subproceso se ha indicado que el resultado se escriba en la variable "stdout" permitiendo su lectura. Se debe decodificar para que se interprete como texto. Una vez decodificado, se obtiene el primer identificador, el cuál es el asociado al servidor y se utiliza el método "*split*" para conseguir el valor y finalizar el proceso mediante el comando "*taskkill*".

```
def stopServerAuto(self):
        result = subprocess.run('netstat -ano | findstr
            :8000', stdout=subprocess.PIPE, shell=True)

        a = result.stdout.decode('utf-8')
        b = a.split("LISTENING", 1)[1]
        pidServer = b.split()[0]

        subprocess.run(f'taskkill /PID {pidServer} /F',
            stdout=subprocess.PIPE, shell=True)
```

4.4. Gestión de torneos

En esta sección se explicarán las herramientas creadas para crear y manejar los torneos.

4.4.1. Creación de un torneo

Se comienza creando un torneo y se registra en ElasticSearch. La plantilla web del maestro de ceremonias dispone de una sección con tres botones: el primero sirve para seleccionar el ejercicio en el que se basará el torneo, el segundo parámetro es el día límite que tienen los usuarios para registrarse y el último, el día que se celebrará el torneo (ver Figura 4.16).

Figura 4.16: Creación de un nuevo torneo

Es necesario guardar de modo permanente la información de todos y cada uno de los torneos, para ello se usará un documento de ElasticSearch donde se almacenarán los datos correspondientes. El proyecto cuenta con un archivo llamado `probe.py` donde se muestra la estructura de los documentos y se añadió una nueva clase de Python para los torneos con los siguientes campos:

- Identificador del torneo: número entero que sirve para identificarlo, funciona

como la clave primaria.

- Ejercicio: se almacena el nombre del ejercicio en el que consistirá el torneo.
- Fecha límite de inscripción: fecha hasta la que se admitirán nuevos registros de usuarios.
- Fecha de celebración: día que tendrá lugar el evento.
- Última modificación: campo que indica la última modificación que se realizó sobre el torneo y sirve para llevar un registro de las modificaciones en el documento.

Una vez creada la tabla donde almacenar los torneos, es posible gestionar la petición en el servidor, pero se necesita un identificador para cada uno, el cual se crea dependiendo del número de torneos registrados previamente en el documento de ElasticSearch. A continuación se introducen los datos necesarios (ejercicio, día del evento...) y se guardan.

4.4.2. Editar un torneo

Se creó la página web donde editar los torneos existentes para casos en los que el alumno hubiera olvidado registrarse y quisiera participar o errores en la creación del torneo o simplemente si se quiere cambiar alguno de sus datos. En esta vista el maestro de ceremonias visualiza todos los torneos registrados, se introduce el identificador del torneo y se modifican los campos del mismo (ver Figura 4.17).

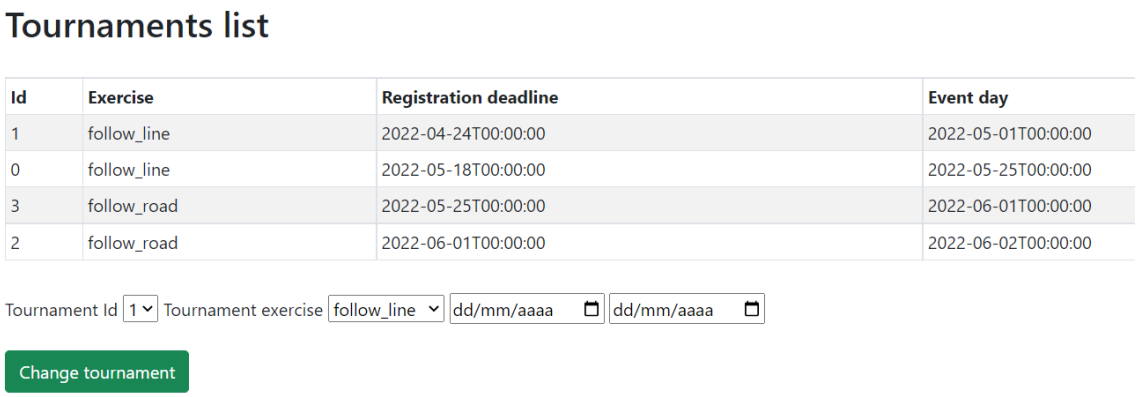

Figura 4.17: Vista para la edición de torneos

Para gestionar la petición en el servidor se debe acceder al documento de ElasticSearch donde se almacenan los torneos creados. Se realiza una consulta

del tipo "*match*" devolviendo todos los elementos que cumplan la condición. En este caso se utiliza el identificador del torneo sabiendo que funciona como clave primaria. Una vez obtenido se modifican los campos de los mismos.

4.4.3. Registro de un usuario en un torneo

Para gestionar los registros en un torneo se necesitará un documento de ElastiSearch, este documento se conforma de los siguientes campos:

- Identificador del torneo: identificador del evento en el que están registrados, funcionaría como una clave ajena referenciando al documento de torneos creado en apartados anteriores
- Nombre del usuario: nombre de la cuenta del usuario.
- Puntuación: puntuación obtenida en el torneo

Los usuarios cuentan con una vista donde pueden ver todos los torneos disponibles. Cada evento cuenta con un botón que nos dirige a los detalles del mismo (ver Figura 4.18). En esta página los usuarios podrán inscribirse en el torneo. Señalar que se ha considerado y protegido el caso donde un usuario se registra dos veces, mostrando un mensaje en pantalla.

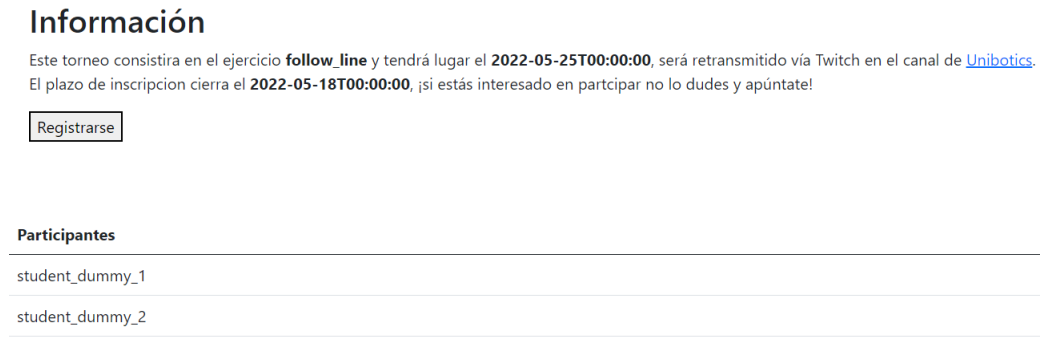

Figura 4.18: Vista para la inscripción en un torneo

Una vez que el servidor recibe la petición de registro en un torneo, se comprueba el plazo de inscripción. Si continúa abierto, se revisa el documento de ElasticSearch creado al comienzo de este apartado. Mediante una consulta tipo "*match*" se revisa si existe alguna entrada con el nombre del usuario y torneo al cual se desea registrar. En caso afirmativo implica que el usuario ya se encontraba registrado. Si no existe, se añade el usuario al documento y recibe un mensaje indicando que el registro ha sido realizado.

4.4.4. Retransmisión en directo

Durante un torneo el maestro de ceremonias dispone de un botón que indica que la retransmisión es en directo cambiando en el servidor la variable de emisión, de manera que cuando un usuario normal carga la página del torneo (Figura 4.19), se muestra completamente diferente. Primero aparece la emisión en directo de Twitch empotrada y en la parte inferior, la tabla con las puntuaciones de los usuarios que están participando.

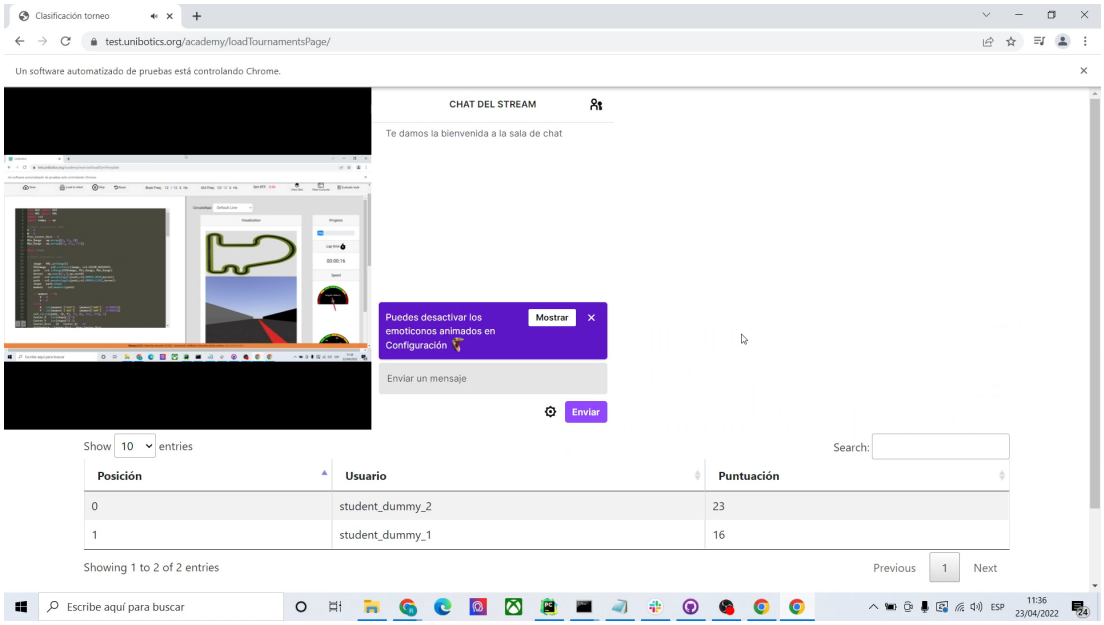

Figura 4.19: Retransmisión en directo en la página de Unibotics

Las puntuaciones se actualizan en tiempo real mediante el siguiente sistema:

- En el servidor existe una variable que aumenta cada vez que llega una nueva puntuación de un usuario.
- Los espectadores disponen de una función que se llama cada 30 segundos. Esta función envía un número indicando la última vez que se actualizó. Una vez llega al servidor, se comprueba si este valor es inferior a la variable del servidor, en caso afirmativo significa que debe actualizar su tabla de puntuación, en ese caso, recibe las nuevas puntuaciones de los usuarios, actualiza su tabla de puntuaciones y refresca el valor que indica la última vez que se actualizó.

Si la fecha de celebración del evento es inferior a la actual entonces cambia la vista del torneo, se muestra la tabla de puntuación con los resultados finales y un texto indicando que el torneo ya ha terminado (ver Figura 4.20):

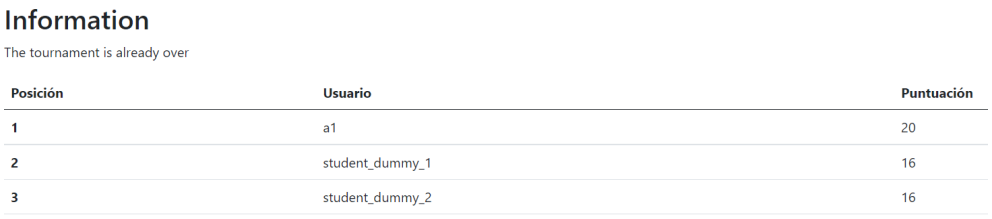

Figura 4.20: Vista de un torneo finalizado

4.5. Validación experimental

Durante el desarrollo del TFG se realizaron dos experimentos para la validación del software desarrollado y descrito en las secciones previas.

El primer experimento se llevo a cabo durante el mes de abril de 2022, se grabó un vídeo con el estado actual del TFG en ese momento. La finalidad era mostrar el funcionamiento automático de los torneos en un entorno local. Todavía no se tenía acceso a los códigos de los usuarios reales, se simuló un torneo con tres usuarios ficticios. El vídeo fue subido al canal de JdeRobot (organización a la que pertenece el proyecto Unibotics).

El segundo experimento se realizó en el mes de junio de 2022, en el despliegue en producción con tres usuario reales pertenecientes al grado de Robótica o al máster en Visión Artificial de la URJC. Por lo tanto, ya se accedió a los códigos fuente de los usuarios almacenados en la nube de Amazon. La finalidad de este experimento era la validación de todo el proyecto realizado en el entorno de producción. El vídeo también fue compartido en el canal de JdeRobot, existe otra versión sin editar explicándolo (vídeo). Las funcionalidades mostradas en el vídeo del entorno en producción son las vistas en el capitulo 4:

- Se comienza la transmisión en directo de manera automática en Twitch. Al lanzar el programa se inicia la conexión WebSocket con el programa OBS, mandando la orden de iniciar directo y a continuación, se finaliza la conexión ya que no se enviarán más peticiones (ver Figura 4.21).

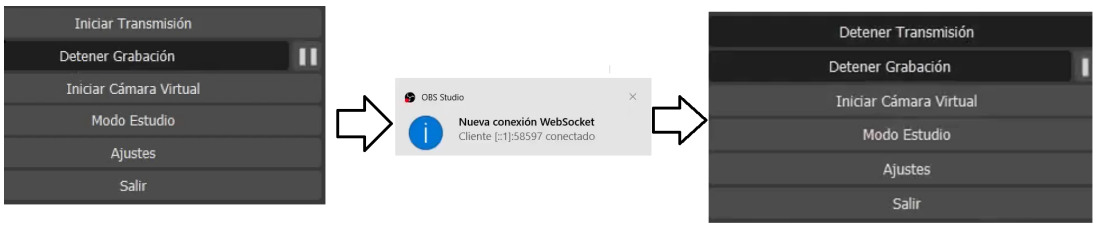

Figura 4.21: Inicio de emisión de directo OBS

- A continuación, se realiza el proceso de registro y selección del ejercicio en el que se basará el torneo (ver Figura 4.22).

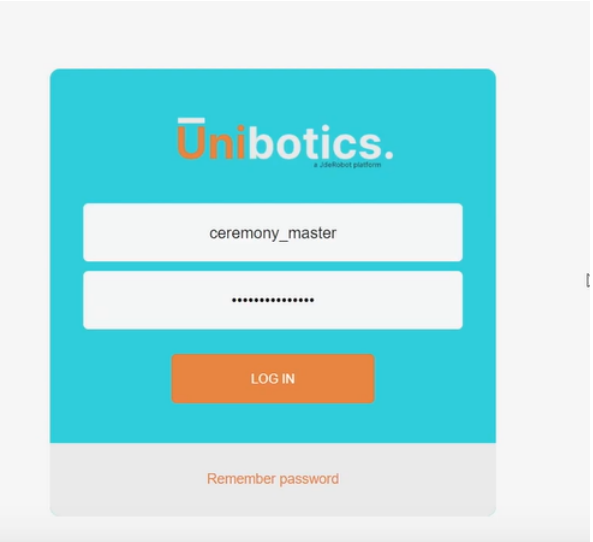

Figura 4.22: Inicio de sesión con el maestro de ceremonias en D3

- Se obtienen los usuarios que participarán en el torneo introduciendo el identificador del torneo, esto es necesario ya que Selenium se está ejecutando en un ordenador cliente que no tiene acceso a las bases de datos de ElasticSearch. Está función se ocultará en el futuro, se dejó visible únicamente para reflejar en el vídeo todos los pasos realizados por Selenium (ver Figura 4.23).

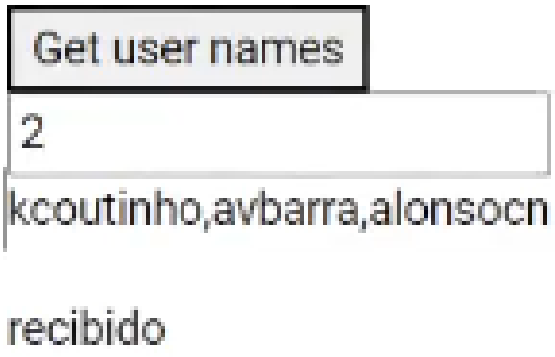

Figura 4.23: Obtención de concursantes del torneo

- A continuación se introduce el nombre del usuario para obtener su código fuente y cargarlo dentro del cerebro del robot (ver Figura 4.24).

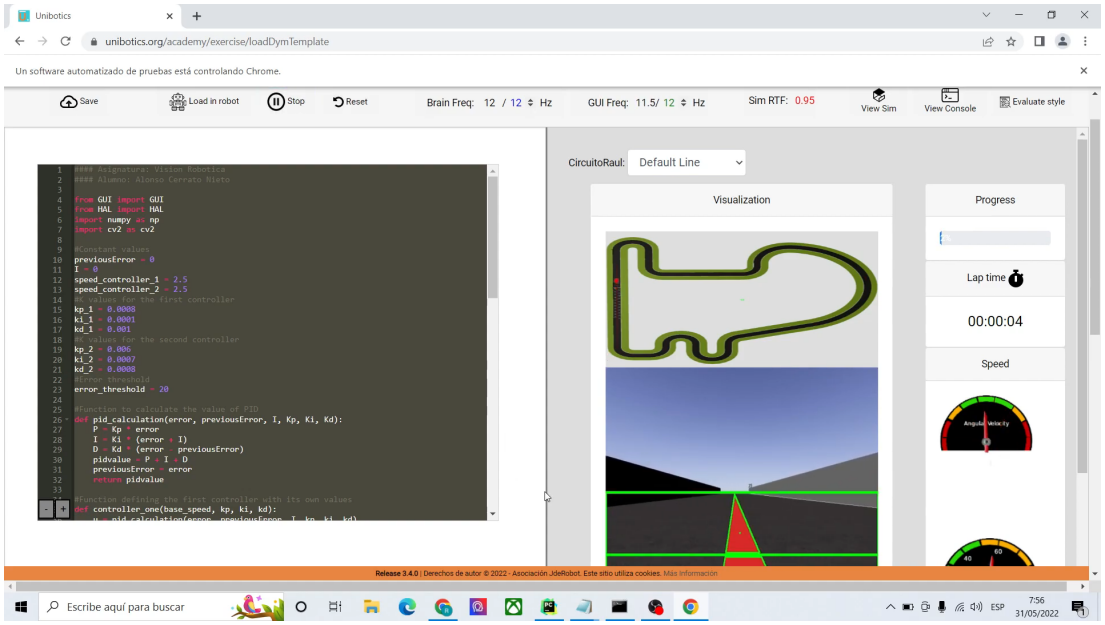

Figura 4.24: Ejecución del código fuente de un usuario

- Por último, una vez termina la ejecución del ejercicio se manda el resultado al servidor y se actualiza la página de puntuaciones (ver Figura 4.25).

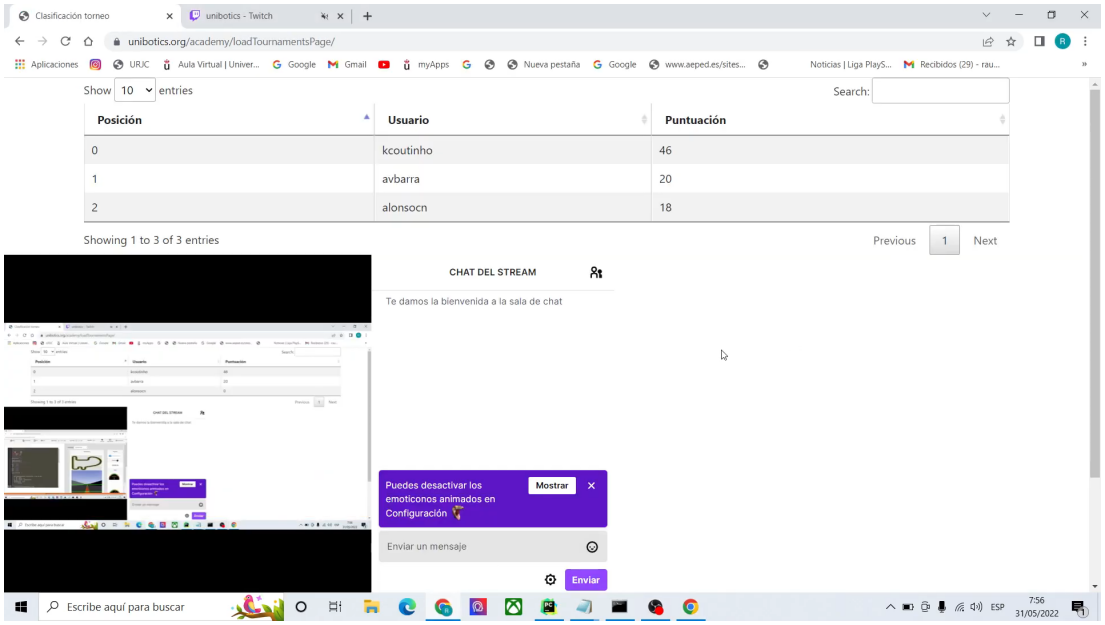

Figura 4.25: Página de retransmisión en vivo

5 Conclusiones

5.1. Conclusiones

Se ha conseguido satisfactoriamente el objetivo principal del TFG, que consistía en la creación y automatización de torneos en la página web Unibotics.

- **Realización de torneos**: el primer subobjetivo ha sido completado, para lograrlo se realizaron los cambios necesarios en los modelos Django y en las bases de datos para añadir un nuevo rol, el maestro de ceremonias. Además, se han implementado los cambios correspondientes en el lado del cliente y servidor, proporcionando al maestro de ceremonias vistas especiales para la gestión de los torneos, por ejemplo, obtener códigos fuente de otros usuarios e iniciar una emisión en vivo vía Twitch.

- **Automatización de los torneos**: el segundo subobjetivo se ha alcanzado con éxito, utilizando Selenium para la automatización de la página web y subprocesos de Python para el manejo de los contenedores Docker. Asimismo, se ha automatizado el control de la emisión en directo mediante el paso de mensajes vía WebSocket entre el programa OBS y el maestro de ceremonias.

- **Gestión de torneos**: el tercer subobjetivo se ha cumplido con el desarrollo de las plantillas Django, realizando las operaciones *CRUD* sobre los documentos de ElasticSearch y gestionando las peticiones entre cliente y servidor, dando lugar a la infraestructura necesaria para la implementación

de torneos en Unibotics, permitiendo a los usuarios registrarse en torneos, puntaciones en vivo, retransmisión embebida dentro de la página web, etc.

- **Validación**: el cuarto subobjetivo se ha logrado mediante la realización de un torneo de prueba en D3 con tres participantes. Se ha verificado el correcto funcionamiento de todo el software desarrollado para la realización los torneos en Unibotics.

A nivel personal se han adquirido conocimientos de gran variedad de tecnologías. Cuando comenzó el TFG no tenía ninguna experiencia con ellas, ni siquiera con el lenguaje Python. Las herramientas empleadas han sido: Python, Django, Selenium, ElasticSearch, HTML, JavaScript, CSS, subprocesos, Docker y OBS.

5.2. Trabajos futuros

Este trabajo puede ser ampliado de la siguiente manera:

- Extender a torneos con varias etapas como podrían ser cuartos, semifinal y final.
- Retransmitir en más plataformas durante las emisiones en directo como Youtube o Facebook Gaming.
- Crear clasificaciones globales según las posiciones que vayan obteniendo los usuarios en los torneos. A modo de ligas competiciones de una temporada que consten de varios torneos eventuales repartidos en el tiempo, de modo similar a como ocurre por ejemplo con las carreras de Formula1 reales, que forman parte de un único campeonato anual.
- Proporcionar estadísticas en los perfiles de los usuarios como ejercicios completados, número de torneos ganados, etc.

Bibliografía

[1] Parente, D. (2016). ludificación en la educación. ludificación en aulas universitarias, 11, 15.

[2] Robin Hunicke, Marc LeBlanc, and Robert Zubek. Mda: A formal approach to game design and game research. In Proceedings of the AAAI Workshop on Challenges in Game AI, volume 4, page 1722. San Jose, CA, 2004.

[3] Información sobre le estructura de ElasticSeacrh.https://www.knowi.com/blog/what-is-elastic-search/

[4] Documentación HTML y JavaScript. https://developer.mozilla.org

[5] Docker documentación oficial. https://docs.docker.com/

[6] Documentación oficial de Selenium. https://www.selenium.dev/documentation/

[7] Documentación oficial ElasticSearch. https://www.elastic.co/es/

[8] Documentación oficial de OBS. https://obsproject.com/es

[9] Documentación oficial Django. https://www.djangoproject.com

[10] Documentación oficial Python. https://docs.python.org/3/

[11] Riders.ai. https://riders.ai/about

[12] The Construct. https://www.theconstructsim.com/

[13] Funcionamiento WebSocket. https://sookocheff.com/post/networking/how-do-websockets-work/

[14] Información sobre la virtualizción. https://www.vmware.com/es/solutions.html

[15] Información sobre pseudo streaming. https://www.ontrack.com/es-es/blog/difusion-de-videos-en-la-red-streaming-o-pseudo-streaming

[16] Información sobre Python. https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

[17] Información sobre WebSockets. https://www.wallarm.com/what

[18] Información sobre ElasticSearch. https://www.davincigroup.es/introduccion-a-elasticsearch-que-es-casos-de-uso-instalar/

[19] Información sobre Selenium. https://inmediatum.com/blog/piensa-digital/que-es-selenium-y-para-que-sirve/

[20] Información Selenium Webdriver. https://www.browserstack.com/guide/selenium-webdriver-tutorial